ÎNVĂȚARE MULTI-AGENT

Universitatea din Craiova, Facultatea de Automatică, Calculatoare și Electronică

Clasificarea imaginilor folosind Multi-Layer Perceptron

Student : ***Ristea Andreea-Elena***

Grupa : ***IS1.1C***

Anul de studiu : *I*

Specializarea : ***Inginerie Software***

ABSTRACT

Acest document reprezintă o introducere în obiectivele de dezvoltare ale temei de cercetare pentru disciplina Învățare Multi-Agent.

Cuprins

Lista Figurilor

ACCESIBILITATEA LA CODUL SURSĂ

Codul sursă pentru simularea agentului pentru clasificarea imaginilor:

- Link GitHub:
  - ► https://github.com/AndreeaRistea/Classification-of-images-MLP.git.

Setul de date folosit:

- MNIST Dataset:
  - ► https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz.

1 Enunțul problemei

1.1 *Cerința*

Sa se implementeze un agent pentru clasificarea imaginilor folosind Multi-Layer Perceptron. Se va folosi pentru experimente setul de date MNIST pentru cifrele zecimale scrise de mana sau setul de date MNIST pentru articole de imbracaminte.
https://colab.research.google.com/drive/1JGUHtZnuVHarcyd4UWRPtOLxPTx80xsx?usp=sharing.

2 Problema studiată

Clasificarea imaginilor utilizând Multi-Layer Perceptron (MLP)

2.1 *Descrierea și înțelegerea problemei*

Clasificarea imaginilor este un domeniu esențial al învățării automate care implică procesarea și etichetarea imaginilor pe baza conținutului lor vizual. Această sarcină este întâlnită frecvent în aplicații precum recunoașterea scrisului de mână, analiza medicală a imaginilor și identificarea obiectelor în imagini.

Pentru a aborda această problemă, Multi-Layer Perceptron (MLP) reprezintă o arhitectură clasică de rețele neuronale artificiale. Acest tip de rețea funcționează prin procesarea datelor de intrare prin mai multe straturi complet conectate (fully connected layers), fiecare aplicând o transformare liniară urmată de o funcție de activare non-liniară. MLP este potrivit pentru probleme de clasificare, deoarece poate învăța relații complexe între datele de intrare și etichetele lor asociate.

Concepte cheie:

1. **Setul de date MNIST:**
   - MNIST este un set de date standard în domeniul învățării automate, utilizat pentru antrenarea și testarea algoritmilor de clasificare a imaginilor. Acesta conține:
     - *Cifre scrise de mână (MNIST clasic):* Imagini alb-negru de dimensiune 28x28 pixeli, fiecare reprezentând o cifră între 0 și 9.
     - *Fashion MNIST:* Un set de imagini similare ca format, dar care reprezintă articole de îmbrăcăminte, cum ar fi tricouri, pantaloni și încălțăminte.
2. **Preprocesarea datelor:**
   - Datele brute din imagini sunt normalizate prin împărțirea valorilor pixelilor la 255, astfel încât acestea să fie în intervalul $[0, 1]$. Aceasta accelerează procesul de antrenare și îmbunătățește performanța modelului.
3. **Arhitectura Multi-Layer Perceptron (MLP):**
   - *Stratul de intrare:* Imaginile de dimensiune 28×28 sunt transformate într-un vector unidimensional de 784 de elemente folosind un strat de aplatizare (`Flatten`).
   - *Straturi ascunse:*
     - Rețeaua include două straturi complet conectate (`Dense layers`), fiecare cu un număr de neuroni și funcții de activare ReLU (Rectified Linear Unit). Funcția ReLU introduce non-linearitate, permițând modelului să învețe relații complexe între datele de intrare și ieșire.
   - *Stratul de ieșire:* Este un strat complet conectat cu 10 neuroni (câte unul pentru fiecare clasă posibilă). Funcția de activare `softmax` convertește ieșirile într-un set de probabilități care indică probabilitatea ca o imagine să aparțină fiecărei clase.
4. **Optimizatorul:**
   - *Adam (Adaptive Moment Estimation):* Este un algoritm de optimizare avansat care combină avantajele descentului gradientului stocastic (SGD) cu momentele de adaptare. Adam ajustează dinamic ratele de învățare ale fiecărui parametru al modelului, asigurând o convergență mai rapidă și mai stabilă.
5. **Funcția de pierdere:**
   - *Sparse Categorical Crossentropy:* Măsoară diferența între distribuția probabilistică generată de model și etichetele reale. Aceasta este potrivită pentru probleme de clasificare multi-clasă în care etichetele sunt reprezentate sub formă de valori întregi.
6. **Măsura de performanță:**
   - *Acuratețea:* Reprezintă procentul de imagini clasificate corect de model pe un set de testare, fiind o metrică esențială pentru evaluarea performanței.

2.2 *Metoda de rezolvare*

Pentru a implementa soluția de clasificare a imaginilor cu MLP, am utilizat limbajul de programare Python, datorită ecosistemului său bogat de biblioteci și instrumente. Printre tehnologiile și resursele folosite se numără:

- **Biblioteci utilizate:**
  - *TensorFlow și Keras:* Pentru definirea, antrenarea și evaluarea modelului de rețea neuronală.
  - *NumPy:* Pentru manipularea eficientă a datelor și efectuarea operațiilor matematice necesare.
  - *Matplotlib:* Pentru vizualizarea rezultatelor, inclusiv grafice de pierdere și acuratețe.
- **Instrumente și medii de dezvoltare:**
  - *Google Colab:*
    - Un mediu bazat pe cloud care permite rularea codului Python fără a necesita instalări locale.
    - Oferă acces gratuit la GPU-uri și TPU-uri pentru antrenarea mai rapidă a modelelor.
    - Permite colaborarea în timp real, fiind ideal pentru proiecte educaționale și cercetare.
  - *PyCharm:*
    - Oferă o interfață bogată și funcționalități avansate precum completarea automată, depanarea integrată și analiza codului.
    - Suportă diverse tehnologii și dispune de o comunitate activă și documentație extinsă.
  - *Anaconda:*
    - Asigură gestionarea eficientă a pachetelor și mediilor prin intermediul Conda.
    - Include distribuții complete ale pachetelor Python, actualizări regulate și compatibilitate cu multiple sisteme de operare.

În ansamblu, utilizarea Python împreună cu Google Colab, PyCharm și Anaconda oferă un mediu integrat și eficient pentru dezvoltarea și testarea soluției de clasificare a imaginilor, maximizând productivitatea și facilitând gestionarea complexității proiectului.

3 Strategia de învățare

Am implementat o soluție pentru problema de clasificare a imaginilor folosind un Multi-Layer Perceptron (MLP). Modelul a fost antrenat pe setul de date MNIST pentru cifrele scrise de mână sau pe setul de date MNIST pentru articole de îmbrăcăminte, așa cum am menționat și în capitolul anterior.

3.1 *Învățare Supervizată și Rețele Neuronale*

Învățarea Supervizată este o ramură a învățării automate, în care modelul învață să facă predicții sau să clasifice datele pe baza unui set de date etichetat (în cazul meu, setul de date MNIST cu cifrele scrise de mână). Scopul este ca modelul să învețe o funcție care mapază datele de intrare (imaginile) la etichetele corecte (cifrele 0-9).

Rețelele Neurale sunt modele de învățare automată, inspirată de funcționarea creierului uman, care sunt utilizate pentru a rezolva probleme de clasificare, regresie, recunoaștere de imagini și multe altele. Acestea sunt formate dintr-o rețea de neuroni conectați, care procesează informațiile în mai multe straturi. Un Multi-Layer Perceptron (MLP) este un tip de rețea neuronală complet conectată, unde fiecare neuron dintr-un strat este conectat la fiecare neuron din stratul următor.

3.1.1 *Structura Rețelei Neuronale*

Rețelele neuronale sunt de obicei organizate în trei tipuri de straturi:

1. **Stratul de intrare (Input Layer)**: Este primul strat, care primește datele brute de intrare. Fiecare neuron din acest strat reprezintă o caracteristică a datelor de intrare (de exemplu, fiecare pixel dintr-o imagine).
2. **Straturile ascunse (Hidden Layers)**: Acestea sunt straturi intermediare între stratul de intrare și stratul de ieșire. Aceste straturi sunt denumite "ascunse" deoarece nu sunt direct observabile. În rețelele adânci (Deep Neural Networks - DNN), pot exista mai multe astfel de straturi, fiecare procesând informațiile și extrăgând caracteristici din datele brute.
3. **Stratul de ieșire (Output Layer)**: Acesta este ultimul strat, care produce rezultatele modelului. De exemplu, în cazul unui model de clasificare a imaginilor, acest strat va conține un neuron pentru fiecare clasă (cifră, obiect, etc.).

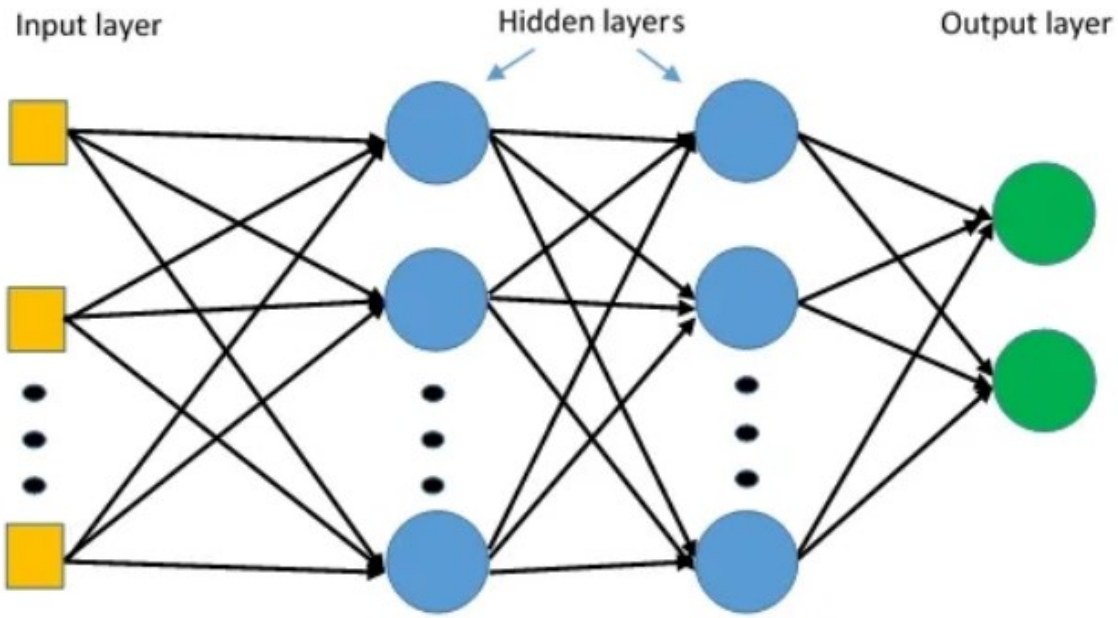

Figura 1: Rețea neuronală

3.1.2 *Neuroni și Funcții de Activare*

Un neuron în rețea primește un set de intrări (care sunt valorile de la neuronii din stratul anterior), le procesează și apoi produce o ieșire:

- **Intrările:** Sunt semnalele transmise de neuronii din stratul anterior.
- **Greutățile (Weights):** Coeficienții care determină importanța fiecărei intrări.
- **Bias-ul (Bias):** Este o valoare adăugată la suma ponderată a intrărilor, pentru a ajuta la ajustarea ieșirii.
- **Funcția de activare:** Determină activitatea neuronului, adică dacă acesta va trimite un semnal mai departe sau nu. Funcțiile de activare comune includ:
  - *ReLU (Rectified Linear Unit):* Folosită în straturile ascunse, ea permite doar valorilor pozitive să treacă mai departe (valoarea este trecută ca atare pentru valori pozitive și 0 pentru valorile negative).
  - *Sigmoid:* Produce valori între 0 și 1, fiind folosită mai ales pentru clasificarea binară.
  - *Softmax:* Folosită la stratul de ieșire pentru clasificarea multiclase, produce un vector de probabilități.

3.1.3 *Calculul Sumei Ponderate*

În cadrul unui neuron, fiecare intrare este multiplicată cu o greutate (*weight*) și rezultatele sunt adunate împreună. Apoi, se adaugă bias-ul și se aplică o funcție de activare pentru a obține ieșirea neuronului.

Matematic, pentru un neuron i cu:

- Intrările $x_1, x_2, \ldots, x_n$,
- Greutățile corespunzătoare $w_1, w_2, \ldots, w_n$,
- Bias-ul b,

ieșirea y_i este dată de:

$$y_i = \text{activare}\left(w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n + b\right).$$

Suma ponderată este:

$$S = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n + b.$$

Funcția de activare este aplicată pe suma ponderată pentru a obține ieșirea finală a neuronului:

$$y_i = \text{activare}(S).$$

3.1.4 *Greutăți și Actualizarea acestora*

Greutățile sunt parametrii pe care modelul le învață în timpul procesului de antrenament. În fiecare iterație, greutățile sunt ajustate pentru a minimiza diferența între predicțiile modelului și etichetele reale. Această diferență este cuantificată prin intermediul unei funcții de pierdere (*loss function*).

Ajustarea greutăților se realizează printr-un algoritm de optimizare, folosind:

- **Backpropagation:**
  - Este un algoritm care calculează gradientul funcției de pierdere (L) în raport cu fiecare greutate (w_i).
  - Gradientul este propagat de la stratul de ieșire către stratul de intrare.
- **Gradient Descent:**
  - Este o metodă de optimizare care ajustează greutățile w_i astfel încât să minimizeze funcția de pierdere.

- Formula de actualizare a greutăților este:

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i},$$

unde:

* η este rata de învățare (*learning rate*),
* $\frac{\partial L}{\partial w_i}$ este gradientul funcției de pierdere în raport cu greutatea w_i.

- **Algoritmul Adam:**
  - Este o variantă avansată a Gradient Descent, care:
    * Combină avantajele algoritmilor de tip *momentum* și adaptarea ratelor de învățare.
    * Ajustează dinamic ratele de învățare pentru fiecare greutate, folosind momentele primului și celui de-al doilea ordin ale gradientelor.
  - Formula generală de actualizare este:

$$w_i \leftarrow w_i - \frac{\eta \cdot m_t}{\sqrt{v_t} + \epsilon},$$

unde:

* m_t este media mobilă a gradientelor,
* v_t este media mobilă a pătratelor gradientelor,
* ϵ este o constantă mică pentru stabilitate numerică.

3.1.5 *Funcția de Pierdere (Loss Function)*

Funcția de pierdere măsoară cât de departe sunt predicțiile modelului față de etichetele reale. Scopul procesului de antrenament este să minimizeze această funcție. În funcție de tipul problemei, se utilizează diferite funcții de pierdere:

1. **Cross-Entropy Loss:** Folosită pentru probleme de clasificare multiclase, cum ar fi clasificarea cifrelor MNIST. Aceasta măsoară distanța între distribuția de probabilități prezisă de model ($\hat{y}$) și distribuția etichetelor reale (y).

   Formula este:

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i),$$

   unde:

   - C este numărul de clase,
   - y_i este eticheta reală (valoare binară: 1 pentru clasa corectă, 0 altfel),
   - $\hat{y}_i$ este probabilitatea prezisă pentru clasa i.

2. **Mean Squared Error (MSE):** Folosită pentru probleme de regresie, măsurând diferența pătratică între valorile prezise ($\hat{y}$) și cele reale (y).

   Formula este:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2,$$

   unde:

   - N este numărul total de exemple din setul de date,
   - y_i este valoarea reală,
   - $\hat{y}_i$ este valoarea prezisă de model.

3.1.6 *Antrenamentul Rețelei Neuronale*

În timpul antrenamentului, rețeaua neuronală procesează un lot de date (*batch*) și actualizează greutățile folosind procesul de *backpropagation* și optimizare. În fiecare epocă, rețeaua face un pas mai aproape de învățarea unei funcții care să mapeze datele de intrare la ieșirile corecte.

1. **Faza de propagare înainte (Forward Propagation):**
   - Datele de intrare trec prin rețea.
   - Ieșirea este calculată pe baza sumelor ponderate și a funcțiilor de activare.
2. **Calculul pierderii (Loss Calculation):**
   - Se compară ieșirile obținute cu etichetele reale.
   - Funcția de pierdere este calculată pentru a cuantifica diferența.
3. **Faza de propagare înapoi (Backward Propagation):**
   - Se calculează gradientele funcției de pierdere în raport cu fiecare greutate din rețea.
4. **Actualizarea greutăților:**
   - Greutățile sunt ajustate folosind un algoritm de optimizare (cum ar fi Gradient Descent sau Adam).
   - Scopul este reducerea valorii funcției de pierdere.

4 ABORDAREA FOLOSITĂ PENTRU PROBLEMA CLASIFICĂRII IMAGINILOR

Abordarea folosită pentru rezolvarea problemei tale implică implementarea unui model de clasificare a imaginilor folosind un Multi-Layer Perceptron (MLP), care este o rețea neurală artificială. Iată o explicație detaliată a pașilor și componentelor implicate:

1. **Preprocesarea datelor:**
   - **Încărcarea datelor:** Setul de date MNIST este încărcat folosind funcția `mnist.load_data()`, care furnizează imagini cu dimensiunea de 28x28 pixeli pentru fiecare cifră scrisă de mână (0-9).

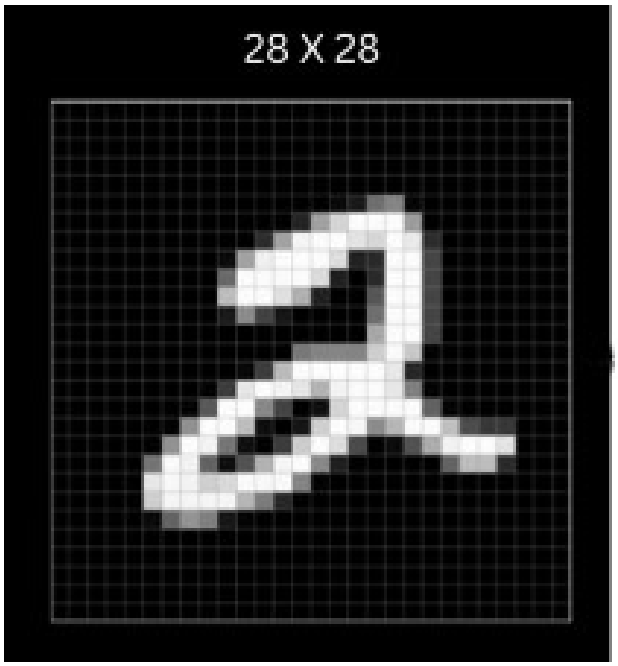

Figura 2: **Exemplu imagine din setul de date**

   - **Normalizarea imaginilor:** Imaginile sunt normalizate prin împărțirea valorilor pixelilor la 255.0, pentru a le aduce într-un interval de la 0 la 1. Acest lucru ajută la stabilitatea și performanța modelului.
2. **Crearea modelului MLP:**
   - **Modelul Sequential:** Modelul este creat folosind `Sequential`, care înseamnă că fiecare strat este conectat secvențial la următorul.

- **Stratul de Flatten:** Prima etapă a rețelei este un strat `Flatten`, care transformă matricea 28x28 (imaginea) într-un vector unidimensional de 784 de elemente. Acest pas este necesar pentru a face ca datele să poată fi procesate de straturile dense.
- **Straturi Dense:**
  - Primul strat dens are 128 de neuroni și utilizează funcția de activare `relu` (Rectified Linear Unit). Acest strat învață să extragă caracteristici din imagine.
  - Al doilea strat dens are 64 de neuroni și utilizează tot `relu`, pentru a adânci învățarea caracteristicilor.
  - Al treilea strat dens are 10 neuroni și utilizează funcția de activare `softmax`, care este utilizată pentru clasificarea multiclase (cifrele de la 0 la 9). Aceasta produce probabilități pentru fiecare dintre cele 10 clase.

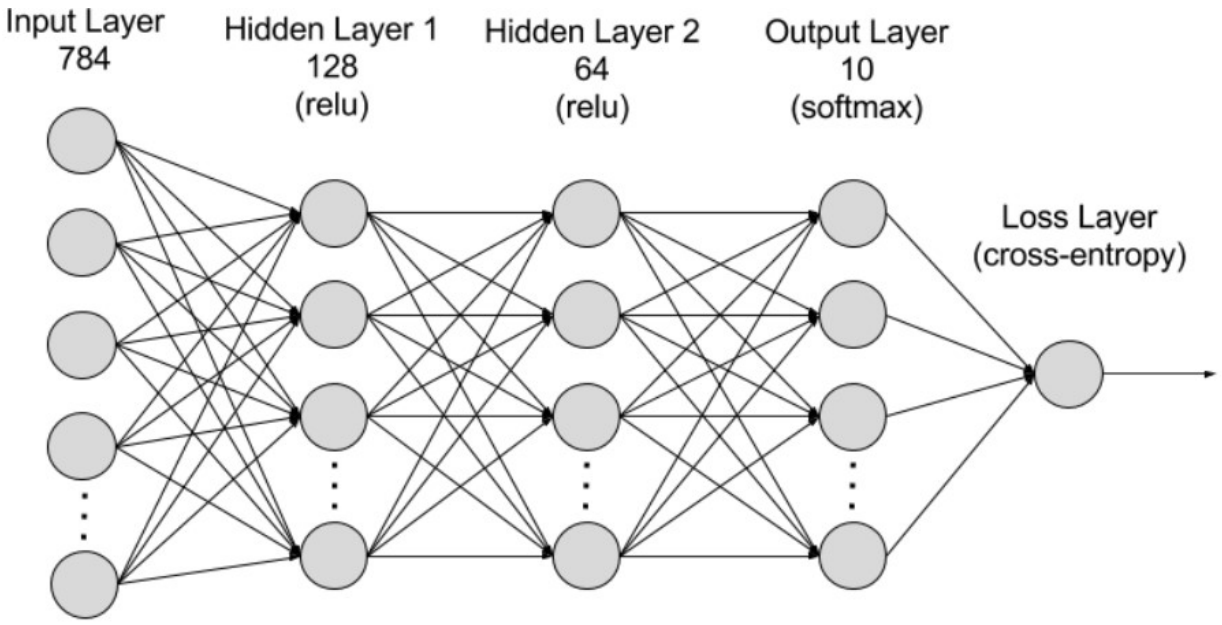

Figura 3: **Arhitectura de rețea MLP pentru MNIST**

3. **Compilarea modelului:**
   - **Optimizerul:** Se folosește optimizatorul `adam`, care este un algoritm eficient de optimizare pentru rețelele neurale.
   - **Funcția de pierdere:** Se folosește `sparse_categorical_crossentropy`, care este adecvată pentru clasificarea multiclase, unde etichetele sunt valori discrete (în cazul de față, cifrele 0-9).
   - **Metrici:** Acuratețea (`accuracy`) este aleasă ca metrică pentru a evalua performanța modelului pe seturile de date.
4. **Antrenarea modelului:**

- Modelul este antrenat pe setul de date de antrenament folosind metoda `fit()`. Aceasta va parcurge întregul set de date pentru un număr de 10 epoci (iterații prin setul de date).
- **Batch size:** Este setat la 32, ceea ce înseamnă că modelul va procesa câte 32 de imagini într-o singură iterație înainte de a actualiza greutățile.

5. **Evaluarea performanței:**
   - După antrenament, modelul este evaluat pe setul de testare folosind metoda `evaluate()`. Aceasta returnează pierderea și acuratețea modelului pe datele de test.
   - Acuratețea este calculată și afișată pentru a arăta cât de bine performează modelul pe datele care nu au fost folosite la antrenament.
6. **Vizualizarea predicțiilor:**
   - După evaluare, se fac predicții pe imagini din setul de testare folosind `predict()`. Apoi, sunt vizualizate primele 5 imagini, iar pentru fiecare imagine sunt afișate predicția (ce cifră a prezis modelul) și valoarea reală a etichetei.
   - Imaginile sunt afișate cu ajutorul bibliotecii `matplotlib`, iar etichetele și predicțiile sunt prezentate ca titluri.

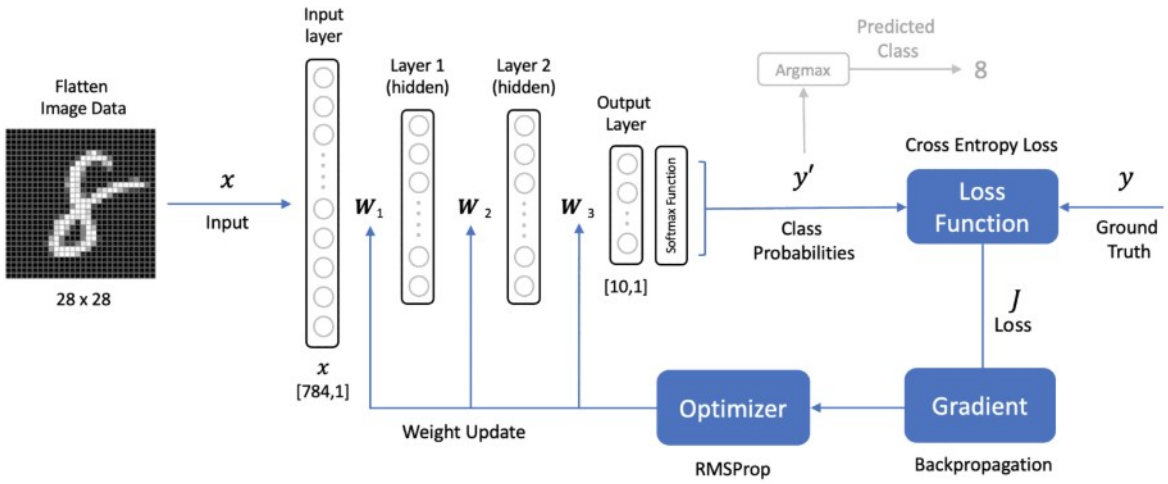

Figura 4: **Diagramă de implementare**

5 Analiza rezultatelor experminetale

5.1 *Analiza rezultatelor obtinute după testarea modelului pe setul MNIST*

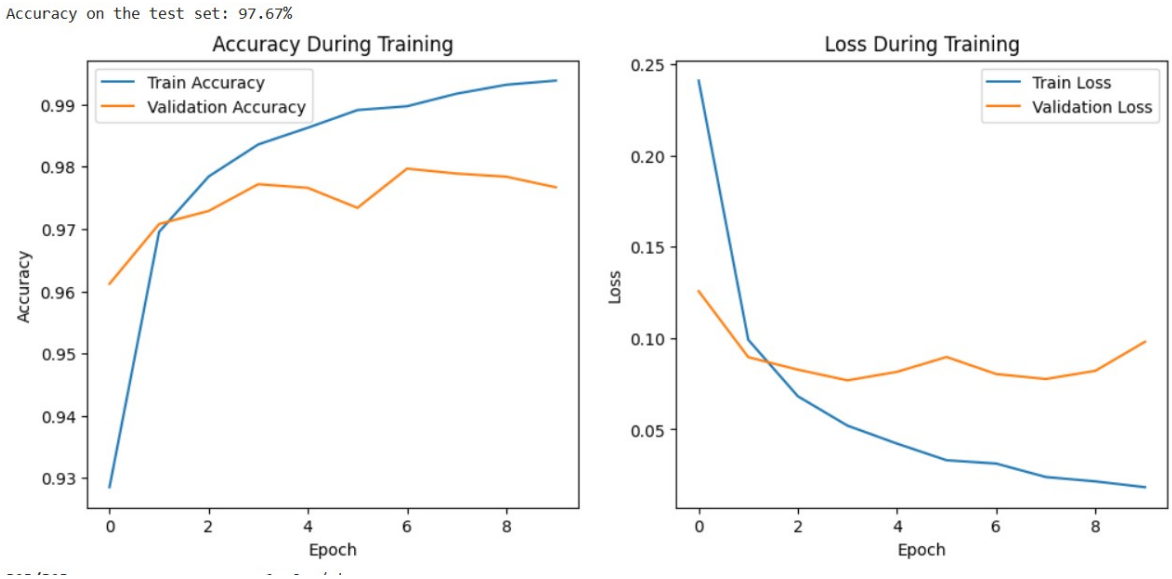

Figura 5: **Acuratețea și pierderea obținute pe setul de testare**

Modelul a obținut o acuratețe de 97.67% pe setul de testare, ceea ce indică o performanță foarte bună pentru o problemă de clasificare a cifrelor MNIST. Aceasta este aproape de maximul posibil pentru această arhitectură de rețea simplă, ceea ce sugerează că modelul a învățat eficient caracteristicile relevante din date.

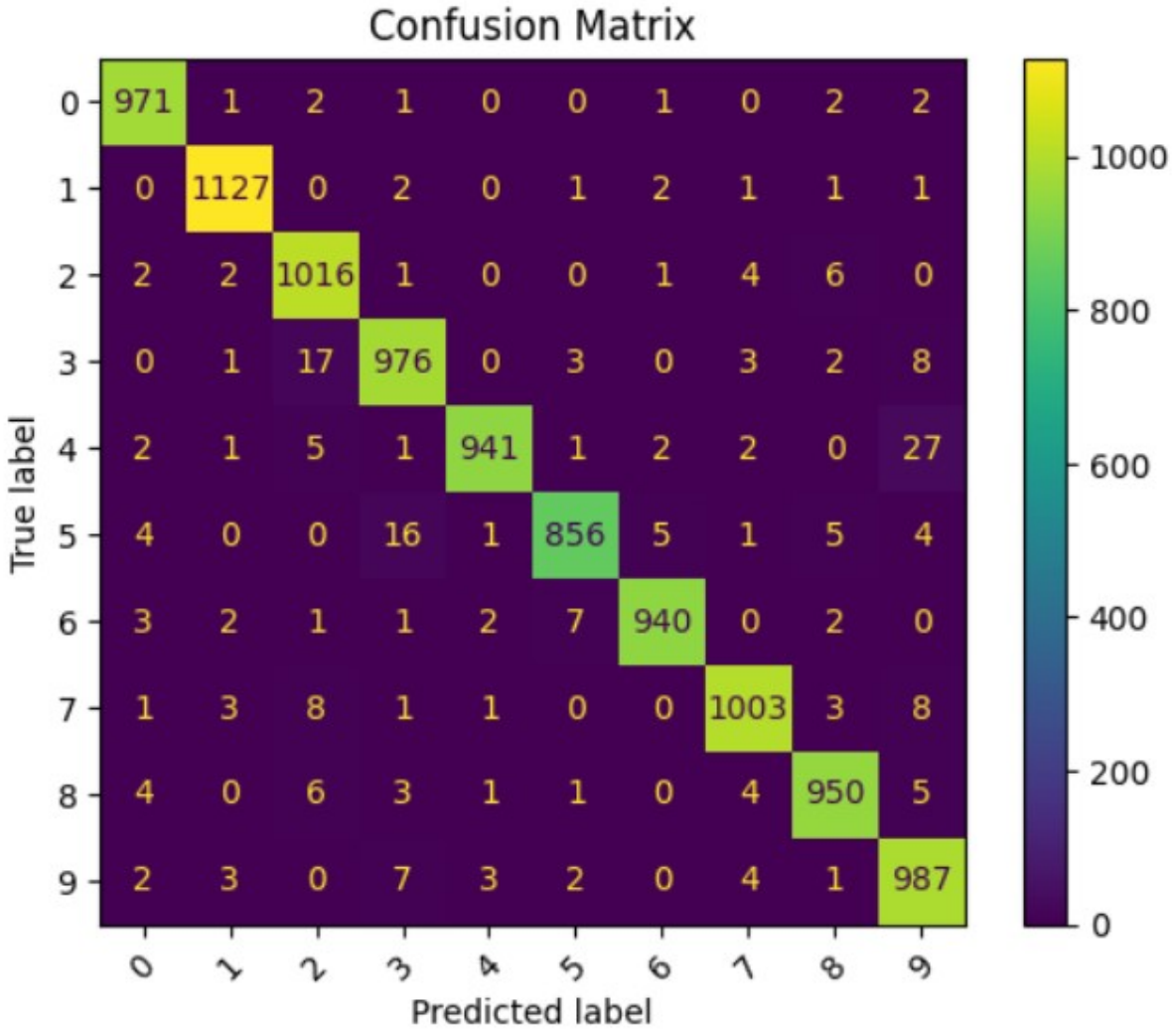

Figura 6: **Matricea de confuzie**

1. **Predicții corecte:** Majoritatea elementelor de pe diagonala matricei de confuzie sunt foarte mari, ceea ce indică o clasificare precisă pentru fiecare clasă.
2. **Erori comune:**
   - Clasa **5** este confundată uneori cu alte cifre, ceea ce sugerează că poate fi mai dificil de diferențiat (ex. cifre similare ca formă).
   - Clasele **8** și **3** au, de asemenea, câteva confuzii (ex. linia **8** și **3** ar putea arăta similar în scris de mână).

5.2 *Analiza rezultatelor obtinute după testarea modelului pe setul MNIST fashion*

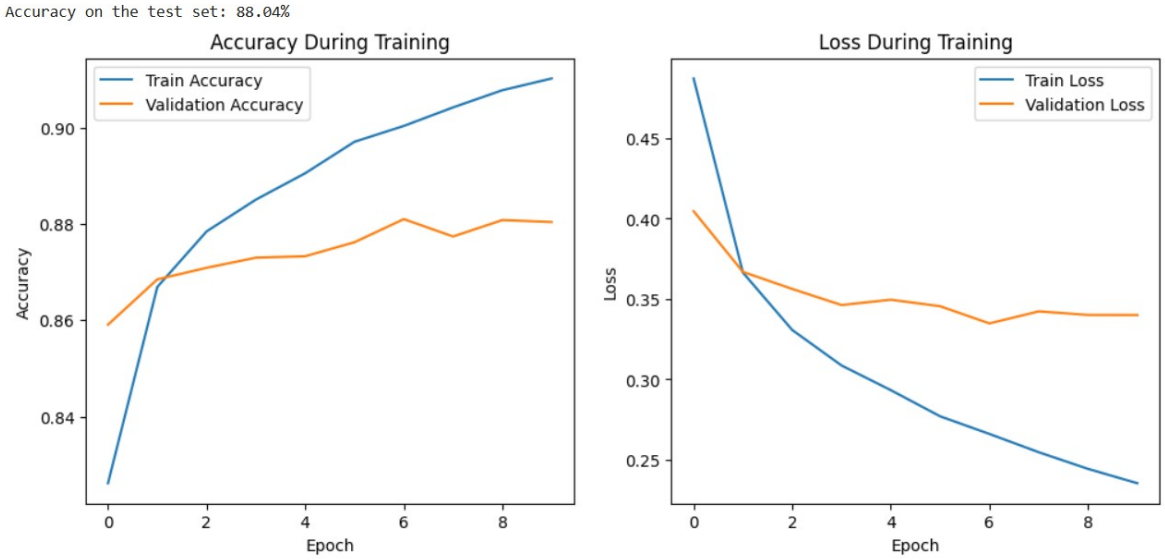

Figura 7: **Acuratețea și pierderea obținute pe setul de testare MNIST Fashion**

- Precizia pe setul de antrenare crește constant și ajunge aproape de 90%.
- Precizia pe setul de validare se stabilizează în jur de 88%, ceea ce sugerează o generalizare decentă, dar cu o ușoară diferență între antrenare și validare.
- Pierderile pe setul de antrenare scad continuu, ceea ce indică faptul că modelul învață bine pe datele de antrenare.
- Pierderile pe setul de validare scad inițial, dar se stabilizează ulterior, ceea ce sugerează că modelul începe să atingă un punct de saturație.

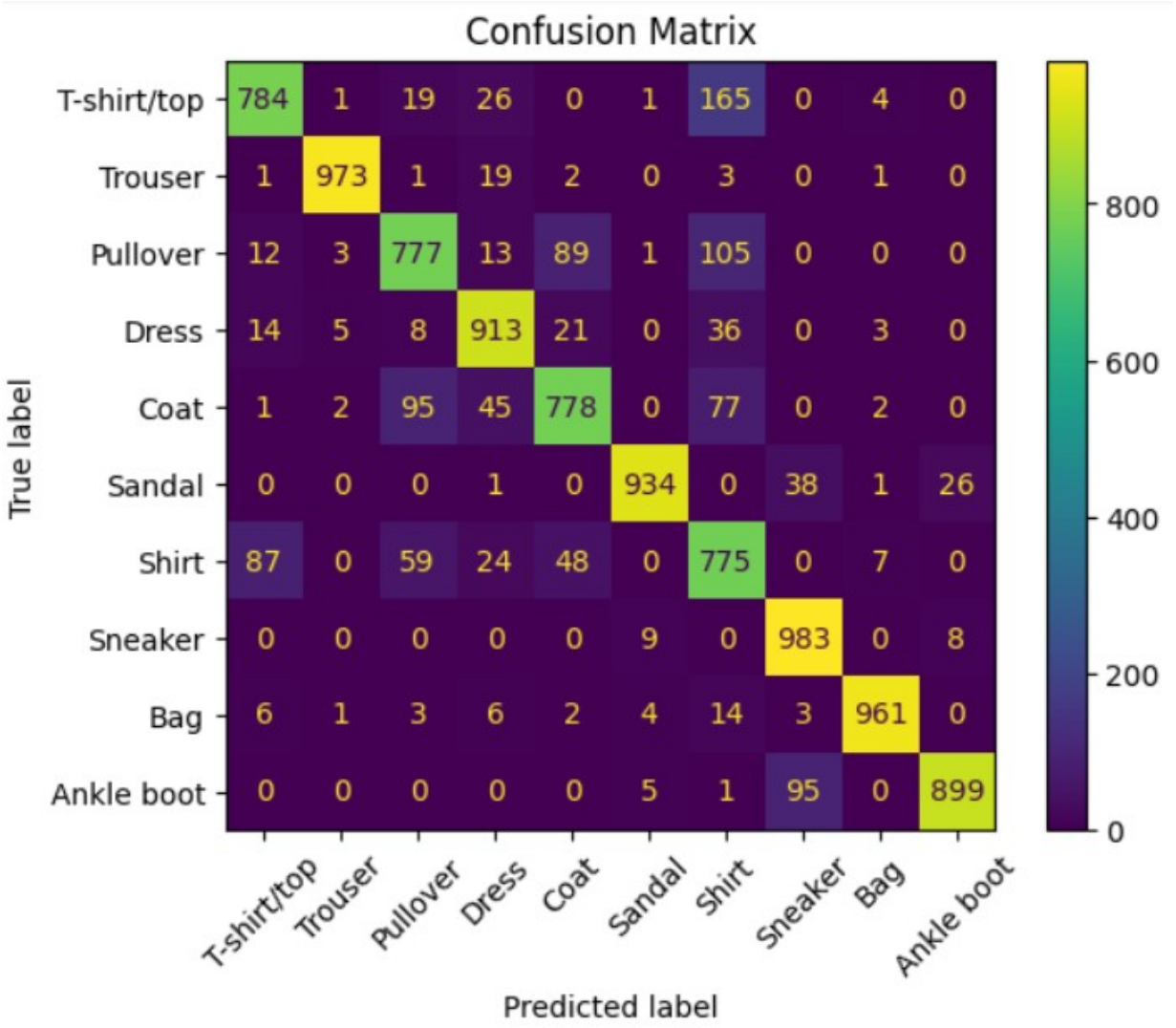

Figura 8: **Matricea de confuzie obținută pentru setul MNIST fashion**

- Diagonala principală are valori mari, ceea ce arată că modelul face majoritatea predicțiilor corecte pentru fiecare clasă.
- **Erori observate:**
  - Clasele Pullover și Shirt par să aibă mai multe confuzii (de exemplu, multe instanțe din clasa Pullover sunt clasificate greșit ca Coat sau Bag).
  - Unele clase, cum ar fi Sandal, au confuzii reduse, ceea ce indică o bună separare față de celelalte clase.

6 BIBLIOGRAFIE

Mai jos se pot observa câteva referințe bibliografice, documente, site-uri web, referințe ce au ajutat în parcurgerea, documentarea și înțelegerea mai amplă a temei.

Referințe Bibliografice

(1) https://colab.research.google.com/drive/1JGUHtZnuVHarcyd4UWRPtOLxPTx80xsx?usp=sharing#scrollTo=JLCsupbd1cZC.

(2) https://learnopencv.com/implementing-mlp-tensorflow-keras/.

(3) https://medium.com/@aungkyawmyint_26195/multi-layer-perceptron-mnist-pytorch-463f795b897a.

(4) https://mxnet.apache.org/versions/1.1.0/tutorials/python/mnist.html.

(5) https://www.youtube.com/watch?v=aircAruvnKk&ab_channel=3Blue1Brown.