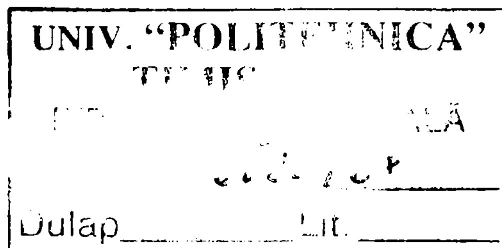
RECUNOAŞTEREA ŞIRURILOR NUMERICE SCRISE DE MÂNĂ

Teză destinată obţinerii
titlului ştiinţific de doctor inginer
la
Universitatea "Politehnica" din Timişoara
în domeniul ŞTIINŢA CALCULATOARELOR
de către

Ing. Dan Cireşan

Conducător ştiinţific: prof.univ.dr.ing. Ştefan Holban
Referenţi ştiinţifici: prof.univ.dr.ing. Ştefan-Gheorghe Pentiuc
prof.univ.dr.ing. Viorel Negru
prof.univ.dr.ing. Marius Crişan

Ziua susţinerii tezei: 17.10.2008



BUPT

Seriile Teze de doctorat ale UPT sunt:

1. Automatică
2. Chimie
3. Energetică
4. Ingineria Chimică
5. Inginerie Civilă
6. Inginerie Electrică
7. Inginerie Electronică şi Telecomunicaţii
8. Inginerie Industrială
9. Inginerie Mecanică
10. Ştiinţa Calculatoarelor
11. Ştiinţa şi Ingineria Materialelor

Universitatea „Politehnica" din Timişoara a iniţiat seriile de mai sus în scopul diseminării expertizei, cunoştinţelor şi rezultatelor cercetărilor întreprinse în cadrul şcolii doctorale a universităţii. Seriile conţin, potrivit H.B.Ex.S Nr. 14 / 14.07.2006, tezele de doctorat susţinute în universitate începând cu 1 octombrie 2006.

Copyright © Editura Politehnica - Timişoara, 2008

Această publicaţie este supusă prevederilor legii dreptului de autor. Multiplicarea acestei publicaţii, în mod integral sau în parte, traducerea, tipărirea, reutilizarea ilustraţiilor, expunerea, radiodifuzarea, reproducerea pe microfilme sau în orice altă formă este permisă numai cu respectarea prevederilor Legii române a dreptului de autor în vigoare şi permisiunea pentru utilizare obţinută în scris din partea Universităţii „Politehnica" din Timişoara. Toate încălcările acestor drepturi vor fi penalizate potrivit Legii române a drepturilor de autor.

România, 300159 Timişoara, Bd. Republicii 9,
tel. 0256 403823, fax. 0256 403221
e-mail: editura@edipol.upt.ro

BUPT

Cuvânt înainte

Teza de doctorat a fost elaborată pe parcursul activităţii mele în cadrul Departamentului de Calculatoare al Universităţii „Politehnica" din Timişoara.

Mulţumiri deosebite se cuvin conducătorului de doctorat, prof.dr.ing. Ştefan Holban, pentru înţelegerea şi îndrumarea pe care mi le-a oferit de-a lungul anilor de doctorat. Fără imboldurile sale în momentele dificile ale activităţii de doctorat, această teză nu ar mai fi fost finalizată vreodată.

De asemenea, as dori să mulţumesc prof. Augustin Devian, cel care, la vârsta marilor întrebări, mi-a arătat frumuseţea matematicii şi, implicit, a ştiinţei.

Mulţumesc mamei mele pentru că totul a început cu prima carte de popularizare a ştiinţei pe care mi-a dăruit-o. Au urmat apoi multe alte cărţi, fiecare mai frumoasă decât cealaltă. Pentru aceasta şi pentru toate lucrurile minunate pe care le-a făcut pentru mine va rămâne veşnic în amintirea mea.

Timişoara, octombrie 2008 Dan Cireşan

BUPT

Cireşan, Dan

Recunoaşterea şirurilor numerice scrise de mână

Teze de doctorat ale UPT, Seria 10, Nr. 10, Editura Politehnica, 2008, 106 pagini, 42 figuri, 13 tabele.

ISSN: 1842-7707

ISBN: 978-973-625-777-3

Cuvinte cheie: recunoaşterea scrisului de mână, reţele neuronale de convoluţie, GPU, optimizare, şiruri numerice, cifre parţial suprapuse, NIST SD 19, transformata Hough.

Rezumat,

Teza abordează problema recunoaşterii şirurilor numerice conţinând un număr nedefinit de cifre izolate sau parţial suprapuse. Metoda propusă nu utilizează segmentarea. Pentru a rezolva cazurile de cifre alipite sau parţial suprapuse, se foloseşte pe lângă clasificatorul uzual, la nivel de cifră, un clasificator antrenat cu perechi de cifre parţial suprapuse.

Sistemul creat atinge o rată de recunoaştere de 93.77%, în condiţiile în care nu foloseşte nici segmentare, nici antrenare cu exemple negative. Analiza erorilor arată că o viitoare antrenare cu exemple negative va duce la creşterea ratei de recunoaştere la 97-98%.

Au fost efectuate teste de performanţă folosind procesoare grafice (GPU). Rezultatele preliminare sunt promiţătoare, obţinându-se o accelerare de peste 20 de ori faţă de varianta scrisă pentru CPU.

BUPT

pentru Dana

BUPT

BUPT

Cuprins

1 Introducere **1**

1.1 Motivaţie 1

1.2 Organizarea tezei 3

1.3 Descrierea metodei 3

2 Stadiul actual al domeniului **5**

2.1 Stadiul actual al recunoaşterii cifrelor 6

2.2 Stadiul actual al recunoaşterii şirurilor numerice 7

2.2.1 Recunoaşterea holistică 7

2.2.2 Recunoaşterea cu segmentare 8

2.3 Concluzii 10

3 Aspecte teoretice **11**

3.1 Noţiuni de prelucrarea imaginilor 11

3.1.1 Centrul de masă al unei imagini 11

3.1.2 Distorsiuni 12

3.1.3 Transformata Hough 13

3.2 Reţele neuronale artificiale 14

3.2.1 Perceptronul 15

3.2.2 Reţele neuronale artificiale cu hărţi de convoluţie (RNC) 15

3.2.3 Metode de antrenare a reţelelor neuronale 16

4 Baze de date cu cifre şi şiruri de cifre **18**

4.1 NIST SD 19 19

4.1.1 Structura NIST SD 19 19

4.1.2 Conţinutul NIST SD 19 21

4.1.3 Ierarhizarea datelor 22

4.1.4 Formatul fişierelor NIST 23

4.1.5 Programe utilitare ale NIST SD 19 24

4.2 MNIST 24

4.3 LNIST - Liu-NIST 25

4.4 TDNS (Two-digit Numeral Strings) 26

4.4.1 Setul de antrenare 27

4.4.2 Setul de test 30

4.5 Seturile de test 3DNS, 6DNS şi 10DNS 30

4.5.1 Metoda de extragere a câmpurilor din formulare 31

BUPT

4.5.2 Setul de test 3DNS 36
4.5.3 Setul de test 6DNS 37
4.5.4 Setul de test 10DNS 38
4.6 Concluzii 38

5 Experimente **40**
5.1 Experimente cu cifre individuale 40
5.1.1 Reţele neuronale de tip perceptron 41
5.1.2 Reţele neuronale cu extractori de trăsături 41
5.2 Experimente cu şiruri compuse din două cifre parţial suprapuse 43
5.2.1 Reţele neuronale de tip perceptron 43
5.2.2 Reţele neuronale cu extractori de trăsături 44
5.2.3 Concluzii 49
5.3 Experimente cu şiruri de cifre din NIST SD 19 49
5.3.1 Sistemul de recunoaştere 49
5.3.2 Analizorul de componente conexe 49
5.3.3 Componenta de clusterizare (agregare) 50
5.3.4 Clasificatorul 52
5.4 Experimente pe setul 6DNS 57
5.5 Experimente pe setul 10DNS 58
5.6 Concluzii 58

6 Optimizarea vitezei de execuţie **60**
6.1 Variante CPU ale optimizării algoritmului de propagare înainte 61
6.1.1 Varianta open source modificată 61
6.1.2 Varianta C++ fară vectori STL 62
6.1.3 Varianta SSE2 63
6.2 Variante GPU ale optimizării algoritmului de propagare înainte 65
6.2.1 Varianta BROOK+ 68
6.2.2 Variantele CUDA 71
6.3 Concluzii 79

7 Concluzii **82**
7.1 Contribuţii 82
7.2 Sumarul tezei 83
7.3 Continuări posibile 85

BUPT

Lista figurilor

2.1 Ratele de recunoaştere pentru şiruri din NIST SD 19. Preluare din [44]. [35] reprezintă [30], [36] reprezintă [9], iar [14] reprezintă [58]. 10

3.1 Transformata Hough pentru detecţia de linii. Fiecărei linii d din spaţiul imagine îi corespunde un punct în spaţiul parametric Hough. 13

3.2 Exemplu de matrice de acumulare rezultată în urma aplicării transformatei Hough. Cercurile marchează maximele corespunzătoare celor şase linii detectate. Afişarea grafică foloseşte negru pentru elemente nule şi tonuri de gri cu iluminare proporţională cu valoarea respectivului element pentru valori nenule. Albul se foloseşte pentru elementul sau elementele cu număr maxim de puncte. 14

3.3 RNC pentru recunoaşterea caracterelor. Preluare din [37]. 15

3.4 Modul de deplasare a nucleului de convoluţie. 16

3.5 Modul de conectare a neuronilor de pe straturile de convoluţie. 17

4.1 Ierarhia seturilor de test şi de antrenare. 18

4.2 Exemplu de formular din NIST SD 19 (formularul f1816_24). A se vedea şi Figura 4.4. 20

4.3 Arborele aferent ierarhizării formularelor în partiţii. 23

4.4 Arborele aferent ierarhizării imaginilor de caractere după partiţii şi persoane. . 23

4.5 Comparaţie între centrarea în centrul de masă (stânga) şi centrarea în dreptunghiul delimitator (dreapta). Punctul roşu reprezintă centrul de masă al imaginii. Pe ultimul rând se află suprapunerea celor două imagini de pe coloana corespunzătoare. 26

4.6 Primul rând - exemple de artefacte în imagini 128×128 pixeli din NIST SD 19; rândul al doilea - imaginile rezultate în urma procesului de corecţie semiautomată. 27

4.7 Primul rând - imagini defectuos segmentate sau deformate din baza NIST SD 19; rândul al doilea - imagini rezultate în urma corecţiei manuale. 28

4.8 Primul rând - imagini 128×128 pixeli din NIST SD 19; rândul al doilea - imagini 20×20 pixeli rezultate după tăiere relativă la dreptunghiul delimitator, centrare şi redimensionare. 28

4.9 Interfaţa programului de corectare semiautomată. 29

4.10 Cifrele a nouă persoane după etapele de tăiere după dreptunghiul delimitator, centrare şi redimensionare la 20×20 pixeli. 30

BUPT

4.11 Procesul de suprapunere parţială: primul rând - imaginile originale; rândul al doilea - imagini deplasate pe verticală; al treilea rând - imaginea finală după deplasarea pe orizontală cu 13 pixeli (distanţa minimă dintre imagini) + 3 (distanţa de suprapunere) pixeli. 31
4.12 Limitarea variaţiei parametrului r. 33
4.13 Imagini obţinute în urma procesului de extragere a câmpurilor din formulare. Primul rând - imagini brute, al doilea rând - imagini după faza de curăţare automată, rândul al treilea - imagini după faza de curăţare manuală. 36
4.14 Imaginile cu numere de trei cifre care au fost scrise eronat de subiecţii din NIST SD 19. 37
4.15 Imagini cu numere de şase cifre, eronat scrise de subiecţii NIST SD 19. 37
4.16 Imagini cu numere de zece cifre, eronat scrise de subiecţii NIST SD 19 (toate ar trebui să conţină şirul 0123456789). 38

5.1 Exemple de imagini din fişierul de antrenare. 43
5.2 Arhitectura reţelei neuronale cu hărţi de convoluţie. 45
5.3 Suprafeţele erorilor reţelelor cu 8 hărţi de convoluţie pe stratul L_1. 48
5.4 Arhitectura sistemului de recunoaştere a şirurilor de cifre. 50
5.5 Componentele conexe înainte (sus) şi după (jos) faza de agregare. Fiecare componentă este colorată distinct şi este desenat şi dreptunghiul delimitator. . 51
5.6 Reconectarea cifrei 5. 51
5.7 Cazurile de unire defectuoasă de tipul $\overline{X1}$. 53

6.1 Codul sursă al algoritmului de propagare înainte pentru varianta C++ cu vectori STL. 62
6.2 Codul sursă al algoritmului de propagare înainte pentru varianta C++. 63
6.3 Codul sursă al algoritmului de propagare înainte pentru varianta C++ cu SSE II. 64
6.4 Evoluţia performanţelor procesoarelor şi a plăcilor grafice. Preluare după [54]. 66
6.5 Comparaţie între evoluţiile lăţimii de bandă pentru memoria sistem (RAM) şi pentru memoria globală a plăcilor grafice. Adaptare după [54]. 67
6.6 Comparaţie între alocarea tranzistoarelor pe suprafaţa de siliciu pentru GPU şi CPU. Preluare după [54]. 68
6.7 Codul sursă al algoritmului de propagare înainte pentru varianta BROOK+. . . 69
6.8 Arhitectura GPU-urilor de la NVIDIA (seriile GeForce 8 şi 200). Preluare după [65]. 71
6.9 Organizarea thread-urilor. Adaptare după [56]. 73
6.10 Kernel-urile pentru straturile de convoluţie. 75
6.11 Kernel-urile pentru straturile complet conectate. 76
6.12 Kernel-urile pentru stratul L3. 78
6.13 Gradul de accelerare a diferitelor variante de kernel pentru stratul L3. Referinţa este considerată varianta C++ optimizată, fără vectori STL. 80

BUPT

Lista tabelelor

4.1 Conţinutul şi modul de utilizare ale seturilor de date. 19
4.2 Versiuni anterioare ale bazei de date NIST cu imagini de caractere scrise de mână. (după [4]) 19
4.3 Bazele de date în care au fost publicate partiţiile. † - câmpuri completate, dar neprocesate. ø - câmpuri necompletate. (după [4]) 21
4.4 Exemplu de fişier cu referinţe (*hsf_page/truerefs/ref_24.txt*). A se vedea şi Figura 4.2. 22
4.5 Numărul de exemplare din fiecare clasă. 22

5.1 Rezultate obţinute cu perceptronul cu un strat ascuns. 41
5.2 Rezultate obţinute cu reţele neuronale cu hărţi de convoluţie. 42
5.3 Rezultate obţinute cu perceptronul cu un strat ascuns. 44
5.4 Rezultate obţinute cu reţele neuronale cu hărţi de convoluţie. 46
5.5 Influenţa strategiei de rejecţie asupra ratei de recunoaştere (forţat = fără rejecţie). 49
5.6 Rate de recunoaştere pentru setul de test 3DNS (1C - cu deplasarea RNC_1 deasupra imaginii, 2C - cu deplasarea RNC_2 deasupra imaginii, 1+2C - cu deplasarea ambelor RNC deasupra imaginii, simplu - fără deplasarea clasificatorilor). 54
5.7 Rate de recunoaştere pentru setul de test 6DNS (1C - cu deplasarea RNC_1 deasupra imaginii, 2C - cu deplasarea RNC_2 deasupra imaginii, 1+2C - cu deplasarea ambelor RNC deasupra imaginii, simplu - fără deplasarea clasificatorilor). 57
5.8 Rate de recunoaştere pentru setul de test 10DNS (1C - cu deplasarea RNC_1 deasupra imaginii, 2C - cu deplasarea RNC_2 deasupra imaginii, 1+2C - cu deplasarea ambelor RNC deasupra imaginii, simplu - fără deplasarea clasificatorilor). 58

BUPT

Capitolul 1

Introducere

1.1 Motivaţie

De ce recunoaşterea scrisului de mână? Deoarece scrisul a reprezentat, de la inventarea sa şi până în prezent, modalitatea cea mai facilă şi sigură de stocare şi de transmitere a informaţiilor. Din punct de vedere evolutiv, caracterul social al omului a fost generat de necesitatea creşterii şansei de supravieţuire. La rândul lui, caracterul social a favorizat prin selecţie apariţia mijloacelor de comunicare din care a emers limbajul în forma orală, pentru ca apoi să apară şi scrisul. Acesta reprezintă poate cea mai importantă invenţie a omului, dar trebuie să admitem că inventarea sa a fost constrânsă de evoluţie.

Scrisul a generat o revoluţie în civilizaţia umană prin cele două avantaje majore pe care le are faţă de comunicarea orală. Primul este capacitatea de stocare sigură a informaţiei, nealterată de transmiterea acesteia pe cale orală. Al doilea avantaj este reprezentat de faptul că permite transferul cunoştinţelor de la o generaţie la alta, facilitând acumularea acestora şi impulsionând progresul. Dacă scrisul nu ar fi existat, având în vedere lipsa memoriei genetice şi a duratei limitate de viaţă a oamenilor, atunci probabil că omul ar fi ramas într-un stadiu incipient de civilizaţie.

Alţi paşi importanţi în evoluţia scrisului au fost reprezentaţi de inventarea hârtiei şi a tiparniţei. Aceste două invenţii au făcut accesibilă informaţia unui număr mult mai mare de oameni, atât prin scăderea costului cărţilor, cât şi prin multiplicarea facilă a acestora. Inventarea calculatoarelor electronice în secolul trecut a generat o nouă revoluţie în ştiinţă, apărând noi modalităţi de stocare, transmitere şi prelucrare a informaţiei. Din acel moment, informaţia a început să fie scrisă în două forme: cea uzuală, utilizând hârtie, şi cea nouă, utilizând suporturi electronice şi optice. A devenit necesară conversia informaţiei dintr-un format într-altul. Dacă transformarea din format electronic în format tipărit este banală, conversia inversă s-a dovedit a fi atât complexă, cât şi complicată. Conversia unor cazuri particulare de scris, cum ar fi cel tipărit pe hârtie, este rezolvată acceptabil, în schimb conversia scrisului de mână, mai ales în forma cursivă, este într-o faza incipientă, departe de finalizare, rezultatele fiind, în general, mult sub necesităţile impuse de utilizarea practică.

Problema recunoaşterii scrisului de mână este unul din subiectele nerezolvate ale Inteligenţei Artificiale. Natura acestei probleme impune o abordare euristică în rezolvarea ei, astfel că multe din părţile procesului de recunoaştere sunt şi probabil vor rămâne perfectibile.

BUPT

În acest domeniu nu există soluţii definitive.

Teza abordează o problemă particulară a recunoaşterii scrisului de mână, cea a recunoaşterii şirurilor numerice alcătuite dintr-un număr nedefinit de cifre. Recunoaşterea şirurilor numerice prezintă atât avantaje, cât şi dezavantaje faţă de recunoaşterea textelor, propoziţiilor etc. Dintre avantaje se pot aminti varietatea mai mică a simbolurilor (nu există variante minuscule şi majuscule) precum şi numărul mai mic al acestora. În schimb, există un dezavantaj major: la recunoaşterea şirurilor numerice nu se pot aplica modele lingvistice [18].

Un exemplu de aplicaţie practică a recunoaşterii şirurilor numerice este recunoaşterea codurilor poştale de pe plicuri, a cecurilor bancare. Pe plan mondial există multe firme ce au ca domeniu de activitate producerea de echipamente pentru recunoaşterea scrisului. Una dintre cele mai importante, Parascript - Total Recognition Company, are în portofoliul de produse aplicaţii pentru domeniul financiar (recunoaşterea scrisului de pe cecuri, precum şi autentificarea de semnături), domeniul poştal (recunoaşterea adreselor), procesarea formularelor cu diverse tipuri de date. Faptul că produsele acestei firme sortează zilnic peste un sfert de miliard de plicuri în Statele Unite ale Americii demonstrează utilitatea şi necesitatea cercetărilor în acest domeniu.

Recunoaşterea şirurilor numerice este mult mai complicată decât cea a recunoaşterii cifrelor deoarece apar probleme suplimentare: lungimea necunoscută a şirului (spre deosebire de cifre sau litere [19], unde este unu, şi de cuvinte, unde poate fi extrasă din dicţionare), segmentarea, suprapuneri parţiale etc.

După cum se va vedea în capitolul următor, Starea actuală a domeniului, rata de recunoaştere actuală este de aproximativ 96%. Având în vedere că precizam anterior că zilnic peste un sfert de miliard de plicuri sunt sortate automat în SUA, 4% rată de eroare înseamnă extrem de mult, chiar dacă presupunem că 90% din plicuri sunt tipărite, iar în acest caz rata de recunoaştere este de aproape 100%. Ar rămâne aproximativ un milion de plicuri ce trebuie sortate manual, astfel că orice îmbunătăţire a ratei de recunoaştere este importantă.

La modul general, îmbunătăţirea ratei de recunoaştere este importantă în întreg domeniul recunoaşterii formelor şi la dezvoltarea inteligenţei artificiale.

Într-un domeniu matur din prisma numărului de metode care sunt folosite pentru rezolvarea sa, nu este loc de filozofie. Din acest motiv, această teză va adera cu stricteţe la cuvintele lui Sir William Thompson, Lord Kelvin (1824-1907): *"When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge of it is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced it to the stage of science."* dedicând cea mai mare parte a ei experimentelor şi comparării rezultatelor acestora.

Teza de doctorat a fost susţinută de grantul CNCSIS TD 2006-2007 nr. 58GR / 2006, cu titlul "Recunoaşterea textelor scrise de mână în limba română".

BUPT

1.2 Organizarea tezei

Această teză este organizată după cum urmează:

Capitolul 2. Capitolul următor descrie stadiul actual al domeniului, începând cu clasificatorii pentru cifre izolate şi finalizând cu clasificatorii pentru şiruri de cifre. În cazul celor din urmă se prezintă cele două abordări existente: holistică şi cea care implică segmentarea.

Capitolul 3. Acest capitol prezintă succint aspecte teoretice ce vor fi folosite în partea experimentală a tezei. Întrucât nu se doreşte încărcarea inutilă a lucrării cu informaţii care există prezentate în multe alte cărţi şi articole, aspectele teoretice vor fi doar schiţate şi apoi se vor oferi referinţe pentru detalii suplimentare. Prima parte a capitolului prezintă aspecte legate de procesarea imaginii, iar a doua oferă detalii despre arhitectura reţelelor neuronale utilizate.

Capitolul 4. Bazele de date cu cifre şi cu şiruri de cifre sunt detaliate în Capitolul 4. Sunt descrise modurile în care acestea sunt construite, relaţionarea şi alcătuirea lor. Se stabilesc seturile de date pentru antrenare şi pentru testare. Seturile sunt astfel construite încât să poată fi efectuată o comparaţie riguroasă şi sistematică cu rezultatele existente deja în domeniul recunoaşterii cifrelor şi a şiruruilor de cifre scrise de mână.

Capitolul 5. Folosind seturile de antrenare şi de test create, în Capitolul 5 sunt oferite rezultatele experimentelor efectuate. Întâi sunt antrenaţi şi testaţi clasificatorii pentru cifre izolate şi pentru perechi cifre parţial suprapuse, folosind două tipuri de reţele neuronale: perceptronul multistrat şi reţele neuronale cu hărţi de trăsături. Cei mai performanţi clasificatori sunt apoi folosiţi la crearea sistemului de recunoaştere pentru şiruri cu număr nedefinit de cifre. Sunt propuse şi evaluate trei metode de selecţie a clasificatorului corespunzător conţinutului imaginii: o cifră sau o pereche de cifre.

Capitolul 6. Ca urmare a concluziilor capitolului anterior, penultimul capitol are ca scop studierea accelerării antrenării reţelelor neuronale. Sunt studiate şase variante pentru două arhitecturi distincte. Primele trei sunt concepute pentru arhitectura x86, folosind diverse optimizări ale codului C++, precum şi aranjarea cât mai favorabilă a datelor în memorie. Una din variante apelează chiar la limbajul de asamblare şi la instrucţiunile SIMD SSE3. Următoarele trei sunt proiectate pentru arhitecturi masiv paralele ce folosesc capacităţile de calcul ale procesoarelor plăcilor grafice (GPU). Prima este dezvoltată în BROOK+ pentru plăci AMD-ATI, iar următoarele două - în CUDA pentru plăci NVIDIA.

Capitolul 7. Teza se încheie cu o analiză a rezultatelor şi cu evaluarea posibilelor dezvoltări ulterioare.

1.3 Descrierea metodei

Obiectivul tezei este de a oferi o metodă eficientă de recunoaştere a şirurilor numerice scrise de mână, compuse dintr-un număr nedefinit de cifre izolate sau parţial suprapuse. Întrucât marea majoritate a metodelor au ca pas anterior recunoaşterii un proces de

BUPT

segmentare care poate afecta negativ clasificarea, metoda pe care o propun nu utilizează segmentarea. Pentru a rezolva cazurile de cifre alipite sau parţial suprapuse am folosit pe lângă clasificatorul uzual, la nivel de cifră, un clasificator antrenat cu perechi de cifre parţial suprapuse.

În Figura 5.4 se poate observa calea parcursă de o imagine până când este recunoscut şirul numeric pe care îl conţine. Blocul **CCA** extrage componentele conexe ale imaginii, separând cifrele izolate şi toate celelalte părţi de cifre. Etapa de **clusterizare** se ocupă cu concatenarea părţilor de cifre extrase în faza precedentă. Am creat un set de reguli care sunt aplicate iterativ asupra componentelor extrase. La finalul etapei se obţin componente care reprezintă cifre şi grupuri de cifre unite.

Clasificatorul foloseşte rezultatele obţinute de cei doi clasificatori (pentru cifre şi pentru perechi de cifre parţial suprapuse) pentru a determina ce se află în imaginea componentei de la intrarea sa. Am implementat trei algoritmi simpli şi eficienţi care decid rezultatul căruia dintre cei doi clasificatori este corect. Algoritmii au un rol foarte important deoarece nu se cunoaşte a priori câte cifre se află în componenta analizată. Alte metode din literatură folosesc pentru această operaţie antrenarea clasificatorilor cu exemple negative, ceea ce este destul de dificil deoarece exemplele negative trebuie extrase manual sau generate automat. În plus, creşte dimensiunea setului de antrenare, ceea ce conduce la mărirea timpului necesar antrenării.

Întregul sistem de recunoaştere a fost testat cu seturi de imagini din NIST SD 19 [4]. Aceste seturile de imagini sunt folosite şi de alţi cercetători [44], cea mai bună rată de recunoaştere fiind de aproximativ 96%. Sistemul pe care l-am creat atinge o rată de recunoaştere de 93.77%, în condiţiile în care nu foloseşte nici segmentare, nici antrenare cu exemple negative. Analiza erorilor arată că dacă seturile de antrenare ar conţine şi exemple negative, rata de recunoaştere ar fi la nivelul celor mai bune rezultate existente. Din acest motiv, în viitor setul de antrenare va fi augumentat cu exemple negative. Rezultatele au fost publicate în [17].

Ultima parte a tezei este dedicată analizei posibilităţilor de a creşte viteza de execuţie a algoritmului de antrenare a reţelelor neuronale. Acest lucru este necesar deoarece ar permite antrenarea unor reţele neuronale mai mari, obţinându-se în acest mod rate de recunoaştere mai bune. Am încercat variante de optimizare pe CPU folosind inclusiv limbajul de asamblare şi instrucţiuni SSE3. De asemenea, am efectuat teste de performanţă folosind procesoare grafice (GPU) în BROOK+ şi CUDA. Rezultatele preliminare sunt promiţătoare, obţinându-se o accelerare de peste 20 de ori faţă de varianta scrisă pentru CPU.

Pentru că am evoluat ca specie de la homo habilis la homo sapiens sapiens, aceasta teză va încerca să prezinte cât mai mult din procentul de inspiraţie şi, de multe ori fără succes, cât mai puţin din restul de 99% de muncă istovitoare şi de multe ori plictisitoare.

BUPT

Capitolul 2

Stadiul actual al domeniului

Domeniul recunoaşterii scrisului de mână şi, în particular, cel al recunoaşterii cifrelor şi a şirurilor numerice, sunt cercetate intensiv de câteva decade. Din acest motiv, numărul de articole care tratează această problemă este destul de mare.

O căutare în IEEE Xplore returnează 608 articole ce conţin şirul "digit recognition". Numărul acestora creşte la 1304 dacă nu impunem prezenţa alăturată a celor doi termeni. Având în vedere că există articole care sunt indexate în alte baze de date cu articole, numărul lor este şi mai mare de atât.

Dacă efectuăm o căutare după "digit", "recognition", "numeral" (termeni care e foarte probabil să apară într-un articol având ca temă recunoaşterea şirurilor numerice), obţinem "numai" 60 de rezultate. Studiind anii în care au fost publicate aceste articole se observă că majoritatea au fost scrise în ultimii zece ani şi niciunul nu e mai vechi de 1982. Aceste observaţii arată că:

- recunoaşterea cifrelor este o problemă unde este foarte dificil de a mai propune metode noi care să fie mai eficiente decât cele deja existente (se va arăta în continuare că rata de recunoaştere a depăşit 99.6%)
- problema recunoaşterii cifrelor a fost abordată înaintea celei a recunoaşterii şirurilor de cifre. Evident, unul din motive a fost şi puterea scăzută de calcul a calculatoarelor
- în momentul în care recunoaşterea cifrelor a început să ajungă aproape de limita maximă de recunoaştere şi odată cu creşterea performanţelor calculatoarelor, atenţia a fost îndreptată asupra problemei mult mai complicată a recunoaşterii şirurilor numerice

Înainte de a prezenta cele mai importante realizări din domeniu se impune o explicaţie. Există multe articole care prezintă metode ce sunt verificate experimental pe seturi de test limitate (numărul de exemplare este prea redus pentru a avea însemnătate statistică) sau nedocumentate / private (deci imposibil de reprodus). Unele dintre aceste articole pretind rate de recunoaştere foarte mari, chiar de 100%. Întrucât nu există posibilitatea de a verifica rezultatele pretinse (decât, eventual, prin reimplementarea metodelor şi verificarea lor cu alte seturi de test), am decis să consider aceste rezultate ca având o relevanţă scăzută. Am considerat ca fiind relevante doar rezultatele articolelor care erau obţinute folosind seturi de antrenare şi de test accesibile (gratuit sau nu) şi care, mai important, erau folosite de cât mai mulţi alţi autori. Un alt mod de a aprecia însemnătatea rezultatelor unui articol a fost şi rangul conferinţei

BUPT

sau al jurnalului unde a fost prezentat / publicat respectivul articol. Sumarizând, rezultatele pe care le-am considerat a fi relevante au fost obţinute cu baze de date publice, folosite de cât mai mulţi cercetători din domeniu, şi au fost publicate la conferinţe renumite (ex: NIPS, ICDAR, CVPR, IPCV, AAAI, ICPR etc). Bineînţeles, acesta nu este un mod infailibil de a afla care sunt cele mai bune rezultate în domeniu, dar consider că este o soluţie cu probabilitate mare de a oferi rezultate relevante fără a examina exhaustiv toate articolele existente, ceea ce este practic imposibil (http://www.visionbib.com/bibliography/contentschar.html# Numbers, Digits, Zip (Postal) Codes).

Nefolosirea aceloraşi seturi de test de către toate articolele este o problemă ce afectează tot domeniul recunoaşterii scrisului de mână. Există câteva baze de date standard şi voi considera numai rezultatele obţinute folosind seturile de test din aceste baze de date. Două din aceste baze de date sunt disponibile contra cost: NIST SD 19 [4] şi CEDAR [22], iar una este oferită gratuit: MNIST [40].

2.1 Stadiul actual al recunoaşterii cifrelor

Întrucât este foarte puţin probabilă conceperea unui sistem de recunoaştere a şirurilor numerice ce nu încorporează un clasificator pentru cifre, vor fi prezentate cele mai importante realizări în acest domeniu. Experimentele sunt efectuate îndeosebi cu seturi extrase din NIST SD 19 [4] sau cu seturile MNIST [40]. Detalii despre aceste baze de date se găsesc în Capitolul 4.

Jie Zhou [70] propune un model de reţea neuronală cuantică. Este detaliat algoritmul de antrenare şi sunt oferite rezultatele experimentelor asupra unui set de 2000 de cifre din baza de date CENPARMI (http://www.cenparmi.concordia.ca/). Rata maximă de recunoaştere este de 95.65%.

Nishiwaki [53] propune o metodă de recunoaştere a şirurilor numerice care verifică existenţa cifrelor care se ating. Sunt propuse şase tipuri de atingere între cifre. Deşi aceaştă metodă verifică perechi de cifre, ea este dezvoltată în ideea de a fi folosită pentru recunoaşterea cifrelor izolate, în scopul rejectării imaginilor care conţin mai mult de un caracter. Nu sunt oferite date referitoare la ratele de recunoaştere.

Yann LeCun a creat baza de date MNIST ca subset al NIST SD 19 pentru a oferi un punct comun tuturor celor care doreau să-şi verifice metodele de recunoaştere. Faţă de NIST SD 19 care este foarte generală şi oferă date brute, lasând la latitudinea utilizatorului modul de alcătuire al seturilor de antrenare şi de test, MNIST propune două seturi, unul pentru antrenare şi unul pentru test, ce conţin 60000, respectiv 10000 de imagini de cifre. Ratele de eroare raportate pentru MNIST variază de la 12% pentru un perceptron fără straturi ascunse [38], deci un clasificator liniar, până la 0.39% pentru reţele neuronale de convoluţie [48].

Franke [27] prezintă o metodă de îmbunătăţire a clasificatorilor polinomiali prin învăţare iterativă. Rezultatul cel mai bun obţinut pe setul MNIST este de 1.13% rată de eroare.

Lauer et al. [36] a creat un sistem de recunoaştere ce foloşte vectori suport. Rezultatele sunt foarte bune (eroare 0.54%), dar sunt necesare resurse computaţionale mari.

BUPT

Cele mai multe încercări de rezolvare a acestei probleme folosesc reţele neuronale. Simard, Steinkraus şi Platt [64] demonstrează că reţelele neuronale oferă cea mai bună rată de recunoaştere pe setul de test MNIST. Rezultatele prezentate de LeCun, Simard şi Ranzato sunt la fel de bune, oferind o rată de eroare de 0.39% [60, 12]. Performanţe foarte bune au fost obţinute şi prin combinarea clasificatorilor [13].

Pe pagina MNIST (http://yann.lecun.com/exdb/mnist/) există câteva zeci de articole ale căror metode au fost validate cu seturile de date din MNIST.

Având în vedere rezultatele foarte bune obţinute pentru recunoaşterea de cifre cu reţele neuronale de convoluţie, am ales drept metodă pentru propriul clasificator la nivel de cifră utilizarea reţelelor neuronale de convoluţie (RNC).

2.2 Stadiul actual al recunoaşterii şirurilor numerice

Aşa cum arătam în Capitolul 1, recunoaşterea şirurilor numerice ce conţin cifre care se suprapun este foarte importantă deoarece este utilă într-un număr mare de aplicaţii practice: recunoaşterea codurilor poştale [24], procesarea documentelor complexe [34].

2.2.1 Recunoaşterea holistică

Recunoaşterea holistică încearcă să evite procesul de segmentare prin implementarea de clasificatori care primesc la intrare toată informaţia. Facând o paralelă cu recunoaşterea textelor, un clasificator holistic nu segmentează un cuvânt în litere urmând să recunoască literele, ci recunoaşte tot cuvântul dintr-odata. La recunoaşterea cuvintelor acest lucru este posibil deoarece numărul de cuvinte, chiar dacă este foarte mare, este limitat. În schimb, la recunoaşterea holistică a şirurilor numerice numărul claselor este mult mai mare, fiind imposibilă crearea unei metode de acest fel pentru cazul general. Din acest motiv clasificatorii holistici se rezumă la şiruri numerice de două, maxim trei cifre.

În [67] este descrisă o metodă de recunoaştere a cifrelor care se ating. Acest articol foloseşte ca seturi de antrenare şi de test imagini de coduri poştale provenite de pe corespondenţă. Nu se dă niciun alt detaliu asupra datelor folosite, deci experimentele nu pot fi verificate. Oricum, autorii pretind o rată de recunoaştere de 86.8% pe setul limitat pe care l-au folosit (291 de imagini de test). Sunt oferite şi rezultate (86.5%) pe un set neprecizat nici măcar ca număr de imagini din CEDAR [22]. Se afirmă că pe acelaşi set de test o metodă care implică segmentarea oferă rate de recunoaştere cu până la 6.1% mai mici. Clasificatorul utilizat analizează structura imaginilor, căutând concavităţi, măsurând gradienţi etc, informaţii cu care apoi construieşte un scor ce caracterizează imaginea. O observaţie interesantă a articolului este că din cele 500 de coduri poştale analizate, 20% conţin cifre unite, 16% fiind perechi de cifre, iar restul de 4% fiind alcătuite din trei sau mai mai multe cifre unite. În [35] aceiaşi autori prezintă un sistem multiexpert pentru recunoaşterea şirurilor numerice cu cifre care se ating. De această dată setul folosit este mult mai mare, 28464 de exemplare cu cifre unite fiind colectate manual. Şi acest set este privat. Cea mai bună rată de recunoaştere este de 91.1%.

Behnke [8] (Capitolul 7.4) a creat un sistem de recunoaştere pentru mărci poştale (timbre

BUPT

valorice imprimate pe plicuri) care utilizează un clasificator convoluţional pentru a recunoaşte perechi de cifre tipărite. Chiar dacă este o aplicaţie OCR (Optical Character Recognition = recunoaşterea caracterelor tipărite) şi nu a recunoaşterii scrisului de mână, am amintit şi acest sistem de recunoaştere deoarece abordează problema în mod holistic.

LeCun et al. [39] propun o metodă de recunoaştere a codurilor poştale folosind reţele neuronale de convoluţie. Ideea este dezvoltată mai departe de grupul său în [49, 50]. Pentru a evita segmentarea se construieşte o reţea neuronală extinsă pe orizontală în ideea de a încăpea în câmpul de intrare şi contextul cifrei ce se doreşte a fi recunoscută. Reţeaua este antrenată cu triplete de cifre suprapuse sau nu, generate automat. Acest clasificator este aplicat în fiecare poziţie a imaginii pentru a detecta cifrele. Deoarece se fac multe aplicări ale clasificatorului, metoda este destul de lentă (între timp puterea de calcul a crescut de câteva zeci de ori, aceast aspect ne mai constituind o problemă). Testele pe care le-au efectuat asupra codurilor poştale de cinci cifre au arătat o rată destul de redusă de recunoaştere: 70% pentru setul de antrenare şi 66% pentru cel de test.

2.2.2 Recunoaşterea cu segmentare

Multe din soluţiile propuse în literatura de specialitate combină procesele de segmentare şi recunoaştere. Având în vedere că multe cifre se pot atinge sau pot fi chiar parţial suprapuse, efectuarea unei segmentări corecte care să nu ţină cont de conţinutul imaginii este foarte dificilă [69]. Dacă se cunoaşte conţinutul imaginii, înseamnă că deja etapa de clasificare a avut loc.

Casey şi Lecolinet [11] prezintă o abordare care integrează segmentarea şi recunoaşterea folosind o strategie ipoteză-test (recunoaştere bazată pe segmentare). Cordella şi Vento [21] analizează în detaliu această metodă.

Shi, Wakabayashi et al. [47] propun o metodă de a creşte gradul de recunoaştere când numărul de imagini din setul de antrenare este redus. Algoritmul generează o imagine în oglindă a imaginii unei clase care aparţine unei perechi de clase foarte apropiate. Imaginea rezultată este adăugată setului de imagini al celeilalte clase.

Au fost implementate numeroase metode care folosesc segmentarea şi o extragere complicată de caracteristici pentru a obţine cifre din şirul iniţial [45]. Aceste cifre sunt recunoscute folosind un clasificator la nivel de cifră. Procesul de segmentare este aplicat în două moduri distincte [59]. În primul, modulul de segmentare transferă secvenţele de cifre izolate la clasificator [63]. Oliveira et al. [58] au rafinat acest concept propunând o verificare efectuată prin suprasegmentare şi prin subsegmentare. Aceiaşi autori au construit un sistem de recunoaştere folosind o metodă euristică de suprasegmentare bazată pe clasificatori cu vectori suport [57]. Al doilea mod utilizează ipoteze motivate probabilistic, iar decizia finală este dată de cel mai bun scor segmentare-recunoaştere obţinut pentru imaginea de intrare [29]. Acest procedeu este utilizat de Cheong et al. [14] pentru a genera mai multe ipoteze de segmentare bazate pe secvenţa de tuşe ce formează o posibilă cifră.

Autorii din [32] propun o metodă de preprocesare a segmentării cu scopul îmbunătăţirii ratei de recunoaştere pentru cifrele care se ating. Experimentele sunt efectuate pe NIST SD 19. Rata de recunoaştere este de 96.4%. Seturile de test sunt similare cu cele pe care le-am

BUPT

utilizat, dar mai mici, doar 1000 de imagini, în loc de aproape 1500 cât au seturile 3DNS şi 6DNS. Oricum, rezultatele pot fi folosite pentru comparaţii. Aceiaşi autori au dezvoltat o metodă de recunoaştere [33] care foloseşte segmentarea simplă şi segmentarea combinată cu recunoaşterea. Întâi (segmentare fără recunoaştere) sunt separate cifrele izolate de cifrele care se ating. Apoi sunt examinate şi segmentate cifrele care se ating. Se folosesc 1500 de imagini din NIST SD 19, dar nu se specifică indicii lor. Rata de recunoaştere este de 91.8%.

Cel mai important articol recent din domeniul recunoaşterii şirurilor numerice scrise de mână este [44]. A apărut în anul 2004 şi sumarizează cele mai performante metode apărute anterior. De asemenea, oferă o modalitate clară de comparare a rezultatelor, specificând cu exactitate provenienţa şi alcătuirea seturilor de test. Am reconstruit aceste seturi de test pentru propriile experimente (Secţiunile 4.5.2 şi 4.5.3). Autorii folosesc abordarea clasică a problemei: segmentare în cât mai multe părţi posibile, urmată de o concatenare ce are ca scop obţinerea de cifre singulare. Etapa de concatenare este combinată cu cea de recunoaştere, analizându-se toate modurile de concatenare. Se obţine un graf orientat, ce are două noduri importante între care se alfă toate modurile în care pot fi asamblate părţile de imagine. Pentru reducerea numărului de drumuri dintre cele două noduri se impun restricţii geometrice, cea mai importantă dintre ele fiind gradul de proximitate a două părţi de simboluri. Fiecare astfel de drum este apoi analizat şi în urma clasificării primeşte un scor obţinut ca sumă a scorurilor oferite de clasificatorul la nivel de cifră. Drumul cu cel mai mare scor este ales ca fiind cel mai probabil a fi corect. Existând mai multe drumuri, este uşor de implementat un algoritm de rejecţie care respinge un anumit rezultat ca probabil incorect în momentul în care diferenţa de scor dintre cele mai bune două drumuri este sub un anumit prag. În articol sunt analizate mai multe metode de clasificare pentru cifre: trei clasificatori neuronali, doi cu densitate discriminativă şi doi care folosesc vectori suport.

Un alt aspect important al articolului este **antrenarea cu exemple negative** (imagini care nu conţin cifre) a clasificatorilor pentru recunoaşterea de cifre. Se face o comparaţie între rezultatele obţinute utilizând seturi de antrenare ce conţin sau nu exemple negative. Prezenţa non-cifrelor în setul de antrenare îmbunătăţeşte rata de recunoaştere cu până la 30% pentru unele metode (reţele neuronale şi clasificatori cu vector suport).

La nivel de cifră, rata maximă de recunoaştere este de 99.26%, obţinută cu un clasificator cu vectori suport ce foloseşte funcţii radiale. Adăugarea de exemple negative în setul de antrenare produce o scădere a ratei de recunoaştere la 99.24%. La nivel de şir numeric, ratele maxime de recunoaştere se situează la 96.82% pentru şiruri de trei cifre şi la 96.74% pentru şiruri de şase cifre (Figura 2.1). Toate rezultatele sunt obţinute folosind clasificatori antrenaţi cu exemple negative. Acelaşi autor propune o metodă combinată de recunoaştere [43] folosind atât segmentarea, cât şi recunoaşterea holistică. Utilizând o metodă adaptată din recunoaşterea vorbirii, obţine o rată de recunoaştere de 97.42%.

Figura 2.1 sumarizează cele mai bune rezultate obţinute în domeniul recunoaşterii şirurilor de cifre scrise de mână. Coloana marcată *forced* conţine ratele de recunoaştere cu rejecţie nulă. RBF înseamnă funcţii radiale, PC - clasificator polinomial, DLQDF - clasificator cu funcţii discriminante cuadratice, bazat pe învăţare discriminativă, SVC-rbf - clasificator cu vectori suport şi funcţii radiale. Detalii despre implementarea clasificatorilor pot fi găsite în [44].

BUPT

Method	strlen	#string	Forced	1% err	0.5% err
[35]	3	986	92.7	79.5	70.5
	6	982	90.3	75.5	66.5
[36]	3	956	97.0	N/A	N/A
	6	961	96.7	N/A	N/A
[14]	3	2,385	95.38	90.60	89.81
	6	2,167	93.12	85.68	84.03
RBF	3	1,176	96.00	91.87	89.70
	6	1,471	96.13	91.98	88.72
PC	3	1,176	96.34	93.77	91.33
	6	1,471	96.16	93.20	91.71
DLQDF	3	1,476	96.54	93.36	88.21
	6	1,471	96.19	91.16	87.70
SVC-rbf	3	1,176	96.82	95.26	91.17
	6	1,471	96.74	94.15	91.64

Figura 2.1: Ratele de recunoaştere pentru şiruri din NIST SD 19. Preluare din [44]. [35] reprezintă [30], [36] reprezintă [9], iar [14] reprezintă [58].

2.3 Concluzii

Sistemul propriu de recunoaştere a şirurilor de numere conţine părţi care există şi în alte sisteme de recunoaştere. Mai exact, este vorba de clasificatorul la nivel de cifră. Întrucât ideea tezei nu este de a concepe un nou clasificator (eventual, mai bun) pentru cifre, am decis să folosesc cea mai bună metodă existentă pentru recunoaşterea de cifre în propriul sistem de recunoaştere a şirurilor numerice. Astfel, am ales reţelele neuronale de convoluţie. De asemenea, pentru acest clasificator am extins setul de antrenare folosind deformări afine şi elastice, aşa cum au făcut şi alţi autori (Simard [64] a introdus aceste deformări, aplicându-le pe setul MNIST).

Primul pas în construirea unui clasificator pentru recunoaşterea şirurilor numerice cu cifre ce se ating sau se suprapun parţial este reprezentat de rezolvarea cazului cel mai simplu de suprapunere, cel dintre două cifre.

Având în vedere că majoritatea cercetătorilor folosesc metode ce implică segmentarea şi că ratele de recunoaştere sunt deja foarte mari (peste 96%), iar recunoaşterea holistică este mult mai puţin studiată, am hotărât să aplic o metodă holistică pentru perechile de cifre parţial suprapuse, în ideea evitării segmentării.

Deoarece reţelele neuronale de convoluţie s-au dovedit a fi foarte bune pentru recunoaşterea de cifre, le-am folosit şi pentru construirea unui clasificator la nivel de perechi de cifre parţial suprapuse. Reţelele neuronale s-au dovedit a fi eficiente în foarte multe domenii, fiind şi foarte rapide în procesul de clasificare, aspect care este important în aplicaţii ce trebuie să ruleze în timp real sau cât mai repede posibil (de exemplu un sortator de corespondenţă).

BUPT

Capitolul 3

Aspecte teoretice

3.1 Noţiuni de prelucrarea imaginilor

În această teză, imaginile vor fi abstractizate în felul următor: o colecţie de pixeli structuraţi într-un tabel bidimensional, accesibili prin indicele liniei (coordonata y) şi cel al coloanei (ordonata x). Indicii sunt valori întregi nenegative. Fiecare pixel are o anumită culoare, pentru experimentele efectuate fiind suficient a considera cazul tonurilor de gri cu 256 valori discrete, 0 însemnând alb şi 255 - negru. Dacă pixelii pot avea doar două valori, atunci 0 repezintă alb şi 1 - negru. În cazul în care se face excepţie de la aceste reguli, se precizează în text. Pentru imagini color se aplică o transformare în tonuri de gri folosind, de exemplu, metoda prezentată în [15] la pagina 9 sau în [16].

3.1.1 Centrul de masă al unei imagini

Din fizică se cunoaşte că pentru sistem de n particule punctiforme, fiecare fiind caracterizată prin masa m şi prin coordonatele x_i într-un spaţiu euclidian p-dimensional ($i \in \{1, \ldots, p\}$), coordonatele centrului de masă pot fi calculate cu formula:

$$x_i^{CM} = \frac{\sum_{j=1}^{n} m_j \cdot x_i^j}{\sum_{j=1}^{n} m_j} \tag{3.1}$$

În cazul unei imagini bidimensionale cu $n \times m$ pixeli (n linii şi m coloane), intensitatea fiecărui pixel fiind reprezentată prin p_{xy}, cu $x \in \{0, \ldots, m-1\}$ şi $y \in \{0, \ldots, n-1\}$, putem adapta formula de calcul a coordonatelor centrului de masă în felul următor:

$$x^{CM} = \frac{\sum_{x=0}^{m-1} \sum_{y=0}^{n-1} p_{xy} \cdot x}{\sum_{x=0}^{m-1} \sum_{y=0}^{n-1} p_{xy}} \tag{3.2}$$

$$y^{CM} = \frac{\sum_{x=0}^{m-1} \sum_{y=0}^{n-1} p_{xy} \cdot y}{\sum_{x=0}^{m-1} \sum_{y=0}^{n-1} p_{xy}} \tag{3.3}$$

BUPT

3.1.2 Distorsiuni

În experimentele efectuate cu imagini de cifre am folosit augumentarea setului de antrenare prin aplicarea de distorsiuni asupra imaginilor sale. Transformările efectuate sunt similare cu cele aplicate de Simard [64]. Lauer et al. [36] detaliază aceste transformări. Distorsiunile au constat în mai multe transformări liniare: rotaţii, scalări şi deformări elastice.

Transformări afine

Intuitiv, transformările afine într-un spaţiu euclidian bidimensional pot fi privite ca transformări geometrice în urma cărora se păstrează coliniaritatea punctelor şi proporţiile segmentelor.

Transformările afine aplicate asupra imaginilor de antrenare sunt de două feluri: rotaţii şi scalări. Se calculează câte un câmp de deplasare pentru fiecare direcţie: orizontală (ΔX) şi verticală (ΔY). Astfel, pentru punctul de coordonate (x, y) vom avea deplasările $\Delta X(x, y)$ şi $\Delta Y(x, y)$.

Scalările pot fi reprezentate matricial în felul următor:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \alpha & 0 \\ 0 & \beta \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.4}$$

unde α reprezintă coeficientul de scalare pe orizontală şi β reprezintă coeficientul de scalare pe verticală.

Rotaţiile în formă matricială se calculează cu formula:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\gamma & \sin\gamma \\ \cos\gamma & -\sin\gamma \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.5}$$

unde γ reprezintă unghiul de rotaţie a imaginii.

Transformări elastice

În [64] transformările elastice sunt obţinute prin prelucrarea unor câmpuri de deplasare aleatoare. Astfel, se iniţializează un câmp de deplasare cu valori reale aleatoare din $[-1, 1]$. Distribuţia valorilor se consideră a fi uniformă. Asupra acestui câmp se aplică o convoluţie cu un nucleu gaussian cu deviaţia standard σ.

Pentru două dimensiuni, nucleul gaussian normalizat are următoarea formulă:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\pi\sigma^2}} \tag{3.6}$$

Lăţimea nucleului este dată de valoarea lui σ.

Rezultatul acestor operaţii este un câmp de deplasare care aplicat unei imagini produce o deformare elastică în cazul în care σ are valori mici. Dacă σ are valori mari, deformarea devine transformare afină, iar pentru σ foarte mare se ajunge la translaţie [64]. Se foloseşte un coeficient α cu care sunt multiplicate valorile nucleului pentru a controla intensitatea locală a deformării.

BUPT

3.1.3 Transformata Hough

Transformata Hough a fost inventată de Paul Hough în 1962 şi patentată de IBM. Utilizarea iniţială a constat în detectatea traiectoriilor particulelor încărcate electric din experimentele efectuate în camera cu bule. Richard Duda şi Peter Hart ([23]) au creat transformata Hough generalizată în 1972. Dacă metoda iniţială detecta numai linii, transformata generalizată poate detecta orice fel de forme [62] (pag. 382-383), în particular segmente de dreaptă, arce de cerc şi de elipsă.

Se va prezenta succint transformata Hough pentru detectarea de linii dintr-o imagine. Problema poate fi formalizată astfel: fiind dată o mulţime de puncte prin coordonatele x şi y dintr-un spaţiu euclidian bidimensional, se cere să se determine care din aceste puncte sunt coliniare şi care nu. Pentru aceasta, se aplică o transformare a spaţiului euclidian la spaţiul Hough, utilizând ecuaţia 3.7.

$$r = r_1 + r_2 = \frac{y}{\cos\theta} + \sin\theta\,(y - \tan\theta) = x\sin\theta + y\cos\theta \tag{3.7}$$

În ecuaţia de mai sus, θ reprezintă unghiul dreptei cu axa Ox, iar r este distanţa de la originea sistemului de coordonate la dreaptă (Figura 3.1). Deoarece pentru fiecare punct există o infinitate de perechi în spaţiul parametric (r, θ), trebuie găsită o modalitate de reducere a dimensionalităţii problemei pentru a putea fi aplicată practic. Pentru aceasta, se discretizează problema prin discretizarea valorilor parametrilor. Astfel, dacă imaginea iniţială, în forma discretă, avea $N \times M$ pixeli, putem discretiza spaţiul parametrilor alegând, de exemplu, $\Delta r = 1pixel$ şi $\Delta\theta = 1^\circ$. Discretizarea se face în funcţie de problemă.

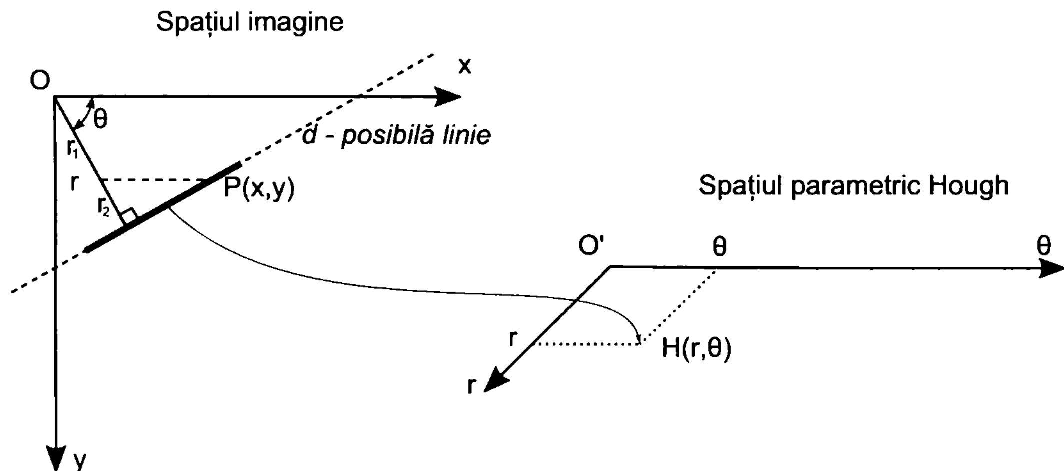


Figura 3.1: Transformata Hough pentru detecţia de linii. Fiecărei linii d din spaţiul imagine îi corespunde un punct în spaţiul parametric Hough.

Pasul următor rezidă în construirea unei matrici cu $\frac{x+\sqrt{x^2+y^2}}{\Delta r} \times \frac{180}{\Delta\theta}$ elemente, unde linia reprezintă distanţa, iar coloana reprezintă unghiul. Elementele matricei sunt iniţial nule. Pentru toate valorile discrete posibile ale lui r şi θ, se construiesc dreptele corespunzătoare şi se numără prin câte din punctele imaginii trec. Acest număr reprezintă valoarea celulei respective. Matricea mai poartă numele de acumulator Hough. Un exemplu de astfel de matrice este prezentat în Figura 3.2. Folosind acest algoritm, la sfârsit, în matrice se vor afla numere

BUPT

întregi nenegative care reprezintă numărul de puncte prin care trece fiecare dreaptă. Cercurile marchează punctele de maxim ale matricei, puncte care reprezintă drepte cu număr maxim de puncte.

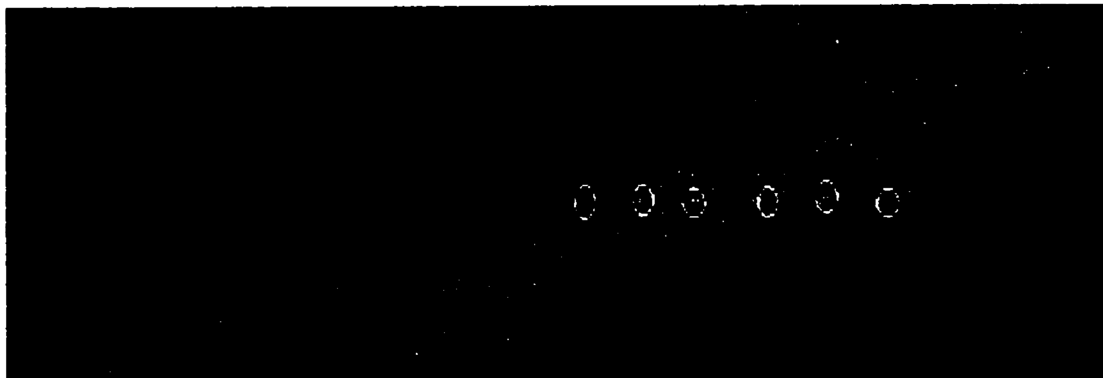


Figura 3.2: Exemplu de matrice de acumulare rezultată în urma aplicării transformatei Hough. Cercurile marchează maximele corespunzătoare celor şase linii detectate. Afişarea grafică foloseşte negru pentru elemente nule şi tonuri de gri cu iluminare proporţională cu valoarea respectivului element pentru valori nenule. Albul se foloseşte pentru elementul sau elementele cu număr maxim de puncte.

3.2 Reţele neuronale artificiale

Reţelele neuronale artificiale (RNA) fac parte din subdomeniul *Machine Learning (Învăţare automată)* al *Inteligenţei Artificiale*. RNA sunt metode statistice de clasificare, cea mai importantă aplicaţie a lor fiind în recunoaşterea formelor. Există multe cărţi care tratează acest domeniu, unele din cele mai importante fiind [10] (abordează întreg domeniul recunoaşterii formelor) şi [9] (dedicată exclusiv reţelelor neuronale artificiale). Pentru o introducere concisă şi clară se poate utiliza [51] (Capitolul 6).

RNA sunt alcătuite din unităţi de procesare numite neuroni. Neuronul artificial este un model matematic simplist al neuronului biologic. Neuronii unei RNA sunt, de obicei, dispuşi pe mai multe straturi.

RNA pot fi împărţite în două mari categorii: RNA cu propagare înainte şi RNA recurente. Prima categorie este de departe mult mai studiată, RNA cu propagare înainte fiind mult mai bine înţelese şi din punct de vedere teoretic. RNA utilizate în această teză sunt de tipul RNA cu propagare înainte.

Într-o RNA cu propagare înainte există conexiuni numai între neuroni de pe straturi consecutive, informaţia de la intrare propagându-se secvenţial prin fiecare strat până la ieşire.

În scopul evitării confuziilor, în această teză am considerat ca strat al reţelei neuronale inclusiv stratul de intrare, strat necomputaţional. Astfel, pentru o reţea cu trei straturi, stratul L_0 este stratul de intrare, iar straturile L_1 şi L_2 sunt straturi computaţionale. Straturile interioare, în acest caz stratul L_1, se numesc straturi ascunse.

BUPT

3.2.1 Perceptronul

Perceptronul [46] (pagina 458) este un model simplu de reţea neuronală cu propagare înainte. Are două straturi, unul fiind de intrare şi unul de ieşire. Stratul de ieşire este unicul strat computaţional. Întrucât nu poate rezolva decât probleme liniar separabile, are o arie redusă de aplicabilitate.

Perceptronul multistrat este o variantă mai complexă de perceptron, putând avea unul sau mai multe straturi ascunse. Prezenţa straturilor ascunse elimină limitarea aplicabilităţii la probleme liniar separabile.

3.2.2 Reţele neuronale artificiale cu hărţi de convoluţie (RNC)

Având în vedere modul de funcţionare al percepţiei vizuale [8] (Capitolul 2), s-a încercat modelarea acestuia folosind reţele neuronale. Unul din aceste modele foloseşte hărţi de convoluţie. RNC sunt tot RNA multistrat cu propagare înainte, unica diferenţa fiind arhitectura reţelei, mai exact, modul de conectare a neuronilor. Evident, RNC sunt concepute pentru recunoaşterea de caracteristici ale imaginilor.

În 1979 Fukushima [28] a propus o reţea neuronală cu hărţi de convoluţie, numită Neocognitron, care se putea adapta la deplasarea caracterelor în câmpul de intrare. Detalii pot fi găsite la *http://www4.ocn.ne.jp/~fuku_k/files/research-e.html#neocog.*

LeCun a folosit în 1995 o reţea de convoluţie pentru recunoaşterea de caractere. În Figura 3.3 este prezentată arhitectura acestei reţele. În [38] arhitectura este dezvoltată şi mai mult, prin adăugarea de noi straturi.

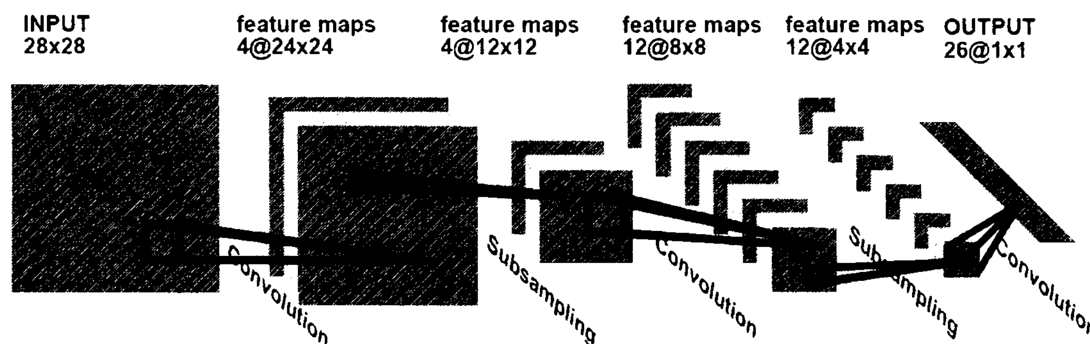


Figura 3.3: RNC pentru recunoaşterea caracterelor. Preluare din [37].

Spre deosebire de perceptron, unde fiecare neuron al unui strat era conectat cu toţi neuronii stratului anterior, la RNC fiecare neuron este conectat numai la o parte a neuronilor stratului anterior. Pentru a imita funcţionarea sistemului vizual uman, neuronii cu care este conectat se află toţi în aceeaşi zonă. O reţea neuronală poate conţine atât straturi complet conectate, cât şi straturi de convoluţie. Straturile de convoluţie sunt, uzual, alcătuite din mai multe hărţi de convoluţie.

În RNC utilizate în Capitolul 5 există două moduri de conectare a neuronilor de pe straturile de convoluţie, al doilea mod fiind o extindere a primului [64, 38].

Primul mod de conectare (stratul L_1 din Figura 3.5) constă în a conecta fiecare neuron al unei hărţi la un nucleu de neuroni ai stratului anterior. Toţi neuronii unei hărţi împart aceleaşi

BUPT

ponderi, astfel numărul de ponderi fiind mult redus faţă de cazul unui strat complet conectat. Nucleul de convoluţie este deplasat asupra stratului anterior din doi în doi neuroni, atât pe orizontală, cât şi pe verticală (Figura 3.4). Dimensiunea uzuală [64] pentru nucleu este de 5×5 neuroni. Aceasta trebuie să fie impară pentru ca nucleul să poată fi centrat într-un pixel. Simard [64] a arătat că o dimensiune mai mică, 3×3 neuroni, este prea redusă pentru a detecta caracteristici primare (colţuri, intersecţii, capete de linie) în imagine, iar 7×7 neuroni este deja prea mult, depăşind dimensiunea caracteristicilor primare.

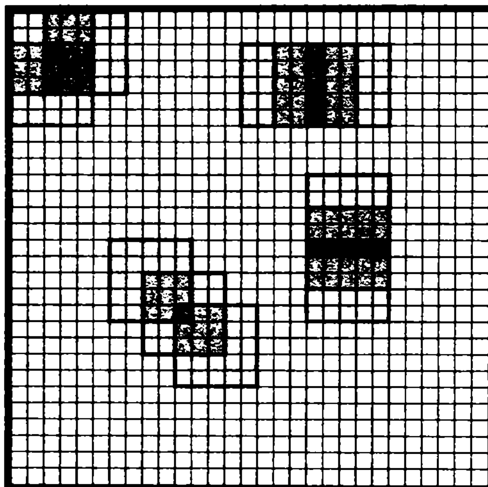


Figura 3.4: Modul de deplasare a nucleului de convoluţie.

Al doilea mod de conectare poate fi observat între straturile L_1 şi L_2 din Figura 3.5. Şi în acest caz neuronii dintr-o hartă împart acelaşi set de ponderi, dar fiecare neuron de pe stratul L_2 este conectat cu neuroni din nuclee de convoluţie din *fiecare* hartă a stratului anterior.

3.2.3 Metode de antrenare a reţelelor neuronale

Antrenarea reţelelor cu propagare înainte se face de obicei cu algoritmul de învăţare cu propagare înapoi a erorii (backpropagation learning). Acesta este prezentat în toate cărţile ce tratează reţelele neuronale, astfel că nu va mai fi expus în această lucrare. Programul de antrenare a reţelelor neuronale pe care le-am folosit în cadrul experimentelor este conceput după explicaţiile din [61](Capitolul 20.5). Am ales această carte deoarece expune într-un mod matematic riguros şi detaliat algoritmul de antrenare.

În cazul algoritmilor ce au ca principiu minimizarea erorii medii pătratice, se poate folosi o tehnică de accelerare a antrenării. Algoritmul Levenberg-Marquardt [41] (o descriere a sa pentru RNA este dată de Bishop în [9], Capitolul 7.11) poate fi aplicat pentru minimizarea funcţiei de eroare.

BUPT

harta 0

harta 0

harta 1

harta 1

harta k-1

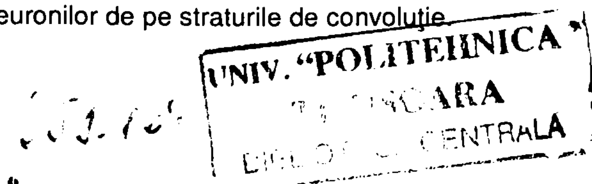
harta p-1

strat de intrare L0

strat de convoluție L1

strat de convoluție L2

Figura 3.5: Modul de conectare a neuronilor de pe straturile de convoluție.



BUPT

Capitolul 4

Baze de date cu cifre şi şiruri de cifre

Acest capitol prezintă bazele de date cu cifre şi cu şiruri de cifre care au fost folosite pentru obţinerea seturilor de antrenare şi de test. Cu aceste seturi au fost apoi efectuate experimentele din capitolele ce urmează.

Prima bază de date, NIST SD 19 ([4]), este şi cea mai mare dintre toate. Celelalte trei, MNIST ([40]), LNIST şi baza de date la nivel de două cifre parţial suprapuse sunt derivate din aceasta. Seturile de test 3DNS, 6DNS şi 10DNS sunt obţinute tot din NIST SD 19, prin extragerea câmpurilor numerice de 3, 6, respectiv 10 cifre. Figura 4.1 prezintă modul în care relaţionează seturile de antrenare şi de test, iar Tabelul 4.1 - conţinutul şi modul de utilizare al seturilor. În Figura 4.1 am evidenţiat şi clasificatorii la nivel de cifră şi de două cifre parţial suprapuse ce vor fi folosiţi în sistemul de clasificare general (Figura 5.4).

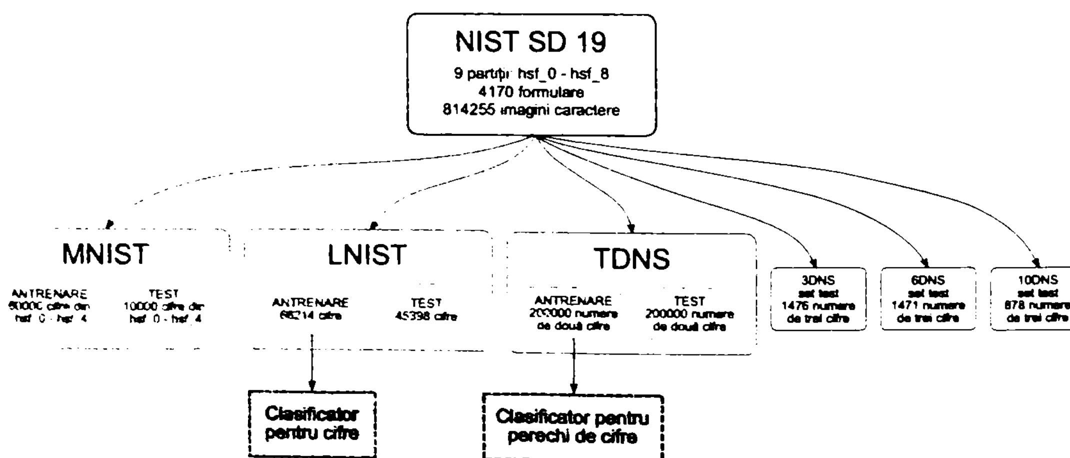


Figura 4.1: Ierarhia seturilor de test şi de antrenare.

BUPT

Tabelul 4.1: Conţinutul şi modul de utilizare ale seturilor de date.

nume	număr de imagini	tip imagini	utilizare	nr. persoane
NIST SD 19	814255 + 4170	variat	variată	4170
MNIST TRAIN	60000	cifre	antrenare	≈500
MNIST TEST	10000	cifre	test	necunoscut
LNIST TRAIN	66214	cifre	antrenare	600
LNIST TEST	45398	cifre	test	400
TDNS TRAIN	200000	numere a 2 cifre	antrenare	1000
TDNS TEST	200000	numere a 2 cifre	test	1000
3DNS	1476	numere a 3 cifre	test	296
6DNS	1471	numere a 6 cifre	test	296
10DNS	878	numere a 10 cifre	test	296

4.1 NIST SD 19

Baza de date NIST SD 19 ([4]) conţine imagini de formulare scanate scrise de 4170 de persoane. Un exemplu de astfel de formular este dat în Figura 4.2. Formularele vor fi referite în text prin HSF (Handwriting Sample Form). În continuare vor fi prezentate numai informaţiile necesare înţelegerii desfăşurării experimentelor.

CD-ul cu baza de date conţine gata extrase, eventual segmentate, cifrele şi literele din aceste formulare, în total 814255 de imagini. Pe lângă imagini, baza de date conţine detalii privind originea, organizarea, data publicării respectivelor imagini, precum şi modul recomandat de utilizare a imaginilor. Caracterele segmentate sunt memorate în imagini de 128×128 pixeli şi sunt etichetate folosind codurile ASCII corespunzătoare caracterelor 0-9, a-z, A-Z. Autorii NIST SD 19 afirmă că etichetarea imaginilor a fost verificată manual, iar eroarea care mai există este de aproximativ 0.1%.

NIST SD 19 conţine toate imaginile din versiunile anterioare, SD3 şi SD7 (Tabelul 4.2), plus multe altele. Imaginile segmentate sunt organizate în diverse ierarhii pentru a facilita moduri variate de utilizare a lor. Astfel, există referinţe pentru ierarhizarea imaginilor în funcţie de persoană, clasă, clasă fără informaţie de specificare a tipului literei (majusculă sau minusculă), câmpul din formularul HSF.

Tabelul 4.2: Versiuni anterioare ale bazei de date NIST cu imagini de caractere scrise de mână. (după [4])

nume	data	versiune	conţinut	observaţii
Special Database 1	mai 1990	HWDB Release 1-1.1	formulare şi câmpuri	
Special Database 3	februarie 1992	HWSC Release 4-1.1	caractere	
Special Database 7	aprilie 1992	TST1 Release 6-1.1	caractere	Test Data 1.

4.1.1 Structura NIST SD 19

NIST CD 19 conţine opt partiţii de imagini HSF, numerotate *hsf_{0,1,2,3,4,6,7,8}*. HSF-urile au fost scanate la 300dpi şi, aşa cum se poate observa şi în Figura 4.2, conţin 34 de câmpuri

BUPT

HANDWRITING SAMPLE FORM

NAME DATE CITY STATE ZIP

This sample of handwriting is being collected for use in testing computer recognition of hand printed numbers and letters. Please print the following characters in the boxes that appear below.

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9

08 395 5268 49171 235969

111 2956 81207 758298 90

4671 34560 368704 27 493

75434 281512 02 564 3000

335706 48 863 4699 82771

l a n g u o y b x q d t p e m c h r v f i s j z w k

M O F W T U I X Y H S D Z K E B Q R L P G C V N J A

Please print the following text in the box below:

We, the People of the United States, in order to form a more perfect Union, establish Justice, insure domestic Tranquility, provide for the common Defense, promote the general Welfare, and secure the Blessings of Liberty to ourselves and our posterity, do ordain and establish this CONSTITUTION for the United States of America.

Figura 4.2: Exemplu de formular din NIST SD 19 (formularul f1816_24). A se vedea şi Figura 4.4.

BUPT

Tabelul 4.3: Bazele de date în care au fost publicate partiţiile. † - câmpuri completate, dar neprocesate. ø - câmpuri necompletate. (după [4])

partiţia	persoane	formulare	cifre	majuscule	minuscule	constituţia	originea persoanelor
hsf_0	0000-0499	SD1	SD3	SD3	SD3	SD19	Census Field
hsf_1	0500-0999	SD1	SD3	SD3	SD3	SD19	Census Field
hsf_2	1000-1499	SD1	SD3	SD3	SD3	SD19	Census Field
hsf_3	1500-2099	SD1	SD3	SD3	SD3	†	Census Field
hsf_4	2100-2599	SD19	SD7	SD7	SD7	ø	High School
hsf_5	2600-3099	păstrate de NIST pentru teste viitoare				ø	High School
hsf_6	3100-3599	SD19	SD19	SD19	SD19	ø	Census MD
hsf_7	3600-4099	SD19	SD19	SD19	SD19	ø	Census MD
hsf_8	4100-4169	SD19	†	†	†	†	Census MD

cu informaţii diverse. În această lucrare au fost utilizate doar câmpurile numerice.

În Tabelul 4.3 este dată alcătuirea NIST SD 19, precum şi conţinutul SD1, SD3 şi SD7.

Primele 429 de formulare din *hsf_5* au fost scrise de elevi de liceu, iar ultimele 71, precum şi toate formularele din partiţiile *hsf_{6,7,8}* au fost scrise de angajaţi ai biroului de recensământ din Suitland, Maryland. Informaţiile acestea se vor dovedi utile la construirea seturilor de antrenare şi de test, precum şi la evaluarea rezultatelor.

4.1.2 Conţinutul NIST SD 19

Datele fiecărei persoane se află în directoare cu nume de forma *fyyyy_zz*, unde *yyyy* reprezintă codul persoanei, putând lua valori între 0000 şi 4099, iar *zz* este un cod prin care se precizează tipul formularului completat de persoana respectivă. Toate formularele au acelaşi număr de câmpuri, câmpurile cu acelaşi număr având aceeaşi dimensiune. Variază numai ordinea cifrelor, respectiv a caracterelor, din interiorul câmpului. Acest cod poate fi utilizat pentru a accesa formularul corespunzător persoanei respective, în directorul *data/hsf_pages/truerefs*. Conţinutul (Tabelul 4.4) acestor fişiere poate fi utilizat pentru a afla ce ar fi trebuit să scrie persoana în câmpurile formularului. Am folosit aceste informaţtii la construirea seturilor de test cu numere formate din 3, 6 şi 10 cifre. În Figura 4.4 este descris conţinutul câmpurilor din formularul din Figura 4.2. Formularele există şi în variantă necompletată, pentru a putea fi utilizate la colectări suplimentare de date.

Câmpurile de la 3 până la 31 conţin cifre şi numere formate din mai multe cifre. Conţinutul câmpurilor a fost extras şi e gata segmentat la nivel de caracter (cifră sau literă) în baza NIST SD 19. Toate caracterele au fost etichetate automat folosind fişierele cu referinţe, şi apoi verificate manual ([68]) de două ori de către persoane distincte. În urma segmentării imaginiile au fost stocate într-o ierarhie de fişiere. Pentru a afla exact provenienţa şi conţinutul unei imagini se foloseşte numărul partiţiei, numărul persoanei şi tipul formularului.

Formularele au fost concepute astfel încât pentru fiecare clasă să existe acelaşi număr de exemplare. După operaţiile de extragere şi de segmentare, aceste numere nu mai sunt egale, dar sunt destul de apropiate (Tabelul 4.5).

Din păcate, baza de date nu conţine şi imaginile cu şirurile de cifre extrase din câmpurile formularelor, ci numai cu cifrele gata segmentate din aceste imagini. Din acest motiv, pentru experimentele asupra şirurilor de cifre a fost necesară dezvoltarea unei metode de extragere a acestor şiruri de cifre.

BUPT

Tabelul 4.4: Exemplu de fişier cu referinţe (*hsf_page/truerefs/ref_24.txt*). A se vedea şi Figura 4.2.

fld_0		fld_1	
fld_2		fld_3	0123456789
fld_4	0123456789	fld_5	0123456789
fld_6	08	fld_7	395
fld_8	5268	fld_9	49171
fld_10	235969	fld_11	111
fld_12	2956	fld_13	81207
fld_14	758298	fld_15	90
fld_16	4671	fld_17	34560
fld_18	368704	fld_19	27
fld_20	493	fld_21	75434
fld_22	281512	fld_23	02
fld_24	564	fld_25	3000
fld_26	335706	fld_27	48
fld_28	863	fld_29	4699
fld_30	82771	fld_31	languoybxqdtpemchrvfisjzwk
fld_32	MOFWTUIXYHSDZKEBQRLPGCVNJA	fld_33	

Tabelul 4.5: Numărul de exemplare din fiecare clasă.

Clasa	0	1	2	3	4	5	6	7	8	9
Exemplare	40363	44704	40072	41112	39154	36606	39937	41893	39579	39533

4.1.3 Ierarhizarea datelor

Datele sunt ierarhizate după: formulare şi partiţii, autor, câmp, numărul clasei în hexa, numărul clasei unite (fără discriminare între literele minuscule şi cele majuscule). Din aceste ierarhii prezintă interes pentru lucrarea de faţa numai cele la nivel de formular şi partiţie, precum şi cea la nivel de autor.

Formatul fişierelor este detaliat în [4]. NIST SD 19 conţine patru tipuri de fişiere, tipul lor fiind determinat de extensie:

.mis – fişier ce conţine (mai multe) imagini de caractere. De obicei este comprimat.

.pct – fişier ce conţine imaginea unui formular. Există câte un astfel de fişier pentru fiecare din cele 3669 de persoane.

.cls – fişier ce conţine clasele (etichetele) imaginilor din fişierul *.mis* cu acelaşi nume.

.mit – fişier ce conţine un pointer la un fişier *.mis*. Conţine identitatea persoanei, precum şi calea şi numele fişierului cu imagini segmentate ale acelei persoane. Deoarece există mai multe ierarhii de date, imaginile ar fi trebuit multiplicate pentru fiecare ierarhie, ceea ce ar fi consumat inutil spaţiu. Pentru a evita acest lucru, şi pentru a permite stocarea întregii baze de date pe un singur CD, imaginile apar o singură dată -în fişiere *.mis*-, iar în rest doar se face referire la ele folosind fişiere *.mit*.

BUPT

```
hsf_page
|
|
hsf_0 ... hsf_8                  truerefs                  template
|                                |                         |
|                                |                         |
f0000_14.pct ... f0499_10.pct    ref_00.txt ... ref_99.txt  ref_00.ps ... ref_99.ps
```

Figura 4.3: Arborele aferent ierarhizării formularelor în partiţii.

Ierarhizare după partiţii şi formulare

Imaginile formularelor sunt structurate într-un arbore (Figura 4.3) ce are pe primul nivel partiţiile *hsf_{0,1,2,3,4,6,7,8}* şi directorul pentru referinţe *truerefs*. De asemenea, mai există şi directorul pentru formularele necompletate (*template*).

Ierarhizare după persoane

Acest arbore (Figura 4.4) conţine caracterele organizate după partiţie şi apoi după persoana care le-a scris. Datele din acestă ierarhie reprezintă datete prime obţinute după procesul de segmentare, celelalte ierarhii fiind doar reorganizări ale datelor din aceasta. În directorul fiecărei persoane se află patru perechi de fişiere *.mis* şi *.cls*, câte una pentru cifre, litere minuscule, litere majuscule, caractere din câmpul Constituţie. Fişiere *.mit* nu sunt necesare deoarece toate datele se află în directorul curent.

4.1.4 Formatul fişierelor NIST

Imaginile sunt stocate în format binar, alb-negru. Dimensiunea unei imagini ce reprezintă un simbol este de 128×128 pixeli. Informaţia de culoare a pixelilor este memorată la nivel de bit, astfel că o imagine necesită $128 \times 128/8 = 2048$ octeţi. Imaginile simbolurilor sunt stocate una dupa celalaltă, iar blocul rezultat este comprimat cu o variantă a algoritmului CCITT Group 4. Toate fişierele care conţin imagini au un antet de tip IHEAD ataşat blocului ce conţine imaginile comprimate. Informaţii detaliate despre modul de codificare pot fi gasite în [4]. Tipul acesta de codificare se foloseşte la fişierele *.mis*, ele fiind unicele care conţin imagini de caractere. Fişierele care conţin imaginea unui formular întreg sunt de tipul *.pct*.

```
by_write
|
|
hsf_0         hsf_7
|
|
f0000_14  f0001_41 ... f0499_10
|
|
d0000_14.mis u0000_14.mis l0000_14.mis c0000_14.mis
d0000_14.cls u0000_14.cls l0000_14.cls c0000_14.cls
```

Figura 4.4: Arborele aferent ierarhizării imaginilor de caractere după partiţii şi persoane.

BUPT

Fiecărui caracter segmentat din formulare îi este atribuită o etichetă prin care se specifică din ce clasă face parte. Această eticheta reprezintă chiar codul ASCII al caracterului, în cazul cifrelor având valori între $30_{(16)}$ şi 39_{16}. Etichetele cifrelor dintr-un fişier *.mis* se află memorate într-un fişier *.cls* corespunzător. Acesta este un fişier text ce are pe prima linie numărul de cifre care se află atât în *.mis* cât şi în *.cls*. Pe rândurile următoare se află câte o etichetă.

4.1.5 Programe utilitare ale NIST SD 19

Pe lângă imaginile din baza de date mai există şi utilitare pentru comprimarea şi decomprimarea imaginilor. Deoarece aceste utilitare sunt scrise în C pentru UNIX, a fost necesară modificarea lor pentru arhitectura x86. Întrucât numărul funcţiilor ce au trebuit adaptate este de peste o sută şi se află în câteva zeci de fişiere (peste 300kB de surse C), portarea utilitarelor pe arhitectura 80x86 nu a fost deloc facilă. Din multele utilitare ale NIST SD 19, am modificat utilitarul *normmis* care normalizează imaginile dintr-un fişier *.mis* la dimensiunea de 32×32 pixeli. Pentru a face această normalizare -care era nedorită pentru modul în care doream să folosesc imaginile din *.mis- normmis* întâi decomprima fişierul *.mis*. Am folosit acest lucru şi am modificat programul astfel încât să salveze imaginile în format *.bmp*, câte un fişier pentru fiecare imagine din *.mis*. Am obţinut peste o sută de mii de fişiere *.bmp*, fiecare conţinând o imagine de 128×128 pixeli. La salvarea fiecărui fişier *.bmp* am avut grijă ca numele să conţină toate datele de identificare a imaginii: formular, persoană, etichetă. Pentru crearea ulterioară a seturilor de antrenare şi de test am scris un utilitar care pe baza unui fişier de configurare alege doar anumite imagini din cele peste 100000 de bmp-uri.

4.2 MNIST

Baza de date MNIST (Modified NIST) [40] a fost creată de Corinna Cortez şi apoi modificată de Yann LeCun prin centrarea imaginilor în centrul de masă.

Conţine două seturi de imagini etichetate, unul pentru antrenare alcătuit din 60000 de imagini de cifre, şi unul pentru testare alcătuit din 10000 de imagini. Aşa cum îi spune şi numele, este o modificare a NIST SD 19. Imaginile în format alb negru din NIST au fost redimensionate la 20×20 pixeli cu păstrarea aspectului. În urma procesului de redimensionare, imaginile sunt reprezentate în tonuri de gri. Imaginea de 20×20 pixeli este apoi poziţionată într-una de 28×28 pixeli, prin suprapunerea centrului de masă al primeia cu centrul imaginii mari.

Baza de date MNIST conţine imagini de cifre scrise de mână din NIST SD 3 şi din NIST SD 1. În urma multor experimente cu aceste baze de date, Yann LeCun a constatat că SD 3 este mult mai uşor de recunoscut decât SD 1 deoarece conţine imagini scrise de personal de la biroul de recensământ al populaţiei, iar SD1 conţine imagini cu cifre scrise de elevi de liceu. Deoarece a dorit ca rezultatele sistemului de clasificare să nu fie influenţate de persoanele care au scris respectivele cifre, a hotărât să creeze o nouă bază de date prin amestecarea datelor din SD 1 şi SD 3. Pentru aceasta, a construit setul de antrenare folosind 30000 de imagini din SD 1 şi 30000 de imagini din SD 3. Pentru setul de test a folosit câte 5000 de imagini atât din SD 1, cât şi din SD 3. Setul de antrenare conţine date de la aproximativ 500 de persoane, iar mulţimile de persoane ale căror imagini de cifre au fost folosite pentru construirea seturilor de antrenare şi de test au fost disjuncte.

BUPT

Întrucât în SD 1 imaginile cifrelor pentru fiecare persoană nu se află la un loc, au fost folosite informaţiile din fişierele "cls" (care conţin date despre provenienţa fiecărei imagini a unei cifre) pentru rearanjarea cifrelor în funcţie de persoana care le-a scris. Detaliile procedurii pot fi urmărite în [40].

Baza de date este structurată în patru fişiere:

- train-images-idx3-ubyte: imaginile setului de antrenare,
- train-labels-idx1-ubyte: etichetele setului de antrenare,
- t10k-images-idx3-ubyte: imaginile setului de test,
- t10k-labels-idx1-ubyte: etichetele setului de test.

În Anexa 7.3 este prezentat modul de codificare a informaţiilor din fişiere.

4.3 LNIST - Liu-NIST

Aceste seturi de antrenare şi de test au fost construite pentru a putea face o comparaţie cât mai echitabilă între metoda propusă în această lucrare şi metodele din [44]. Ele conţin exact aceleaşi imagini ca seturile folosite în [44]. Setul de test este compus din 66214 cifre extrase din 600 de formulare NIST: 0-399 şi 2100-2299. Pentru extragerea acestor cifre am folosit unele din utilitarele mentionate în Secţiunea 4.1. După extragerea imaginilor comprimate din fişierele *.mis*, acestea au fost decomprimate şi s-au obţinut imagini alb-negru cu dimensiunea de 128×128 pixeli. Cifrele din aceste imagini au fost încadrate în dreptunghiul ce le delimitează, rezultatul fiind redimensionat cu păstrarea aspectului la 20×20 pixeli. În urma redimensionării, imaginile au fost transformate din imagini binare $(0/1)$ în imagini cu tonuri de gri $(0 \div 255)$. Apoi s-a calculat centrul de masă al fiecărei imagini şi bitmap-ul de 20×20 a fost poziţionat într-unul de 29×29 prin suprapunerea centrului de masă cu centrul imaginii de 29×29 (Figura 4.5). Toate aceste prelucrări au fost efectuate cu un program C++ care are ca rezultat final două fişiere de tip *.idx* (a se vedea Anexa 7.3), unul conţinând cele 66214 imagini, iar celălalt etichetele corespunzătoare, etichete care au fost obţinute din fişierele *.cls* corespunzătoare fişierelor *.mis*.

Setul de test a fost obţinut în acelaşi mod, dar folosind imaginile cu cifre din 400 de formulare: 500-699 şi 2400-2599. În total sunt 45398 de imagini.

Niciuna din cele 111612 imagini nu a fost curăţată sau îmbunătăţită în niciun fel.

Avantajul folosirii centrării în centrul de masă faţă de metoda dreptunghiului delimitator rezidă în sensibilitatea scăzută la zgomotele ce ar putea apărea în imagine. La metoda dreptunghiului delimitator e suficient ca un singur pixel de zgomot să se afle la distanţă de imaginea cifrei pentu ca imaginea să apară translatată puternic la intrarea clasificatorului. Centrarea în centrul de masă atenuează foarte mult acest efect al zgomotelor exterioare cifrei propriu-zise, păstrând toate imaginile în aproximativ aceeaşi poziţie a câmpului de intrare al clasificatorului. În Figura 4.5 este evidenţiat un caz de zgomot care translatează orizontal imaginea în cazul utilizării metodei dreptunghiului delimitator (imaginile din dreapta). Ultimul rând de imagini arată mai clar efectul centrării în centrul de masă, prin suprapunerea imaginilor de pe coloanele corespunzătoare. Se observă că în cazul centrării în centrul de masă imaginea obţinută prin suprapunere arată mult mai puţin distorsionată faţă de imaginea obţinută prin suprapunerea imaginilor centrate folosind metoda dreptunghiului delimitator. Cel mai bine este evidenţiată diferenţa pentru golul cifrei nouă, gol care este mult diminuat în cazul din dreapta. Dacă ar fi existat şi un exemplar al cifrei nouă cu zgomot în partea opusă

BUPT

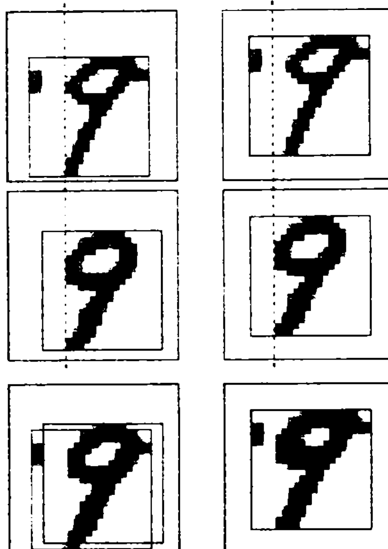


Figura 4.5: Comparaţie între centrarea în centrul de masă (stânga) şi centrarea în dreptunghiul delimitator (dreapta). Punctul roşu reprezintă centrul de masă al imaginii. Pe ultimul rând se află suprapunerea celor două imagini de pe coloana corespunzătoare.

-în dreapta-, iar la un set de antrenare format din zeci de mii de imagini aproape sigur există şi astfel de exemplare, atunci golul cifrei nouă până la urmă ar fi dispărut dacă nu s-ar fi folosit metoda centrării în centrul de masă.

4.4 TDNS (Two-digit Numeral Strings)

Această bază de date am creat-o pentru antrenarea unui clasificator care să recunoască şiruri de două cifre parţial suprapuse. NIST SD 19 conţine câmpuri cu numere formate din două cifre, dar numărul lor este relativ redus deoarece există numai 4170 de persoane şi pentru fiecare persoană sunt câte 4 câmpuri de două cifre. Dar extragerea acestor informaţii din formulare nu este facilă deoarece trebuia implementat un program care să detecteze şi să extragă imaginile din formular. Programul a fost implementat deoarece am avut nevoie şi de imaginile cu numere de trei sau mai multe cifre. Însă, multe din imagini cu două cifre nu conţin cifre parţial suprapuse, astfel că numărul final de imagini pentru antrenarea reţelei ar fi insuficient deoarece antrenarea necesită o cantitate considerabilă de date.

Considerând numerele formate din două cifre, există 100 de clase, de la 00 la 99. Pentru fiecare clasă sunt necesare cel puţin câteva sute de exemplare pentru ca antrenarea să reuşească. Colectarea manuală, urmată de procesare şi etichetare a atât de multor date este aproape imposibilă, oricum, foarte indelungată şi plictisitoare. În consecinţă, sunt necesare metode alternative. Am prezentat anterior mai multe baze de date care conţin imagini cu cifre etichetate. Metoda alternativă aleasă constă în conceperea unei metode de generare automată a imaginilor cu două cifre parţial suprapuse. Pentru intrare am folosit imagini din NIST SD 19 ([4]) şi nu din MNIST, deoarece în MNIST imaginile au fost deja prelucrate în diverse moduri şi am dorit sa nu influenţez în niciun fel rezultatele procesului de recunoaştere.

BUPT

NIST SD 19 conţine nouă partiţii, de la hsf_0 la hsf_8. Am folosit partiţiile $hsf_0 \div hsf_3$ care conţin imagini de la 2100 de persoane. Din aceste date am generat automat atât setul de antrenare, cât şi cel de test. În NIST SD 19 se specifică a se folosi partiţiile $hsf_0 \div hsf_3$ pentru antrenare, dar aceste specificaţii sunt date în ideea procesului de recunoaştere la nivel de cifră, neaplicându-se şi pentru şiruri de cifre. Un alt motiv pentru care nu am respectat îndrumările din NIST este acela că partiţia hsf_4 care este menţionată a se folosi ca set de test conţine numai 500 de persoane, şi am dorit să creez seturi de test cât mai diverse şi cât mai mari. În consecinţă, am utilizat imaginile primelor 1000 de persoane pentru antrenare, şi imaginile următoarelor 1000 de persoane pentru test. Detaliile vor fi prezentate în secţiunile dedicate setului de antrenare, respectiv celui de test.

4.4.1 Setul de antrenare

Acest set conţine primele 1000 de persoane, exceptându-le pe cele cu indicii: 48, 52, 741, 785, 802, 825 şi 848. Nu s-au utilizat datele de la aceste persoane deoarece erau incomplete în baza de date NIST SD 19. Lipsea unul din fişierele cu imagini sau cel cu etichete, sau nu existau cifre pentru unele clase. Rezultatul este format din 1000 de grupuri a zece imagini de cifre. Rezoluţia în NIST SD 19 este 128×128 pixeli, şi imaginile sunt codificate alb-negru. Am efectuat iniţial experimente direct cu aceste imagini fără a le corecta în vreun fel, dar erorile au fost semnificative, indiferent de complexitatea reţelei neuronale utilizate. Era evident că imaginile au probleme, astfel că le-am studiat mai în amănunt şi am observat că multe conţin pixeli disparaţi sau artefacte generate de preprocesarea NIST. (Figura 4.6)

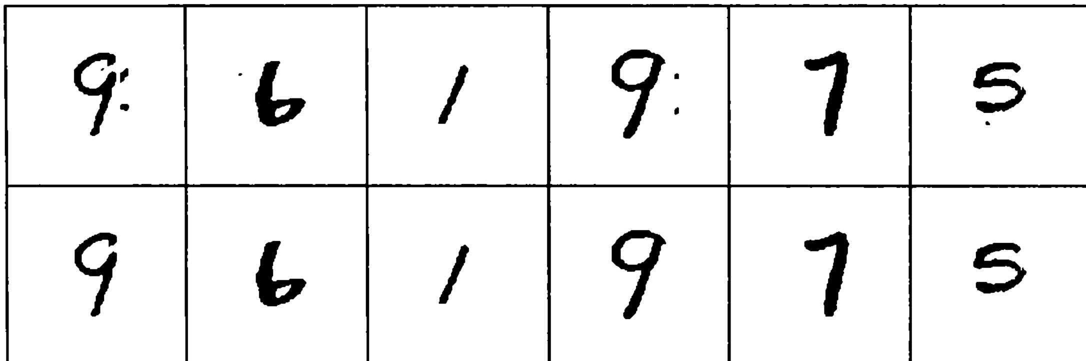


Figura 4.6: Primul rând - exemple de artefacte în imagini 128×128 pixeli din NIST SD 19; rândul al doilea - imaginile rezultate în urma procesului de corecţie semiautomată.

Aceşti pixeli şi artefacte nu afectează direct capacitatea de recunoaştere a reţelei neuronale. Dacă imaginile sunt utilizate pentru un clasificator la nivel de cifră, nu este necesară nicio corectare a lor. Demonstraţia acestui fapt este prezentată în Capitolul 5 la experimentele efectuate cu cifre singulare. Aceste artefacte afectează negativ procesul de unire a cifrelor. Problema majoră rezidă în calcularea poziţiei de suprapunere, cauzând generarea de imagini nerealiste. Pentru a evita astfel de erori am hotărât să elimin artefactele şi pixelii eronaţi. Am folosit o metodă parţial automată prin scrierea unui program care detectează numărul componentelor conexe din imagine. Dacă acesta este mai mare decât unu, imaginea este prezentată operatorului uman şi fiecare componentă conexă este marcată cu o altă culoare

BUPT

(Figura 4.9). În majoritatea cazurilor eliminarea tuturor componentelor exceptând-o pe cea mai mare a fost suficientă pentru corecţie. A existat un număr redus de cazuri de cifre cu linii întrerupte. Soluţia aleasă pentru corectarea lor a fost ştergerea manuală a componentelor care ar fi afectat procesul de suprapunere. Într-un număr foarte redus au existat cazuri de imagini foarte deformate rezultate în urma unui proces defectuos de extragere când baza de date NIST SD 19 a fost colectată şi procesată. Aceste cazuri au fost corectate manual. Câteva exemple pot fi observate în Figura 4.7.

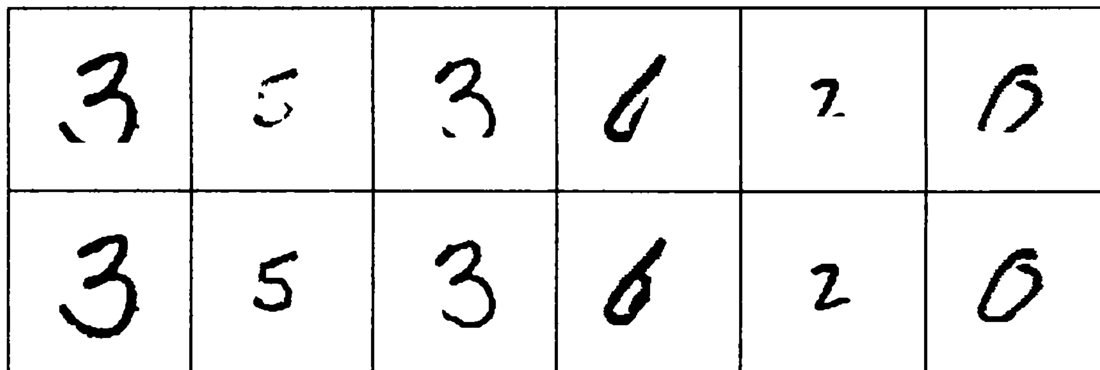


Figura 4.7: Primul rând - imagini defectuos segmentate sau deformate din baza NIST SD 19; rândul al doilea - imagini rezultate în urma corecţiei manuale.

Fiecare imagine de 128×128 pixeli conţine un caracter care poate avea dimensiuni foarte diferite. A fost necesară o redimensionare pentru normalizarea cifrelor din setul de imagini. S-a detectat dreptunghiul delimitator al cifrei şi apoi conţinutul acestuia a fost scalat proporţional până la dimensiunea de 20×20 pixeli (4.8). Rezultatul scalării este o imagine cu tonuri de gri afectată de aliasing. Am ales această dimensiune deoarece este suficient de mare pentru a reprezenta complet şi în detaliu un caracter, şi este mai mică decât marea majoritatea a imaginilor din baza de date NIST. De asemenea, şi baza de date MNIST foloseşte aceeaşi dimensiune pentru redimensionare şi am dorit o tratare unitară pentru toate experimentele următoare.

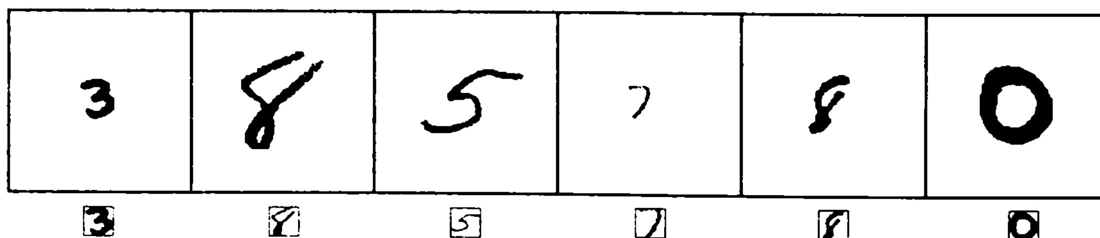


Figura 4.8: Primul rând - imagini 128×128 pixeli din NIST SD 19; rândul al doilea - imagini 20×20 pixeli rezultate după tăiere relativă la dreptunghiul delimitator, centrare şi redimensionare.

Am pornit de la fişierele cu imagini comprimate *.mis* şi de la fişierele cu etichete *.cls* utilizate în NIST SD 19 şi după ce am extras datele de care am avut nevoie, le-am salvat în formatul *.idx* care are avantajul că este un format necomprimat care poate fi utilizat în mod

BUPT

direct. De asemenea, poate conţine tonuri de gri, nefiind limitat doar la alb sau negru. Acest tip de fişiere e folosit şi de Yann LeCun ([40]) şi am dorit să tratăm unitar toate fişierele cu imagini şi etichete. Tipul fişierelor *.idx* este prezentat în Anexele 1 şi 2.

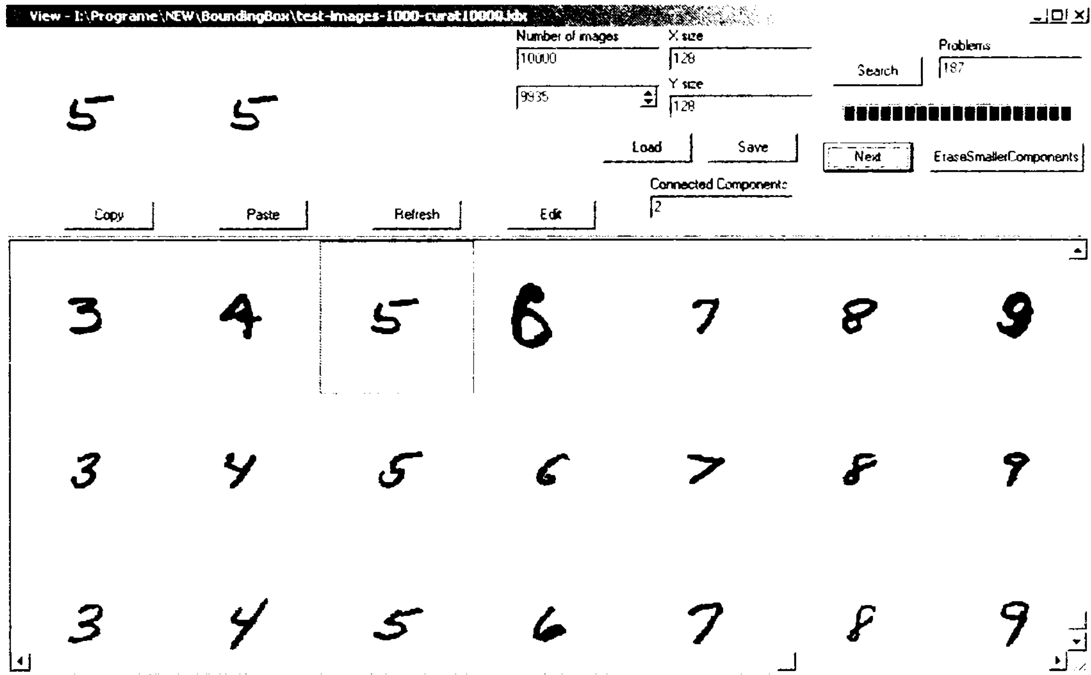


Figura 4.9: Interfaţa programului de corectare semiautomată.

După prelucrările anterioare, rezultatul este un fişier cu imaginile a 10000 de cifre provenite de la 1000 de persoane diferite. Fiecare rând conţine imagini de la o singură persoană. În Figura 4.10 este arătat modul de codificare. Cu aceste zece cifre ale fiecărei persoane construim 200 de imagini cu perechi de cifre parţial suprapuse. Imaginile cu indici pari conţin cifre suprapuse orizontal cu un pixel, iar cele cu indici impari conţin cifre suprapuse orizontal cu 3 pixeli. Am folosit suprapuneri orizontale doar pe 1 şi 3 pixeli, deoarece mai puţin de 1 înseamnă doar atingere - caz uşor rezolvat de un clasificator la nivel de cifra care e aplicat pe o fereastră ce se deplasează deasupra imaginii -, iar mai mult de trei (3 reprezentând 15% într-o imagine de 20×20 pixeli) e puţin probabil să întâlnim într-un text scris de mână. Cazul suprapunerii pe o distanţă de doi pixeli a fost evitat deoarece e prea asemănător cu celealte două şi doar ar fi contribuit la mărirea inutilă cu 50% a setului de antrenare şi, implicit, a timpului necesar antrenării.

În direcţie verticală ambele imagini de cifre au fost deplasate aleator cu zero sau un pixel, în sus sau în jos. În acest mod s-a dorit simularea scrierii nealiniate orizontal din scrisul de mână. Pixelii obţinuţi în urma suprapunerii au ca valoare a nivelului de gri minimul dintre 255 şi suma celor două nivele de gri ale pixelilor ce au fost suprapuşi. Înainte de procesul de suprapunere parţială fiecare imagine de 20×20 pixeli a fost centrată într-o imagine de 28×28 pixeli pentru a permite deplasările orizontale şi verticale. Imaginea rezultată în urma unirii a două cifre are dimensiunea de 40×28 pixeli (Figura 4.11). Acestea sunt datele brute care vor fi redimensionate ulterior pentru antrenarea şi testarea reţelelor neuronale. Am ales

BUPT

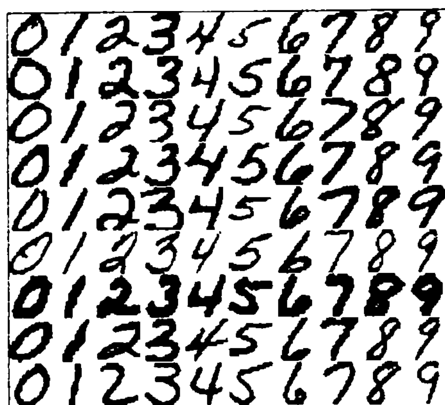


Figura 4.10: Cifrele a nouă persoane după etapele de tăiere după dreptunghiul delimitator, centrare şi redimensionare la 20×20 pixeli.

lăţimea de 40 de pixeli deoarece imaginile originale aveau 20 de pixeli, din acest motiv în urma procesului de suprapunere parţială nu putea rezulta o imagine mai lată de 40 de pixeli. Înălţimea stabilită la 28 de pixeli permite deplasarea pe verticală a imaginilor cu patru pixeli, atât în sus, cât şi în jos. Setul de antrenare obţinut conţine 200000 de imagini, câte 2000 pentru fiecare din cele 100 de clase.

4.4.2 Setul de test

Acest set este compus din persoane începând cu poziţia 1010 în partiţiile $hsf_0 \div hsf_3$. Din motive similare cu cele de la setul de antrenare, formularele următoarelor persoane au fost ignorate: 1102, 1107, 1110, 1209, 1432, 1571, 1572, 1723, 1725, 1726, 1727, 1756, 1767, 1965, 1969, 1974, 1984, 1990, 1992 şi 1995. Am aplicat procesul semiautomat de îndepărtare a pixelilor care ar fi afectat nedorit procesul de unire a cifrelor. Pe aceste seturi am îmbunătăţit doar aproximativ 10 imagini care au fost prea distorsionate pentru a fi interpretate de un om. Nu am făcut alte îmbunătăţiri ale calităţii imaginilor din acest set deoarece am dorit să se poată observa capacitatea de generalizare a reţelei. Un alt motiv a fost acela că s-a dorit testarea reţelei cu imagini cât mai apropiate de cele întâlnite în realitate. Restul paşilor de procesare sunt identici cu cei de la setul de antrenare:

- tăierea imaginii la limitele dreptunghiului delimitator al cifrei,
- centrarea imaginii,
- redimensionarea la 20×20 pixeli,
- deplasări orizontală şi verticală,
- unirea a două imagini într-una de 40×28 pixeli.

4.5 Seturile de test 3DNS, 6DNS şi 10DNS

Întrucât lucrarea de faţă tratează subiectul recunoaşterii şirurilor de cifre scrise de mână, sunt necesare seturi de test cu astfel de şiruri. În literatură se folosesc şirurile din NIST SD 19 [4]

BUPT

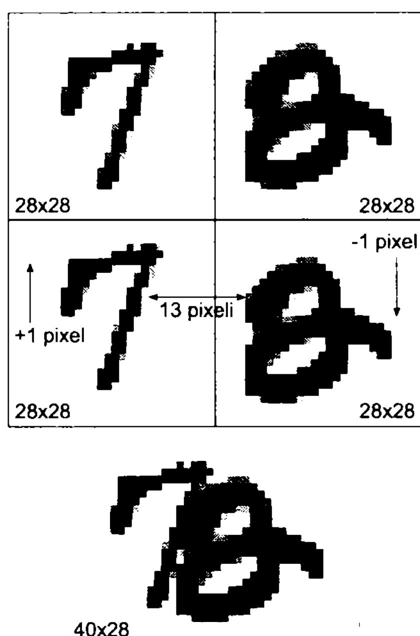


Figura 4.11: Procesul de suprapunere parţială: primul rând - imaginile originale; rândul al doilea - imagini deplasate pe verticală; al treilea rând - imaginea finală după deplasarea pe orizontală cu 13 pixeli (distanţa minimă dintre imagini) + 3 (distanţa de suprapunere) pixeli.

şi din CEDAR [22]. Deoarece am avut la dispoziţie doar NIST SD 19, testele au fost efectuate pe această bază de date.

NIST SD 19 conţine gata extrase cifrele şi literele din cele câteva mii de formulare. Nu conţine, însă, şirurile de cifre, astfel că a trebuit scris un program care să le extragă din imaginile cu formulare (Figura 4.2).

Cu metoda descrisă în secţiunea următoare am extras câmpurile de 3, 6 şi 10 cifre din formularele 1800-2099. Din cele 300 de pagini cu formulare, autorii din [44] au eliminat patru pagini cu imagini foarte şterse: 1965, 1977, 2027 şi 2067. Au fost eliminate şi în cadrul acestor experimente, rămânând 296 de imagini de formulare.

4.5.1 Metoda de extragere a câmpurilor din formulare

Studiind formularele de tip HSF (un exemplu este dat în Figura 4.2), se observă că sunt alcătuite din câmpuri dreptunghiulare în care persoana care le completează trebuie să scrie informaţia tipărită deasupra câmpului respectiv. Se cunoaşte că toate formularele arată aproximativ la fel: au acelaşi număr de câmpuri de aceeaşi dimensiune, poziţionate în aceleaşi locuri. Diferă numai conţinutul câmpurilor, iar informaţia cu cifrele sau literele ce ar trebui să se regăsească în acele câmpuri se află în fişierul cu referinţe corespunzător formularului. Cunoscând aceste detalii, extragerea imaginilor cu şirurile de cifre pare iniţial o operaţie destul de simplă. Doar că formularele au fost întâi tipărite, apoi completate mai mult sau mai puţin corect şi apoi scanate. În urma acestor prelucrări dreptunghiurile nu se mai află în aceleaşi poziţii; liniile au grosimi diferite şi pot fi întrerupte; distanţele dintre câmpuri nu

BUPT

mai sunt egale între ele; toată imaginea poate fi rotită cu un unghi de câteva grade faţă de orizontală, mai mult, acest unghi poate varia în interiorul documentului; scrisul poate depăşi dreptunghiurile ce delimitează câmpurile.

Deoarece câmpurile din formulare sunt dispuse pe rânduri, iar dreptunghiurile aferente sunt aliniate, având aceeaşi înălţime, un mod de a începe extragerea informaţiei ar putea fi reprezentat de detectarea acestor linii cvasiorizontale. În formular există 20 de astfel de linii (Figura 4.2), delimitând 10 rânduri cu informaţie. Doar rândurile 1, 2, 3, 4, 5 şi 6 conţin şiruri de cifre, restul neprezentând interes. Un mod simplu de a detecta liniile dintr-o imagine este cu ajutorul histogramelor. Dezavantajul acestora este acela că funcţionează doar pentru un unghi care trebuie cunoscut a priori. Deoarece imaginile pot fi rotite cu orice unghi -în urma experimentelor am constatat că unghiul de rotire al formularelor este de maxim $\pm 1.5°$-, această metodă nu poate fi aplicată. Deşi unghiul pare mic, luând în considerare că o linie are lungimea de aproximativ 2000 de pixeli, $1.5°$ înseamnă o abatere pe verticală de aproximativ $2000 \cdot \tan 1.5° \approx 52$ de pixeli. Cum grosimea liniei scanate la 300dpi este de aproximativ 8-10 pixeli, devine evident de ce metoda histogramelor orizontale nu funcţionează. Soluţia ar reprezenta-o mai multe histograme cvasiorizontale cu abaterea faţă de orizontală de maxim $\pm 1.5°$, însă există o altă metodă -transformata Hough- care rezolvă aceste probleme. În secţiunea 3.1.3 au fost prezentate aspecte teoretice generale referitoare la această transformată.

Având în vedere că sunt foarte multe formulare de prelucrat, fiecare având dimensiunile de 3300×2560 pixeli, şi că transformata Hough este foarte mare consumatoare de resurse de calcul, aplicarea ei pare a fi neoportună. Se ştie că transformata Hough va lua fiecare din cei 8448000 de pixeli ai imaginii şi va testa să vadă dacă se află pe una din foarte multele drepte posibile. În mod normal parametrii transformatei Hough (a se vedea Sectiunea 3.1.3) ar lua următoarele valori: $\theta \in [-90°, +90°)$, iar $r \in [-3300, +4177]$. Dacă am alege paşii de discretizare $dr = 1pixel$ şi $d\theta = 0.2°$ (e necesar un astfel de pas mic pentru θ deoarece liniile orizontale ale documentului sunt foarte lungi -de aproximativ 2000 de pixeli- şi un pas mai mare ar conduce la abateri prea mari care ar implica eşecul detectării liniilor din imagine), observăm că trebuie să verificăm apartenenţa fiecăruia din cele 8448000 de puncte la fiecare din cele $\frac{(3300+4177)pixeli}{1pixel} \times \left(\frac{180°}{0.2°} + 1\right) = 6736777$ drepte. Ajungem la $8448000 \times 901 = 7611648000$ aplicări ale Formulei 3.7 pentru un singur formular, mult prea mult chiar şi pentru procesoarele actuale.

Evident, sunt necesare îmbunătăţiri ale algoritmului. Pasul de discretizare dr nu poate fi mărit deoarece am începe să detectăm puncte din grosimea liniilor, ceea ce ar duce la alte complicaţii. Mai rămân doi parametri, intervalele de valori pentru r şi pentru θ. Am putea căuta linii orizontale numai în anumite zone ale documentului, dar o matrice de acumulare cu indici ce variază necontinuu iarăşi complică mult algoritmul. Pe lângă acest motiv, e destul de "periculos" să alegem doar anumite benzi orizontale din imagine, deoarece imaginile pot fi translatate pe verticală destul de mult, chiar şi cu câteva sute de pixeli. Ramâne parametrul θ, care, din fericire, poate fi limitat la intervalul $[-1.5°, +1.5°)$, deoarece cunoaştem ca imaginile formularelor nu pot fi rotite mai mult de atât. Această limitare a domeniului de valori pentru θ măreşte viteza algoritmului de $\frac{180/0.2+1}{3/0.2+1} \approx 56$ ori, aducând timpul de procesare la nivelul a câteva secunde pentru un formular (folosind un singur fir de execuţie).

O optimizare a fost adusă şi parametrului r, prin limitarea variaţiei sale. Analizând Figura 4.12 se poate observa că pentru variaţii de $\pm 1.5°$ ale lui θ putem limita variaţia lui r doar la domeniul pozitiv dacă nu mai considerăm posibilele drepte ce ar trece numai prin puncte ale

BUPT

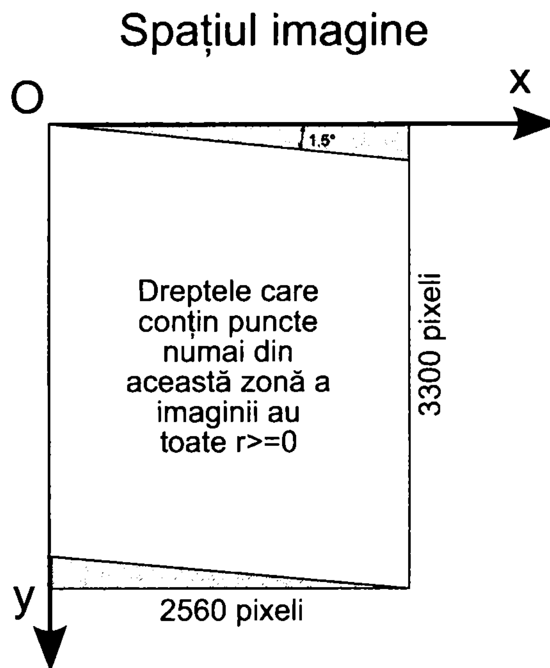


Figura 4.12: Limitarea variaţiei parametrului r.

zonei haşurate din Figura 4.12. Cum informaţiile din formulare se află cu câteva sute de pixeli mai jos, nu afectăm deloc corectitudinea algoritmului. În schimb, va creşte viteza de rulare şi codul va fi puţin mai simplu din cauză că nu mai există şi zone negative pentru indici.

Algoritmul 1 prezintă modul de obţinere a matricei de acumulare. Fiecare element al matricei are ca valoare numărul de pixeli negri ai imaginii prin care trece respectiva linie. Folosind informaţiile prezentate anterior se poate calcula dimensiunea matricei de acumulare: $4177 \times 16 = 66832$. Foarte multe din valorile matricei sunt nenule, astfel că trebuie să găsim o metodă prin care să selectăm doar acele elemente care corespund liniilor căutate. O metodă simplă este aceea de a aplica un filtru care să decidă dacă fiecare valoare a matricei reprezintă o dreaptă utilă sau nu. Arătam mai sus că abaterea maximă pentru o linie de 2000 de pixeli la o înclinaţie de $1.5°$ este de 52 pixeli. Putem aplica o fereastră de forma unui pătrat cu latura de $52 \times 2 + 1 = 105$ pixeli centrată în fiecare pixel al matricei de acumulare. Dacă în această fereastră există cel puţin o valoare mai mare decât cea din centrul ferestrei, valoarea elementului central este setată pe 0. Deşi e o metodă foarte primitivă, este eficientă şi este suficient de rapidă pentru a nu necesita implementarea unor algoritmi mai complicaţi. Timpul necesar procesării întregii matrici este de aproximativ o secundă. Pe lângă procesarea decrisă am mai adăugat condiţia ca fiecare linie să conţină cel puţin 800 de pixeli. Acest prag a fost obţinut studiind suma lungimilor segmentelor coliniare din formulare.

Rezultatul operaţiilor de până acum este reprezentat de câteva zeci de linii candidate. Analizându-le am constatat că pe lângă liniile ce se doreau a fi detectate apăreau şi câteva linii "fantomă". Acestea erau detectate în textul tipărit al paragrafului din Constituţie. Deoarece fontul de acolo era foarte gros şi adunat, într-adevăr existau câteva linii care se puteau forma cu pixeli din caracterele tipărite. De obicei se detecta o linie pentru fiecare rând al paragra-

BUPT

Input: Imagine
Output: matricea de acumulare H
iniţializează toate elementele din H cu 0
foreach $0 \leq y <$ *ÎnălţimeImagine* **do**
  foreach $0 \leq x <$ *LăţimeImagine* **do**
    if $Imagine[y, x] \neq 0$ **then**
      foreach $0 \leq t < t_{max}$ **do**
        $\theta = (0.2t - 1.5)\pi/180$
        $r = x \sin\theta + y \cos\theta$
        if $0 \leq r < 4177$ **then**
          incrementează $H[t, r]$
        end
      end
    end
  end
end

Algorithm 1: Transformata Hough.

fului. Transformata Hough doar **numără** punctele care se află pe o linie, nu verifică **continuitatea** ei, neavând nicio informaţie referitoare la distribuţia punctelor pe dreaptă. Din acest motiv gradul de fragmentare al liniilor detectate în paragraful Constituţiei nu reprezintă un impediment în detectarea lor. Se impune, însă, eliminarea acestor linii false din prisma scopului metodei de extragere a imaginilor cu şiruri de cifre. Pentru aceasta este suficient să măsurăm gradul de fragmentare al liniilor şi să le eliminăm pe cele care depăşesc un anumit prag maxim admis.

Gradul de fragmentare al unei linii poate fi măsurat calculând numărul de discontinuităţi al respectivei linii, folosind Algoritmul 2. Algoritmul reface linia, prin căutarea în spaţiul imaginii originale a pixelilor de pe dreapta cu parametrii r şi θ în spaţiul Hough. Sunt numărate trecerile $0 \rightarrow 1$ şi $1 \rightarrow 0$, adică trecerile de la pixeli albi la negri şi invers. Acest număr e apoi împărţit cu 2 pentru a obţine numărul de discontinuităţi.

Input: Imagine, dreapta (r, θ)
Output: numărul de discontinuităţi ale dreptei curente
treceri=0
foreach $0 \leq x <$ *LăţimeImagine-1* **do**
  $y_1 = r - \frac{x \sin\theta\frac{\pi}{180}}{\cos\theta\frac{\pi}{180}}$
  $y_2 = r - \frac{(x+1)\sin\theta\frac{\pi}{180}}{\cos\theta\frac{\pi}{180}}$
  if $Imagine[y_1, x] \neq Imagine[y_2, x+1]$ **then**
    incrementează treceri
  end
end
return $\frac{treceri}{2}$

Algorithm 2: Determinarea numărului de discontinuităţi ale unei drepte.

BUPT

După experimentarea algoritmului pe câteva formulare am ajuns la un număr maxim admisibil de întreruperi egal cu 60 pentru liniile candidate la a delimita câmpuri ale formularului. Toate liniile cu mai mult de 60 de discontinuităţi sunt ignorate, ele apărând în paragraful tiparit al Constituţiei. În acest moment numărul de linii încă este mai mare decât numărul de linii orizontale care delimiteaza rândurile cu câmpurile formularului. Liniile detectate sunt mai multe deoarece unele dintre ele sunt multiplu detectate, adică există câteva linii cu exact acelaşi număr de puncte şi cu parametri foarte apropiaţi. De obicei diferă numai prin parametrul r, existând linii multiplicate de 2-5 ori. Dintre aceste linii ne este suficientă una singură. Folosind Algoritmul 3 păstrăm doar liniile care delimitează rânduri cu câmpuri din formular. Dacă există mai multe linii consecutive detectate, atunci o păstrăm pe prima, dacă suntem la începutul unui rând, respectiv doar pe ultima, dacă suntem la sfârşitul unui rând. Fiecare linie conţine informaţii despre r, θ şi numărul de puncte. Varibila $start$ are valoarea $true$ dacă se caută o linie pentru începutul unui rând, şi $false$ dacă se caută o linie pentru sfârşitul unui rând.

Input: $L[]$, vector cu liniile detectate
Output: $D[]$, vector cu liniile păstrate
$start$=true //început de rând
foreach *linie* $L[i]$ *din* $L[]$ **do**
  află lista de linii $L[i+j]$ cu proprietatea că $|L[i].r - L[i+j].r| \leq 10$
  if *j=0* **then**
    adaugă $L[i]$ la $D[]$ //nu există linii alăturate
  end
  else
    if *start* **then**
      adăugăm $L[i+j]$ la $D[]$ //o copiem pe ultima
    end
    else
      adăugăm $L[i]$ la $D[]$ //o copiem pe prima
    end
  end
  $start = not(start)$ //schimbăm starea
end

Algorithm 3: Selectarea liniilor ce delimitează rânduri cu câmpuri din formular.

În $D[]$ vom avea 20 de linii care delimitează cele 10 rânduri de câmpuri. Întrucât ne interesează numai câmpurile numerice, păstrăm doar liniile cu indici de la 2 la 13, linii care ar trebui să delimiteze cele cinci rânduri de câmpuri numerice.

Fiecare pereche de linii din cele 10 linii rămase, este trimisă unui modul de extragere a imaginilor câmpurilor numerice. Se verifică dacă numărul de puncte de pe cele două linii nu diferă cu mai mult de 200 de puncte şi dacă $||r_1 - r_2| - 90| > 20$, deoarece înălţimea medie a unui rând este de 90 de pixeli. Dacă vreunul din aceste teste nu este trecut, procesarea continuă cu următoarea pereche de linii. Urmează delimitarea câmpurilor pe orizontală. Pentru aceasta se reformează cele două linii în spaţiul imagine şi apoi se suprapun. Avem o singură linie care ar trebui să prezinte discontinuităţi în spaţiile dintre câmpuri. Câmpurile din formulare sunt separate pe orizontală de cel puţin 55 de pixeli albi. Din acest motiv am ales ca prag 40 de pixeli pentru discontinuitatea maximă permisă. Linia anterioară este segmen-

BUPT

tată în fiecare poziţie unde începe o discontinuitate mai mare de 40 de pixeli. Bineînţeles, se tratează şi cazurile particulare de început şi de sfârşit de linie. Primul rând cu câmpuri numerice conţine 3 câmpuri, iar următoarele 5 conţin câte 5 câmpuri. Numărul de câmpuri detectate este comparat cu aceste constante pentru a se detecta eventuale erori de segmentare. Dacă valorile se potrivesc, atunci se apelează un alt modul care extrage din formular imaginea rectangulară cu dimensiunile şi poziţia detectate anterior. Se construieşte şi un nume sugestiv pentru imaginea câmpului, de forma f*xxxx*_*yy*_fld*zz*_*eticheta*.bmp, unde *zz* reprezintă numărul câmpului, iar *eticheta* - eticheta respectivului câmp. Eticheta este citită din fişierul cu etichete, aferent fiecărui formular. Imaginea încă este în formă brută, conţinând în unele cazuri părţi din dreptunghiul ce delimita câmpul. Primul rând din Figura 4.13 prezintă câteva astfel de imagini înainte de procesul de curăţare.

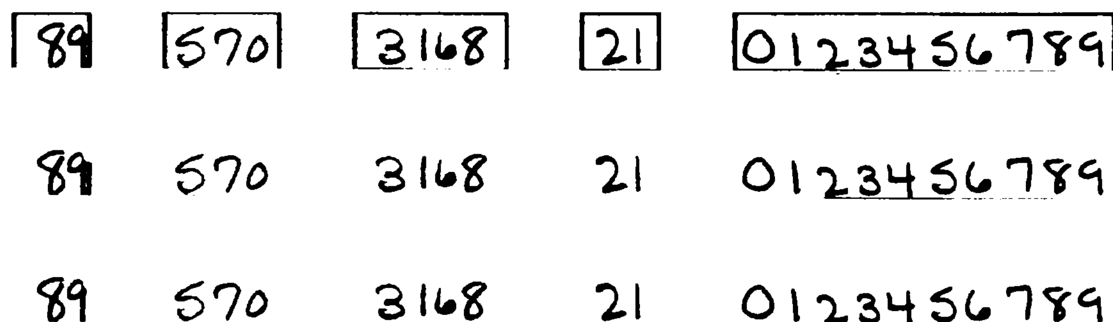


Figura 4.13: Imagini obţinute în urma procesului de extragere a câmpurilor din formulare. Primul rând - imagini brute, al doilea rând - imagini după faza de curăţare automată, rândul al treilea - imagini după faza de curăţare manuală.

Grosimea liniilor care formează dreptunghiurile de delimitare a câmpurilor este de aproximativ 10 pixeli. Am luat o marjă de 50% şi am considerat un dreptunghi interior imaginii, depărtat la 15 pixeli de extremităţi. Ideea constă în ştergerea condiţionată a pixelilor din acest cadru dreptunghiular. Se ia fiecare pixel din această cadru. Dacă este negru, atunci se caută componenta conexă din care face parte. Dacă această componentă conexă nu conţine pixeli din afara cadrului dreptunghiular, atunci pixelul împreună cu toată componenta conexă pot fi şterşi. Determinarea componentei conexe se face în mod uzual prin căutare în opt direcţii, dar condiţionat de a nu se depăşi zona cadrului dreptunghiular. Această metodă simplă rezolvă peste 80% din cazurile de imagini brute cu resturi ale dreptunghiurilor ce delimitează câmpurile (Figura 4.13, rândul din mijloc). Restul de imagini sunt curăţate manual deoarece textul din interior atinge sau suprascrie dreptunghiul delimitator (Figura 4.13, ultimul rând).

4.5.2 Setul de test 3DNS

Setul de test 3DNS este folosit pentru a efectua o comparaţie exactă între metoda propusă în această lucrare şi cele prezentate în [44]. Din acest motiv am extras exact aceleaşi imagini ca în [44]. Formularele conţin câte 5 câmpuri de trei cifre, în total existând $296 \times 5 = 1480$ imagini de numere formate din trei cifre. În [44] se afirmă că au fost eliminate patru imagini care nu corespundeau etichetelor, dar nu se specifică care sunt aceste imagini. Analizând vizual imaginile detectate eronat de clasificator am identificat patru imagini care, într-adevar, nu corespundeau cu etichetele lor (Figura 4.14). Imaginile greşit scrise sunt: formularul 1902

BUPT

- câmpul 7, formularul 2035 - câmpul 28, formularul 2056 - câmpul 11, formularul 2076 - câmpul 11. Deci, la final, setul de test 3DNS conţine 1476 de imagini cu numere de trei cifre. Toate aceste imagini au fost tăiate la dimensiunea dreptungiului delimitator.

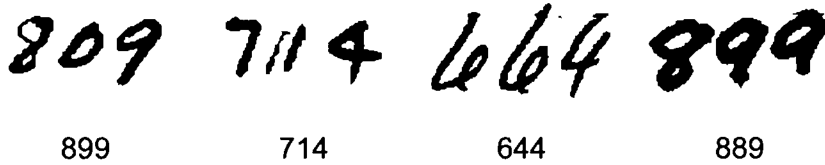


Figura 4.14: Imaginile cu numere de trei cifre care au fost scrise eronat de subiecţii din NIST SD 19.

4.5.3 Setul de test 6DNS

Şi setul de test 6DNS este folosit pentru a compara metoda propusă cu cele din [44], astfel că l-am compus din aceleaşi imagini. Mai exact, din cele 296 de formulare (1800-2099, minus cele patru scrise foarte şters) am extras toate câmpurile de 6 cifre. Fiind câte cinci câmpuri de şase cifre pe formular am obţinut 1480 de imagini de numere a şase cifre. Ca în [44], am eliminat imaginile ce nu conţineau textul scris deasupra câmpului. Am identificat opt exemple clare de astfel de erori, al nouălea -rândul doi, coloana trei- a fost ales deoarece autorii din [44] afirmă că există nouă astfel de imagini eronate (Figura 4.15). Am ales cea mai defectuos scrisă imagine, deoarece în [44] nu se specifică indicii imaginilor. Întrucât detectarea acestor imagini este laborioasă şi plictisitoare, pentru referinţe următoare, voi menţiona indicii imaginilor eronate: formularul 1910 câmpul 26, formularul 1913 câmpul 26, formularul 1937 câmpul 18, formularul 2003 câmpul 22, formularul 2035 câmpul 22, formularul 2043 câmpul 14, formularul 2070 câmpul 18, formularul 2080 câmpul 22 şi formularul 2094 câmpul 18.

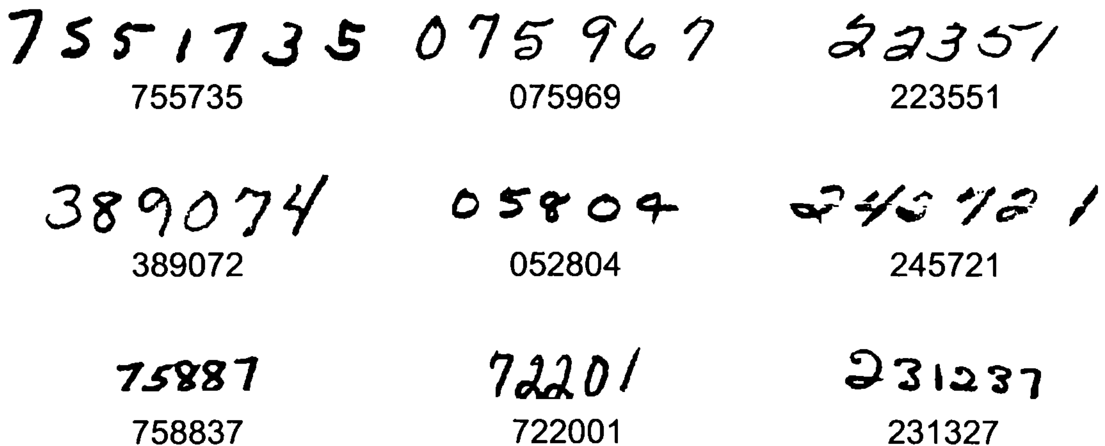


Figura 4.15: Imagini cu numere de şase cifre, eronat scrise de subiecţii NIST SD 19.

La final setul de test 6DNS este compus din 1471 de imagini cu numere formate din şase cifre. Toate imaginile au fost tăiate la dimensiunea dreptunghiului delimitator.

BUPT

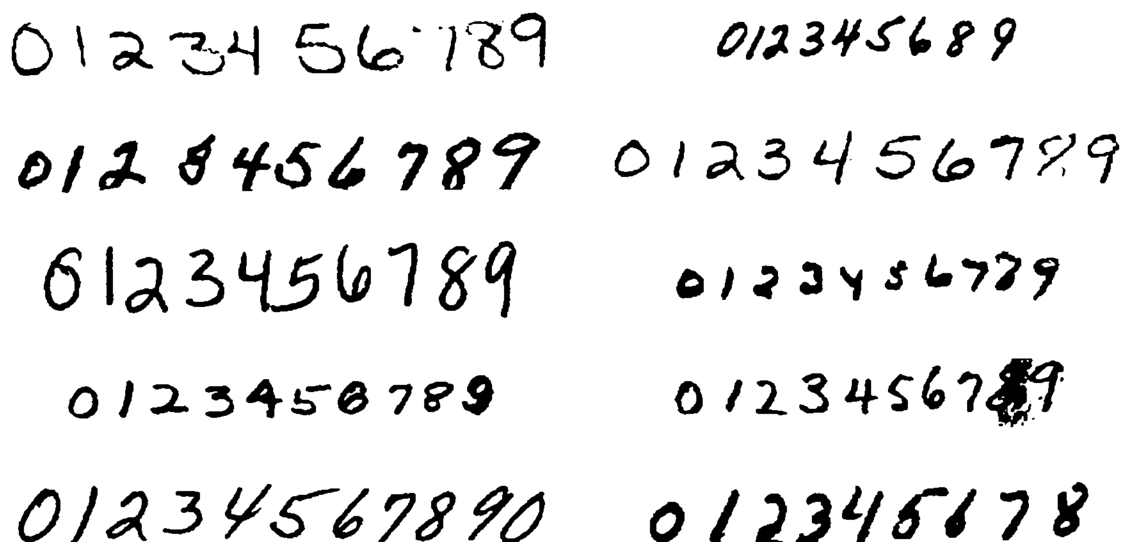


Figura 4.16: Imagini cu numere de zece cifre, eronat scrise de subiecţii NIST SD 19 (toate ar trebui să conţină şirul 0123456789).

4.5.4 Setul de test 10DNS

Din cele 296 de formulare am extras $296 \times 3 = 888$ de imagini cu numere formate din 10 cifre, fiecare formular conţinând trei câmpuri a zece cifre. Din cele 888 de imagini, zece nu corespund cu informaţia scrisă deasupra respectivului câmp şi au fost eliminate (Figura 4.16). Imaginile eronate au următorii indici: formularul 1858 câmpul 4, formularul 1896 câmpul 4, formularul 1909 câmpul 3, formularul 1957 câmpul 4, formularul 1975 câmpul 4, formularul 2021 câmpul 3, formularul 2023 câmpul 3, formularul 2038 câmpul 3, formularul 2039 câmpul 3 şi formularul 2049 câmpul 5.

După excluderea imaginilor eronate au rămas 878 de imagini în setul de test 10DNS. La fel ca pentru seturile 3DNS şi 6DNS, imaginile au fost tăiate la dimensiunea dreptunghiului delimitator.

4.6 Concluzii

Acest capitol a avut drept scop crearea de seturi de imagini pentru antrenarea şi testarea clasificatorilor. De asemenea, au fost prezentate bazele de date NIST SD 19 şi MNIST.

Pentru antrenarea clasificatorului la nivel de cifră au fost create seturile din LNIST (66214 imagini pentru setul de antrenare şi 45398 de imagini pentru setul de test) prin extragerea de imagini din NIST SD 19. Acestea sunt construite identic cu cele folosite de autorii din [44, 45] pentru a permite o comparaţie cât mai riguroasă a rezultatelor.

Pentru antrenarea clasificatorului la nivel de două cifre parţial suprapuse (2RNC) au fost create seturile de antrenare şi de test TDNS. Din NIST SD 19 au fost extrase 20000 de imagini de cifre (10000 pentru antrenare şi 10000 pentru test) din care s-au creat câte 200000 de imagini cu perechi de cifre parţial suprapuse atât pentru setul de antrenare, cât şi pentru setul de test. Pentru formarea imaginilor cu perechi de cifre parţial suprapuse a fost dezvoltat

BUPT

un algoritm care uneşte cifrele în funcţie de morfologia acestora.

Dacă imaginile setului LNIST au fost extrase pentru a fi folosite de către clasificatorul la nivel de cifră, pentru cazul general, cel al şirurilor numerice formate din mai multe cifre, au fost construite seturile de test 3DNS, 6DNS şi 10DNS. Şi în construirea acestor seturi s-a urmărit obţinerea unor seturi cât mai apropiate de cele folosite de alţi cercetători pentru evaluarea performanţelor metodelor de recunoaştere. Cele trei seturi sunt identice cu cele folosite în domeniu, în limita informaţiilor furnizate de autorii care le-au utilizat anterior. Există posibilitatea ca una, maxim două imagini din cele aproape 1500 să fie diferite.

Pentru extragerea din NIST SD 19 a imaginilor necesare construirii seturilor 3DNS, 6DNS şi 10DNS a fost elaborat un algoritm care foloseşte o variantă a transformatei Hough pentru detectarea liniilor din formularele NIST. După detecţia acestor linii, sunt detectate dreptunghiurile ce încadrează câmpurile dorite. Apoi are loc extragerea propriu-zisă a imaginilor şi se aplică un algoritm de înlăturare automată a posibilelor părţi din dreptunghiul delimitator care au fost extrase în acelaşi timp cu imaginile şirurilor numerice. La final a fost aplicat un proces de inspectare şi curăţare manuală a imaginilor. Algoritmul de extragere automată a imaginilor din câmpurile formularelor a diminuat considerabil timpul necesar construirii pe cale manuală a celor trei seturi de test. De asemenea, algoritmul este destul de general, putând fi adaptat facil pentru alte tipuri de formulare.

BUPT

Capitolul 5

Experimente

În acest capitol vor fi evaluate performanţele reţelelor neuronale de tip perceptron şi de convoluţie prezentate în Capitolul 3. Întâi, reţelele vor fi antrenare şi apoi vor fi evaluate folosind seturile de date din Capitolul 4. Se va începe cu evaluarea reţelelor pentru recunoaşterea cifrelor individuale, iar apoi se vor antrena reţele pentru recunoaşterea de numere formate din două cifre parţial suprapuse. După ce ambii clasificatori - pentru cifre singulare şi pentru numere formate din două cifre parţial suprapuse - vor fi antrenaţi şi testaţi, se va implementa o metodă pentru recunoaşterea numerelor formate din 3 sau mai multe cifre. Această metodă va fi testată cu seturile 3DNS, 6DNS şi 10DNS construite în Secţiunile 4.5.2, 4.5.3 şi 4.6.

Reţelele neuronale şi toate programele adiţionale pentru procesarea imaginilor au fost implementate în C++. Am folosit şi cod open source [3] pe care l-am modificat pentru a putea fi folosit la experimentele pe care le-am efectuat. Pentru toţi neuronii reţelei am folosit ca funcţie de activare tangenta hiperbolică $\frac{e^{2x}-1}{e^{2x}+1}$. Am obţinut o creştere semnificativă a vitezei de antrenare prin paralelizarea algoritmului pentru a folosi suportul nativ existent în procesoarele multinucleu existente. Am testat algoritmii pe procesoare cu două şi cu patru nuclee.

5.1 Experimente cu cifre individuale

Aceste experimente au o dublă motivaţie. Prima este dată de evaluarea corectitudinii implementării algoritmilor prin compararea rezultatelor cu cele existente deja, iar a doua de obţinerea unui clasificator de calitate (care are rată mare de recunoaştere) la nivel de cifre singulare. Acest clasificator va fi utilizat apoi în sistemul de recunoaştere (a se vedea Figura 5.4) care recunoaşte şiruri de una, două sau mai multe cifre distincte sau parţial suprapuse.

Ca seturi de antrenare au fost folosite seturile MNIST şi LNIST. La prima vedere, poate părea inutilă folosirea a două seturi de antrenare. Scopul acestei lucrări este de a propune o nouă metodă de recunoaştere a şirurilor scrise de mână. Ca orice metodă nouă, trebuie validată prin efectuarea de experimente ale căror rezultate trebuie comparate cu cele existente deja în literatura de specialitate. Arătam în introducere că articolul [44] prezintă cele mai bune rezultate obţinute până în acest moment. Autorii din [44] folosesc ca set de antrenare pentru clasificatorul la nivel de cifră setul LNIST, astfel că pentru o comparaţie corectă a metodelor am antrenat şi clasificatorul din metoda proprie cu acest set. În plus, setul LNIST

BUPT

are proprietatea că este disjunct cu seturile de test 3DNS, 6DNS şi 10DNS. Setul MNIST a fost folosit pentru a verifica corectitudinea implementării algoritmilor de antrenare a reţelelor neuronale prin compararea rezultatelor cu cele din [40]. Din păcate, setul de antrenare din MNIST conţine imagini cu cifre din toate partiţiile *hsf_{0-4}*, deci nu este disjunct cu seturile de test 3DNS, 6DNS şi 10DNS. Din acest motiv nu poate fi folosit la experimentele ce implică seturile 3DNS, 6DNS şi 10DNS.

5.1.1 Reţele neuronale de tip perceptron

Întrucât perceptronul fără straturi ascunse nu poate fi folosit la clasificarea elementelor mulţimilor în clase neliniar separabile, am folosit perceptronul cu un strat ascuns. Am variat numărul de neuroni de pe stratul ascuns între 60 şi 220 (Tabelul 5.1). Antrenarea a fost efectuată cu setul de antrenare LNIST (Sectiunea 4.3), iar testarea cu cu cele 10000 de imagini din setul de test MNIST (Secţiunea 4.2). Puteam folosi şi setul de test LNIST, dar, după cum se va vedea în secţiunea următoare, rezultatele sunt foarte asemănătoare. Nu am mărit numărul de neuroni de pe stratul ascuns mai mult de 220 pentru că timpul ajunsese la aproximativ 600 de secunde pentru o epocă, iar eroarea nu scădea decât foarte încet.

Tabelul 5.1: Rezultate obţinute cu perceptronul cu un strat ascuns.

# neuroni pe stratul ascuns	eroarea pe setul de antrenare LNIST(%)	eroarea pe setul de test MNIST(%)
60	2.08	4.83
80	1.59	4.02
100	1.39	3.68
120	1.31	3.54
140	1.40	3.58
160	1.26	3.70
180	1.32	4.11
200	1.23	3.63
220	1.26	3.46

Toate reţelele au fost antrenate fără activarea distorsiunilor pe setul de antrenare. Am variat rata de învăţare, η, de la 0.0001 până la 0.000005, înmulţind-o la fiecare epocă cu factorul 0.85. După atingerea valorii minime de 0.000005, η a rămas constant. Când numărul de imagini eronat clasificate din setul de antrenare s-a stabilizat, am oprit antrenarea.

Pentru a verifica dacă aplicarea distorsiunilor ar influenţa mult rata de recunoaştere pentru setul de test, am antrenat cea mai mare reţea -cea cu 220 de neuroni pe stratul ascuns- şi cu activarea distorsiunilor pe setul de antrenare. Eroare a scăzut până la 2.88%, dar această eroare este prea mare faţă de cele mai bune rezultate raportate în [40]. În plus, a crescut timpul de antrenare cu aproximativ 100 de secunde pe epocă.

5.1.2 Reţele neuronale cu extractori de trăsături

Cum perceptronul nu a oferit rate de recunoaştere suficient de bune, am trecut la experimentarea unor reţele de convoluţie despre care există date [64, 40] că oferă rate de recunoaştere mult mai bune. Detalii despre arhitectura reţelelor de convoluţie se află în Capitolul

BUPT

3. Am variat cei trei parametri ai reţelei în jurul valorilor despre care Simard scria în [64] că au oferit cea mai bună rată de recunoaştere. Am refăcut şi experimentul lui Simard pe setul MNIST şi am obţinut eroarea minimă pentru setul de test de 0.66%. Simard obţinuse 0.4% ceea ce este foarte apropiat de 0.66%. Dacă reţeaua ar fi fost lăsată să se antreneze mai mult, probabil ajungeam la eroarea lui Simard. Însă, scopul era doar verificarea corectitudinii algoritmilor de învăţare, nu obţinerea unui minim cât mai mic, deoarece, oricum, nu puteam folosi reţeaua antrenată cu setul MNIST, acesta nefiind disjunct cu seturile de test multicifră ce urmau a fi folosite la clasificatorul general.

Am antrenat reţelele cu setul de antrenare LNIST, iar rezultatele se află în Tabelul 5.2. Am testat fiecare reţea şi cu setul de test din MNIST pentru a vedea dacă rezultatele sunt asemănătoare cu cele obţinute pe setul de test LNIST. Rezultatele testării MNIST sunt puţin mai bune -cu maxim 0.22%-, ceea ce se explică prin faptul ca setul de test MNIST are imagini şi din setul de antrenare LNIST. Cel mai bun rezultat este obţinut cu o reţea cu 6 hărţi de convoluţie pe stratul L_1, 50 de hărţi de convoluţie pe stratul L_2 şi 60 de neuroni pe stratul L_3.

De această dată am urmat recomandările lui Simard [64] şi am antrenat reţelele aplicând distorsiuni asupra setului de antrenare. Valorile maxime ale acestora sunt: 15% pentru factorul de scară, 15% pentru factorul de rotaţie, 8 pentru σ pentru distorsiuni elastice, şi 0.5 pentru factorul de scară pentru distorsiuni elastice. Distorsiunile au fost aplicate fiecărei imagini înainte de a fi trimisă modulului de antrenare. Rata de învăţare, η, a pornit de la 0.0001 şi am scăzut-o la fiecare epocă înmulţind-o cu factorul 0.85. Când numărul de imagini eronat clasificate din setul de antrenare s-a stabilizat, am dezactivat distorsiunile setului de antrenare şi am mai antrenat reţeaua încă 10 epoci cu η=0.000005. Acest pas final a scăzut mult eroarea pe setul de antrenare deoarece acesta nu a mai fost distorsionat, dar a scăzut-o puţin şi pe setul de test deoarece a atenuat efectul distorsiunilor care erau prea mari. Alegând o valoare atât de mică pentru η, nu a existat pericolul ca reţeaua să-şi piardă capacitatea de generalizare.

Tabelul 5.2: Rezultate obţinute cu reţele neuronale cu hărţi de convoluţie.

L_1 (hărţi)	L_2 (hărţi)	L_3 (neuroni)	eroare antrenare LNIST (%)	eroare test LNIST (%)	eroare test MNIST (%)
4	20	40	1.37	1.21	1.04
4	40	80	1.18	1.11	0.89
6	50	60	0.63	0.88	0.72
6	50	100	0.80	**0.86**	0.70
6	60	80	0.54	0.99	0.77
8	60	100	0.92	0.89	0.80
8	60	120	0.96	0.98	0.74

Timpul maxim pentru o epocă de antrenare a fost de 620 de secunde, iar cel minim de 210 secunde. Aceşti timpi au fost obţinuţi pe procesoare Core2 Duo la 2.4Ghz. Folosind procesoare Core2 Quad similare ca frecvenţă, dar cu 4 nuclee, timpii de antrenare s-au redus la 60-70%. Reţeaua neuronală ce a oferit cea mai bună rată de recunoaştere va fi folosită drept clasificator la nivel de cifră în sistemul de clasificare general (Figura 5.4).

BUPT

5.2 Experimente cu şiruri compuse din două cifre parţial suprapuse

În această secţiune vor fi prezentate rezultatele antrenării unui clasificator pentru şiruri de două cifre parţial suprapuse. Acest clasificator va fi utilizat apoi în sistemul de recunoaştere (a se vedea Figura 5.4) care recunoaşte şiruri de una, două sau mai multe cifre distincte sau parţial suprapuse.

Toate experimentele au fost efectuate pe imaginile din TDNS micşorate de la dimensiunea originală de 40×28 pixeli la 21×13 pixeli. Utilizarea imaginilor de aceste dimensiuni pentru imaginile de intrare a permis antrenarea reţelelor într-un timp rezonabil, de maxim 24 de ore pe un calculator cu procesor cu două nuclee rulând la 2.4GHz. Imaginile sunt mici, iar multe dintre ele sunt dificil de recunoscut chiar de om. Câteva exemple de imagini provenite de la două persoane diferite sunt prezentate în Figura 5.1. Dimensiunile exacte ale imaginilor din seturile de antrenare şi de test au fost determinate de constrângerile impuse la proiectarea reţelei de convoluţie. Redimensionarea de la 40×28 pixeli la 21×13 pixeli a fost efectuată cu păstrarea aspectului imaginilor. Timpul necesar antrenării unei reţele a variat între o oră pentru cea mai simplă reţea (număr redus de ponderi) şi câteva zile pentru reţelele complexe care aveau un număr mare de ponderi. Am folosit 15 calculatoare cu procesoare cu două nuclee pentru mai mult de o săptămână pentru a antrena reţelele. Au existat câteva cazuri de reţele pentru care antrenarea a trebuit reluată deoarece s-a împotmolit într-un minim local.

Figura 5.1: Exemple de imagini din fişierul de antrenare.

5.2.1 Reţele neuronale de tip perceptron

Acest tip de reţea neuronală foarte generală a fost ales pentru a stabili o bază de comparaţie pentru arhitecturile specializate prezentate în secţiunile următoare. Tipul de perceptron ales este unul cu trei straturi, din care unul este ascuns, celelalte două reprezentând straturile de intrare, respectiv de ieşire. Am variat doar numărul de neuroni din stratul intermediar, parametrii celorlaltor straturi fiind impuşi de dimensiunile imaginilor, respectiv de numărul de clase. Rezultatele din Tabelul 5.3 arată că eroarea descreşte cu creşterea numărului de neuroni de pe stratul ascuns. Erorile obţinute pe seturile de test şi de antrenare sunt foarte asemănătoare, diferenţa dintre ele fiind de maxim 0.37%. Având în vedere că seturile de antrenare şi de test conţin imagini de la persoane distincte, această diferenţa mică între erorile

BUPT

pe setul de test şi pe cel de antrenare demonstrează capacitatea deosebită de generalizare şi de adaptare la nou a reţelei.

Timpul de antrenare necesar fiecărei epoci de antrenare creşte cu numărul de neuroni din stratul ascuns, de la 200 de secunde pentru 60 de neuroni până la 710 secunde pentru 220 de neuroni. Eroarea a scăzut lent, astfel că am decis oprirea experimentelor la 220 de neuroni pe stratul ascuns deoarece timpul necesar antrenării creştea prea rapid.

Tabelul 5.3: Rezultate obţinute cu perceptronul cu un strat ascuns.

# neuroni pe stratul ascuns	eroarea pe setul de antrenare (%)	eroarea pe setul de test (%)	timpul necesar unei epoci de antrenare (s)
60	26.85	26.74	200
80	24.92	24.65	264
100	23.45	23.47	327
120	22.28	22.21	391
140	22.14	22.29	454
160	21.33	21.54	518
180	21.17	21.35	581
200	20.94	21.31	645
220	20.55	20.74	710

5.2.2 Reţele neuronale cu extractori de trăsături

În [38] Yann LeCun propune o arhitectură sofisticată de reţea neuronală numită LeNet-5. Reţeaua s-a dovedit a fi foarte bună pentru recunoaşterea cifrelor singulare, chiar şi în prezenţa zgomotului. Şirurile de cifre distincte pot fi recunoscute facil folosind un astfel de clasificator aplicat pe o fereastră ce se deplasează în lungul imaginii. Patrice Simard ([64]) a folosit reţele neuronale de convoluţie pentru recunoaşterea cifrelor singulare. Clasificatorul utilizat de mine (Figura 5.2) este derivat din cel al lui Simard, dar este modificat extensiv deoarece am dorit sa poată fi folosit pentru recunoaşterea perechilor de cifre parţial suprapuse.

In Figura 5.2 stratul de intrare este alcătuit din 21x13 neuroni, câte unul pentru fiecare pixel al imaginii. Nivelul de gri ($0 \div 255$) al fiecărui pixel din imaginea de intrare este convertit proporţional la un număr din intervalul $[-1, 1]$. Această valoare este transferată neuronului corespunzător de pe stratul de intrare. La fel ca în [64] am conectat stratul de intrare la un strat de convoluţie. Acest strat de convoluţie este alcătuit din mai multe hărţi de convoluţie (HC). În experimente am variat acest număr între 2 şi 12. Dimensiunea nucleului de convoluţie a fost păstrată la 5x5 pixeli. Acest nucleu de convoluţie a fost aplicat tot din doi în doi pixeli, mai exact, numai pentru pixelii care aveau ambele coordonate pare. Aplicarea nucleului de convoluţie în acest mod a generat o reducere a dimensiunilor hărţilor de convoluţie de $(n-3)/2$, unde n reprezintă înălţimea, respectiv lăţimea stratului L_0. În concluzie, dimensiunea hărţilor de convoluţie din stratul L_1 este de 9x5 neuroni. Fiecare din cei 45 de neuroni ai stratului L_1 este conectat cu 25 de neuroni din nucleul de convoluţie corespunzător. Pe lângă aceste 25 de conexiuni mai există una pentru bias. Deşi există $9 \times 5 \times (5 \times 5 + 1) = 1170$ conexiuni pentru fiecare hartă, numărul de ponderi ce trebuie antrenate este mult redus,

BUPT

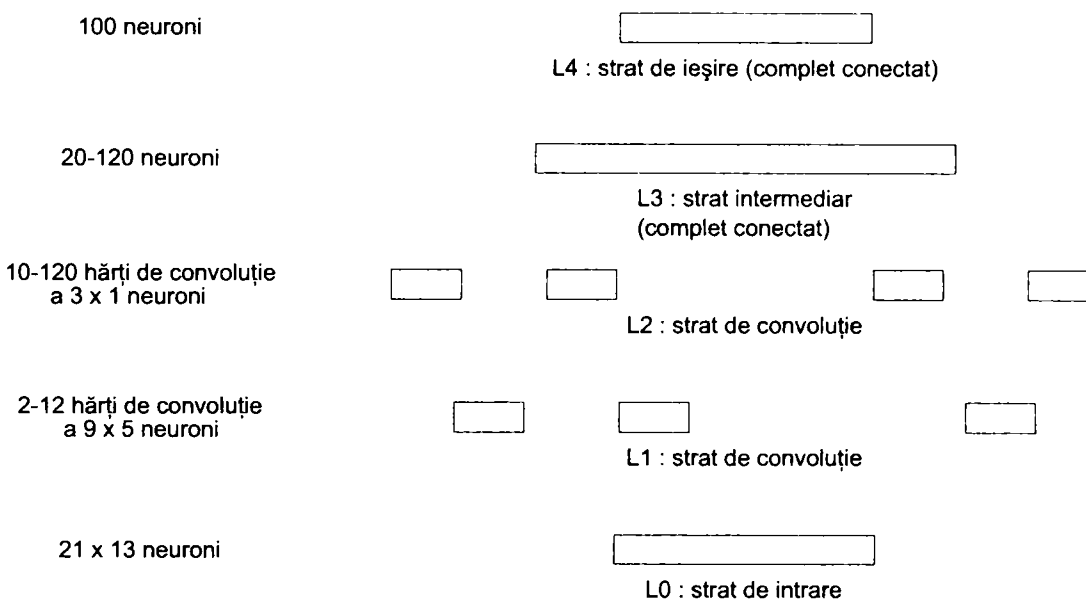


Figura 5.2: Arhitectura reţelei neuronale cu hărţi de convoluţie.

$5 \times 5 + 1 = 26$, acestea fiind partajate de toţi neuronii din stratul L_1. Deoarece antrenarea unei reţele neuronale constă în obţinerea unor valori apropiate de optim pentru ponderile reţelei, RNC pot fi antrenate mult mai rapid decât reţelele de tip perceptron. Conexiunile şi ponderile sunt diferite pentru fiecare hartă de convoluţie. În total sunt 11700 conexiuni şi 260 de ponderi pentru L_1, în cazul unei reţele cu 10 hărţi pe stratul respectiv.

Pentru stratul L_2 am experimentat cu un număr de 10 pană la 120 de hărţi de convoluţie. Având în vedere că s-a folosit aceeaşi regulă de aplicare a nucleului de convoluţie - doar pentru poziţiile pare - şi că dimensiunea nucleului a fost menţinută la 5×5, dimensiunea hărţilor de pe L_2 este de 3×1 neuroni. Fiecare neuron din acest strat este conectat cu 5×5 neuroni din fiecare hartă a stratului anterior (Figura 3.5) plus o conexiune pentru bias. Deci, o hartă de convoluţie din stratul L_2 conţine $3 \times 1 = 3$ neuroni, $3 \times 1 \times (5 \times 5 \times 10 + 1) = 753$ conexiuni spre neuroni din L_1 (am considerat că L_1 are 10 hărţi de convoluţie), şi $5 \times 5 \times 10 + 1 = 251$ ponderi. În total, pentru 100 de hărţi de convoluţie pe stratul L_2, sunt $100 \times 3 = 300$ de neuroni, $753 \times 100 = 75300$ conexiuni şi $251 \times 100 = 25100$ ponderi.

După două nivele de convoluţie reducerea este de $[(n-3)/2 - 3]/2 = p \Rightarrow n = 4p + 9$, unde $p \in 1, 2, 3, 4, 5, \ldots$. În formula anterioară n reprezintă dimensiunea iniţială a imaginii, iar p - dimensiunea unei hărţi de convoluţie după două nivele de convoluţie. În concluzie, n ia valori din mulţimea $13, 17, 21, 25, 29, \ldots$. Pentru a menţine dimensiunea imaginilor de intrare la un nivel cât mai mic, dimesiunea stratului de intrare este aleasă ca fiind 21×13 neuroni. S-a ţinut cont şi de aspectul imaginilor cu două cifre concatenate (setul TDNS), acesta fiind $40/28 \approx 1.43$. Alegând 13 pentru înălţimea stratului de intrare, lăţimea trebuie sa fie cel mai mic număr întreg mai mare decât $40/28 \times 13 = 18.57$. Valoarea din lista anterioară care corespunde acestei cerinţe este 21. Stratul de intrare s-a dorit a fi cât mai mic pentru a obţine un timp de antrenare rezonabil pentru reţeaua neuronală, nu mai mare de câteva zile.

Stratul L_3 este total conectat, iar dimensiunea lui variază de la 40 la 120 de neuroni. Ultimul strat, L_4, are 100 de neuroni, câte unul pentru fiecare clasă, şi este, de asemenea,

BUPT

total conectat cu stratul anterior. Ieşirea fiecărui neuron ia valori în intervalul $[-1, 1]$. Pentru o imagine validă, neuronul care reprezintă clasa imaginii va avea o valoare apropiată de 1, iar ceilalţi 99 de neuroni vor avea valori apropiate de -1. Dacă imaginea este ambiguă şi seamănă cu mai mult de o clasă, ieşirile neuronilor corespunzători claselor cu care seamănă imaginea vor avea valori corelate cu gradul de asemănare.

Tabelul 5.4: Rezultate obţinute cu reţele neuronale cu hărţi de convoluţie.

L_1 (hărţi)	L_2 (hărţi)	L_3 (neuroni)	fără deformări		cu deformări		
			Eroare antrenare (%)	Eroare test (%)	Eroare antrenare (cu def.) (%)	Eroare antrenare (%)	Eroare test (%)
2	10	40	35.00	35.43	44.48	39.79	39.88
		60	34.76	34.61	39.24	34.25	34.21
		80	26.25	26.03	37.71	33.42	32.83
	30	40	21.64	21.51	35.82	31.69	31.21
		60	19.98	19.96	32.19	29.95	29.20
		80	18.26	18.15	30.54	25.77	25.31
	50	40	17.80	18.04	28.95	23.62	22.91
		60	15.53	15.68	28.65	22.96	22.47
		80	15.30	15.30	24.67	18.65	18.16
4	10	40	25.63	25.32	30.86	25.76	25.56
		60	22.27	22.08	30.16	27.00	26.60
		80	18.25	18.44	26.08	21.74	21.20
	30	40	15.19	15.25	26.66	23.20	23.06
		60	12.67	13.26	18.88	15.39	15.23
		80	11.80	12.02	18.07	14.29	14.03
	50	40	11.30	12.34	18.78	16.55	16.57
		60	10.67	11.50	15.42	13.04	12.94
		80	10.33	10.64	15.20	11.86	11.72
6	10	40	15.57	15.63	26.35	24.15	23.32
		60	15.50	15.94	23.30	20.40	19.62
		80	15.45	15.67	21.31	17.49	17.34
	30	40	10.72	11.40	25.20	23.07	23.40
		60	10.15	10.81	16.74	13.44	13.57
		80	9.44	10.33	13.95	11.32	11.46
	50	40	9.15	9.83	13.94	11.50	11.82
		60	7.52	8.48	13.76	10.79	11.15
		80	6.73	7.83	12.10	9.63	9.91
8	10	40	15.36	15.78	20.08	17.90	17.86
		60	14.21	14.78	19.91	16.66	16.78
		80	12.97	13.49	17.08	14.38	14.42
	30	40	9.49	10.26	15.78	12.95	13.25
		60	8.20	9.10	13.62	10.01	10.42
		80	7.75	8.72	12.06	9.33	9.74
	50	40	8.06	9.22	15.25	13.16	13.33
		60	7.22	8.31	11.23	8.82	9.51
		80	6.52	7.86	10.70	8.22	8.71
10	60	100	5.63	6.85	10.71	7.89	8.44
	100	100	3.27	5.35	10.15	7.10	7.60
12	120	120	5.23	6.54	-	-	-

Pentru 2, 4, 6 şi 8 hărţi de convoluţie pe stratul L_1; 10, 30 şi 50 de hărţi pe stratul L_2, respectiv 40, 60 şi 80 de neuroni pe stratul L_3 am efectuat toate combinaţiile posibile, construind $4 \times 3 \times 3 = 36$ de reţele neuronale. Fiecare reţea a fost antrenată în două moduri: cu, respectiv fără activarea deformărilor asupra imaginilor din setul de antrenare. Operaţiile efectuate asupra imaginilor sunt similare cu cele din [64], dar am ales valori mai mici pentru

BUPT

rotaţii, translaţii şi deformări elastice. Motivul descreşterii valorilor acestor parametri este dimensiunea mult mai mică a imaginilor pe care le-am folosit, 21×13, faţă de 29×29 cât a folosit Simard pentru cifre singulare. Explicit, factorul de scalare a fost limitat la maxim 6%, rotaţiile la maxim $\pm 4°$, σ pentru distorsiuni elastice a fost ales 10.0, iar scalarea pentru distorsiuni elastice - 0.2.

Antrenarea celor 36 de reţele cu şi fără imagini distorsionate a generat 72 de seturi distincte de rezultate (Tabelul 5.4). Pentru reţelele antrenate cu imagini distorsionate există două tipuri de erori pentru setul de antrenare. Prima este eroarea obţinută testând reţeaua cu setul de antrenare asupra căruia s-a aplicat o transformare aleatoare (rotaţie + translaţie + deformare) la începutul fiecărei epoci. Al doilea tip de eroare este obţinut pe setul de antrenare original.

Rezultatele din Tabelul 5.4 arată că eroarea este monoton descrescătoare (cu extrem de mici excepţii ce au drept cauză probabilă o subantrenare a reţelei respective sau atingerea unui minim local apropiat de minimul global). Pentru oricare două reţele antrenate similar - ambele cu set de antrenare cu imagini deformate sau ambele cu set de antrenare cu imagini nedeformate - oricare din cele cinci tipuri de erori a variat corespunzător cu modificarea unuia din cei trei parametri ai reţelei. Explicit, dacă doi dintre parametri sunt menţinuţi constanţi (de exemplu, numărul de hărţi din L_1 şi numărul de neuroni din L_3), iar cel de-al treilea (în acest caz, numărul de hărţi de pe stratul L_2) este mărit, erorile descresc şi reciproc. Concluzia evidentă este că erorile pot fi diminuate dacă mărim dimensiunea reţelei. Pentru a verifica această afirmaţie, am antrenat trei reţele şi mai mari (ultimele trei rânduri din Tabelul 5.4). Am obţinut cele mai mici erori (5.35% pentru setul de test) cu reţeaua cu 10 hărţi pe L_1, 100 de hărţi pe L_2 şi 100 de neuroni pe L_3. Eroarea pe setul de test a fost cu 1.60% mai mică decât precedentul minim obţinut, 6.85% (10 hărţi pe L_1, 60 de hărţi pe L_2 şi 100 de neuroni pe L_3). Dezavantajul a fost o creştere accentuată a timpului necesar antrenării, de la 24 de ore la peste 48 de ore.

În concluzie, cele mai bune rezultate, exceptând cazurile speciale cu 10 sau 12 hărţi de convoluţie pe stratul L_1, au fost obţinute pentru opt hărţi pe stratul L_1. Suprafaţele erorilor sunt arătate în Figura 5.3. Cea mai mare reţea antrenată (ultimul rând din Tabelul 5.4) a conţinut 12 hărţi pe L_1, 120 de hărţi pe L_2 şi 120 de neuroni pe L_3. Chiar şi după 96 de ore de antrenare eroarea nu a scăzut sub minimul antins de reţeaua cu 10 hărţi pe L_1, 100 de hărţi pe L_2 şi 100 de neuroni pe L_3. Am concluzionat că am detectat un minim suficient de apropiat de minimul global şi mărirea în continuare a parametrilor reţelei ar fi inutilă şi foarte costisitoare din punct de vedere al timpului de antrenare.

Referitor la timpul necesar unei epoci de antrenare, acesta a variat de la 70 de secunde pentru cea mai simplă reţea până la 670 de secunde pentru cea mai complexă, cea cu 10 hărţi pe L_1, 100 de hărţi pe L_2 şi 100 de neuroni pe L_3. Antrenarea a fost mult mai rapidă decât în cazul perceptronului, iar rata de recunoaştere este semnificativ mai bună.

Chiar dacă am scăzut amplitudinea transformărilor aplicate pe imaginile din setul de antrenare, toate reţelele antrenate cu imagini deformate au atins o eroare minimă mai mare, atât pe setul de antrenare, cât şi pe setul de test. Adăugarea deformărilor a degradat performanţa reţelelor cu cel puţin 0.85% (cazul L_1: 8 hărţi, L_2: 50 de hărţi, L_3: 80 de neuroni). Se poate concluziona că acest tip de aplicare a deformărilor este eficient doar pentru clasificarea cifrelor singulare, nu şi pentru perechi de cifre parţial suprapuse.

BUPT

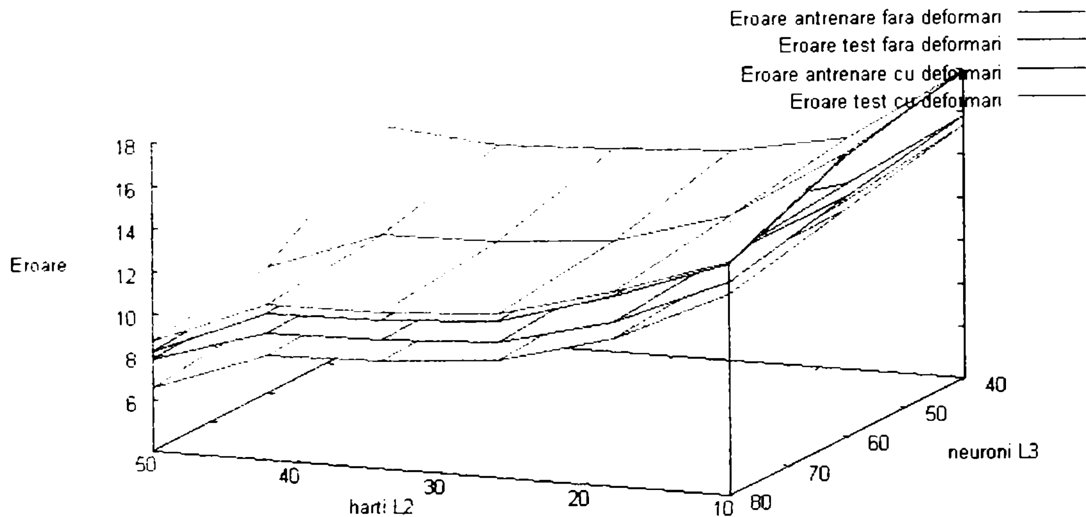


Figura 5.3: Suprafeţele erorilor reţelelor cu 8 hărţi de convoluţie pe stratul L_1.

Analiza rezultatelor celei mai bune reţele

Experimentele efectuate au arătat că cea mai bună reţea are zece hărţi de convoluţie a 9×5 neuroni pe stratul L_1, o sută de hărţi de convoluţie a 3×1 neuroni pe stratul L_2 şi o sută de neuroni pe stratul L_3. Din cele 200000 de imagini din setul de test, 10708 au fost clasificate greşit. Aceasta reprezintă o eroare de 5.35% pe setul de test.

Din cele 10708 erori doar în 663 (0.33%) de cazuri ambele cifre au fost clasificate defectuos. Această rată de eroare atât de redusă oferă premisele dezvoltării unui sistem de recunoaştere care va utiliza acest clasificator pentru recunoaţerea şirurilor de cifre. Cea mai semnificativă cifră - cifra zecilor - a fost clasificată eronat în 5239 de cazuri, iar cea mai puţin semnificativă cifră - cifra unităţilor - în 6123 de cazuri. Putem concluziona că este mai dificil pentru reţeaua neuronală să identifice cifre suprapuse în partea partea stângă decât în partea dreaptă.

Distribuţia erorilor pe clase arată că clasele 67, 64 şi 69 au fost mai dificil de recunoscut, generând 253 (2.36%), 219 (2.04%), respectiv 212 (1.98%) erori de clasificare. În paranteze este trecută procentual contribuţia clasei la eroarea totală.

Rata de eroare de 5.35% înseamnă 94.65% rată de recunoaştere. Aceasta reprezintă o rată de recunoaţere forţată, adică fară rejecţie. Pentru a îmbunătăţii calitatea detecţiei am calculat ratele de recunoaştere considerând rate de rejecţie variabile. O imagine este rejectată dacă $N(C1) - N(C2) < T$, unde $N(C1)$ şi $N(C2)$ reprezintă primele două maxime ale neuronilor de ieşire pentru acea imagine, normalizate la intervalul $[-1, 1]$, iar T reprezintă pragul de rejecţie. Atât autorii din [44], cât şi din alte publicaţii folosesc pentru valoarea de prag valorile 0.5% şi 1%, astfel că am calculat ratele de eroare pentru aceste rate de rejecţie, dar am adăugat şi pentru două valori mult mai mari: 2%, respectiv 5%. Diferenţele foarte mici dintre ratele de recunoaştere din Tabelul 5.5 arată că reţeaua are un grad mare de siguranţă,

BUPT

rata de recunoaştere cu rejecţia maximă de 5% fiind cu numai 0.59% mai mică decât rata de recunoaştere cu fără rejecţie.

Tabelul 5.5: Influenţa strategiei de rejecţie asupra ratei de recunoaştere (forţat = fără rejecţie).

T (%)	forţat	0.5	1	2	5
rata de recunoaştere (%)	94.65	94.59	94.53	94.42	94.06

5.2.3 Concluzii

RNC sunt mult mai performante decât perceptronul cu un strat ascuns, acesta din urmă generând o rată de eroare prea mare pentru a putea fi folosit în clasificatori mai complecşi. RNC reprezintă o soluţie bună atât pentru recunoaşterea de cifre parţial suprapuse, cât şi pentru recunoaşterea de cifre izolate. Rata de eroare minimă obţinută pentru setul de test din TDNS este de 5.35% pentru o reţea cu 10 hărţi pe L1, 100 de hărţi pe L2 şi 100 de neuroni pe L3.

Deformările elastice nu îmbunătăţesc performanţele în cazul cifrelor parţial suprapuse, aşa cum făceau pentru cifre singulare. Cum voi arăta în secţiunile care urmează, există motive întemeiate pentru care deformările nu s-au dovedit utile.

Rata de eroare de 5.35% este suficient de mică pentru a folosi acest clasificator într-unul general pentru recunoaşterea de şiruri numerice formate dintr-un număr indefinit de cifre.

Diferenţele extrem de reduse dintre ratele de recunoaştere cu şi fără rejecţie demonstrează gradul de robusteţe al clasificatorului.

Rezultatele au fost publicate în [20] şi au fost prezentate la ICCP 2008.

5.3 Experimente cu şiruri de cifre din NIST SD 19

După ce am obţinut doi clasificatori cu rate bune de recunoaştere - unul pentru cifre izolate şi unul pentru perechi de cifre parţial suprapuse - am trecut la construirea unui clasificator care să poată recunoaşte numere formate dintr-un număr oarecare de cifre. Acest clasificator a fost apoi testat cu şiruri numerice din NIST SD 19 şi comparat cu cele mai bune rezultate existente în domeniu.

5.3.1 Sistemul de recunoaştere

Diagrama sistemului de recunoaştere este prezentată în 5.4. Fiecare şir de cifre este întâi tăiat la dimesiunile dreptungiului delimitator şi apoi este prezentat Analizorului de Componente Conexe (CCA) care extrage toate componentele conexe (CC) din imagine. În continuare, componentele conexe vor fi clusterizate în imagini ce probabil vor conţine cifre sau mulţimi de cifre. Procesul de clasificare este efectuat de Clasificator folosind două Reţele Neuronale de Convoluţie (RNC).

5.3.2 Analizorul de componente conexe

Toate componentele conexe din imaginea de intrare sunt extrase folosind procedeul recursiv de căutare în doar patru direcţii (mai corect: două direcţii cu două sensuri fiecare), orizontale

BUPT

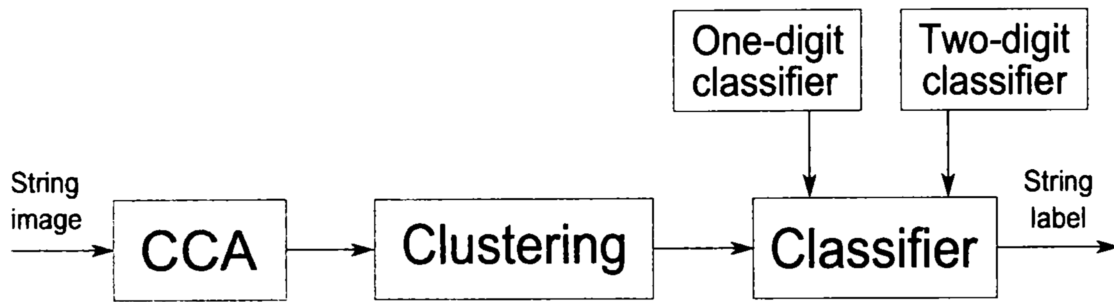


Figura 5.4: Arhitectura sistemului de recunoaştere a şirurilor de cifre.

şi verticale. Nu am folosit căutarea în opt direcţii deoarece neefectuând segmentare acesta este unicul loc unde imaginea de intrare poate fi separată în cât mai multe componente printr-un simplu proces de extragere. Orice separare a imaginii unei cifre sau a mai multor cifre va fi corectată în faza următoare, cea de clusterizare.

Pentru fiecare componentă se calculează diverşi parametri: arie, dreptunghiul circumscris de arie minimă (dreptunghiul delimitator), lăţime, înălţime, aspect (lăţime / înălţime), distanţe la alte componente. CCA atribuie câte o culoare distinctă fiecărei componente. În contrast cu alte metode ([44, 30, 31]), prezenta metodă poate evita procesul de segmentare prin utilizarea suplimentară a unui clasificator pentru perechi de cifre parţial suprapuse, pe lângă clasicul clasificator la nivel de cifră.

5.3.3 Componenta de clusterizare (agregare)

În cazul imaginilor scanate există multe componente conexe care formează o singură cifră sau mai multe cifre alipite. Pentru a putea reconstrui imaginea lor, etapa de agregare efectuează secvenţial următoarele operaţii (constantele din toate testele referitoare la suprafaţa unei componente - numărul de pixeli - au fost stabilite pentru imagini din setul NIST SD 19. O caracteristică a acestor imagini e aceea că au fost scanate la 300dpi. Constantele pot fi folosite şi în cazul altor imagini scanate la alte rezoluţii, dar trebuie multiplicate cu pătratul (deoarece suprafaţa e o arie) factorului de redimensionare de la o rezoluţie la cealaltă):

- toate componentele mici conţinând maxim 4 pixeli sunt şterse dacă cea mai apropiată componentă este la o distanţă mai mare de 5 pixeli. Motiv: sunt prea mici şi prea depărtate de alte componente, deci probabil că reprezintă doar zgomot.
- oricare două componente separate prin doar un pixel sunt concatenate dacă respectă cel puţin una din condiţiile:
  - ambele au înalţimi mai mari de 60% din înălţimea imaginii, iar distanţa dintre ele sau distanţa de suprapunere este mai mică de 35% din lăţimea componentei celei mai late.
  - ambele au arii mai mari de 200 de pixeli şi aspecte mai mari de 0.5, şi cel puţin una dintre ele are aspectul mai mare decât 1şi distanţa dintre ele sau distanţa de suprapunere parţială este mai mică decât 25% din lăţimea componentei mai late.
  - ambele au arie mai mare de 400 de pixeli

BUPT

După acest pas se reia algoritmul pentru componentele separate prin maxim doi pixeli, şi, la final, pentru maxim trei pixeli. Aceste operaţii agreghează cu succes componentele din imaginile scanate defectuos sau care conţin cifre scrise intermitent (Figura 5.5).

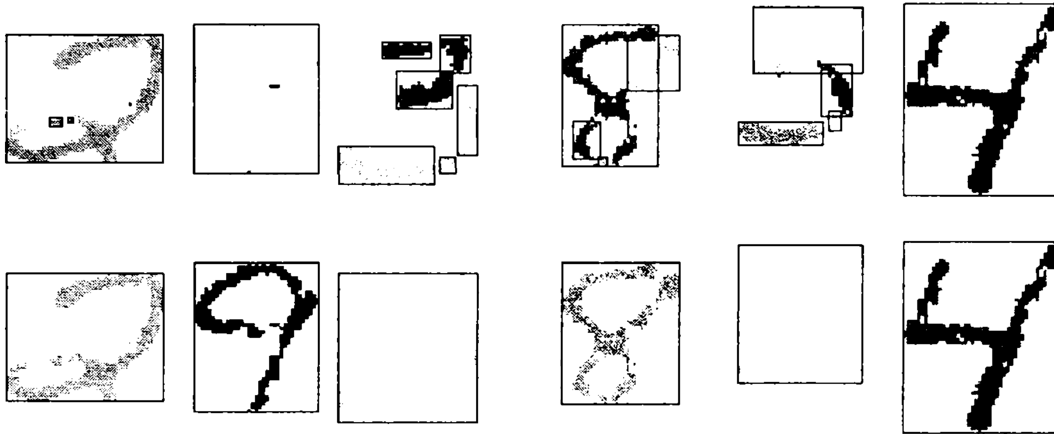


Figura 5.5: Componentele conexe înainte (sus) şi după (jos) faza de agregare. Fiecare componentă este colorată distinct şi este desenat şi dreptunghiul delimitator.

- există multe persoane care scriu cifra 5 din două părţi, una pentru corpul cifrei şi cealalată pentru partea superioară. În mod frecvent cele două părţi sunt foarte distanţate pe orizontală şi etapele anterioare de agregare nu sunt suficiente pentru a reconecta părţile cifrei 5. Acest caz este distinct şi apare doar în cazul cifrei 5. Pentru a reforma cifra, se caută întâi partea superioară a lui 5 ca fiind o componentă cu următoarele proprietăţi: are cel puţin 10 pixeli, are un aspect alungit pe orizontală (mai mare de 1.4), nu este prea groasă (înălţime mai mică decât 40% din înălţimea imaginii), şi este plasată în jumătatea superioară a imaginii. Dacă o componentă de acest tip este detectată, atunci se caută o componentă care să poată reprezenta partea inferioară a cifrei 5. O astfel de componentă e obligatoriu să aibă minim 100 de pixeli.

Figura 5.6: Reconectarea cifrei 5.

Partea superioară a cifrei trebuie să înceapă în partea dreaptă a corpului ei, dar nu mai departe de 50% din lăţimea corpului cifrei după aceasta. Dacă nicio componentă nu este detectată, atunci componenta care se presupunea că reprezintă partea superioară a lui 5 rămâne neconcatenată. Dacă ambele părţi ale cifrei 5 sunt detectate, atunci

BUPT

partea superioară este translatată spre stânga până ajunge pe aceeaşi verticală cu pixelul din dreapta-sus al componentei ce reprezintă corpul cifrei 5 (Figura 5.6).

- acest pas tratează puţinele cazuri de cifre scrise din mai multe părţi separate vertical. Dacă două componente sunt suprapuse orizontal pe minim 80% din lăţimea cel puţin a uneia dintre ele, atunci sunt concatenate.

5.3.4 Clasificatorul

Componenta de clasificare (Figura 5.4) foloseşte două RNC, una pentru cifre şi una pentru perechi de cifre parţial suprapuse. Ambele RNC au fost descrise anterior, iar ratele de eroare obţinute au fost 0.86% pentru o cifră, respectiv 5.35% pentru două cifre.

Testarea sistemelor de recunoaştere a fost efectuată pe seturile de test 3DNS, 6DNS şi 10DNS (detalii sunt prezentate în Secţiunile 4.5.2, 4.5.3 şi 4.6), seturi care sunt folosite şi de alţi autori [44] pentru compararea sistemelor de recunoaştere. Autorii din [44] şi-au comparat metodele cu cele mai bune din literatură, prezentând cele mai eficiente metode existente până în anul 2004.

Folosind cele două RNC am implementat mai multe sisteme de recunoaştere. Ambele RNC preiau rând pe rând câte o componentă din imaginea de la intrare şi încearcă să determine cărei clase i se potriveşte respectiva componentă (să recunoască cifra sau perechea de cifre din imaginea componentei). Înainte de a intra în etapa de clasificare, toate componentele care sunt ori mai mici de 80 de pixeli ori au înălţimea mai mică de 30% din înălţimea imaginii sunt eliminate.

În toţi algoritmii ce urmează, rez_i reprezintă un vector de structuri, fiecare structură conţinând două câmpuri: $class$ şi $score$. Indicele i este 1 pentru 1RNC şi 2 pentru 2RNC. Vectorii rez_i stochează ieşirea RNC. De expemplu, $rez_2[3].class$ reprezintă clasa celui de-al patrulea element al rezultatului obţinut de 2RNC. Câmpul $class$ nu este redundant deoarece este util după ce vectorul este sortat, pentru a ştii ce clasă are cel mai bun scor, ce clasă are al doilea scor etc.

Sistem de recunoaştere bazat pe scorul maxim

Cel mai simplu sistem de recunoaştere care poate fi implementat ia o imagine şi o prezintă la RNC_1 şi la RNC_2. Cele mai mari scoruri al celor două RNC sunt comparate, iar clasificatorul care a obţinut maximul va oferi atât clasa, cât şi numărul de cifre prezente în imaginea de la intrare (Algoritm 4).

Din cele 1476 imagini (setul 3DNS) care conţineau 3 cifre, 158 au fost incorect clasificate. Aceasta înseamnă o rată de recunoaştere de 89.29%. Rezultatul pentru perceptronul multistrat din [44] antrenat fără exemple negative este 82.52%. Având în vedere că metoda prezentă nu foloseşte segmentare deloc, recunoaşterea corectă a 89.29% din imagini este promiţătoare, dar mai există loc pentru îmbunătăţiri. Este foarte important a se observa că rezultatul obţinut reprezintă dovada eficienţei combinării a doi clasificatori, unul pentru recunoaşterea cifrelor şi unul pentru recunoaşterea perechilor de cifre parţial suprapuse, faţă de folosirea doar a unui clasificator la nivel de cifră [44], chiar dacă ultima este ajutată de un complicat mecanism de segmentare [44].

După studierea erorilor sistemului de recunoaştere, am ajuns la următoarele concluzii:

BUPT

foreach *componentă din imagine* **do**
$rez_1[\,] = RNC_1$(componentă);
$rez_2[\,] = RNC_2$(componentă);
sortează descrescător rez_1 şi rez_2;
if $rez_1[0].score \geq rez_2[0].score$ **then**
return $rez_1[0].class$;
end
else
return $rez_2[0].class$;
end
end

Algorithm 4: Sistem de recunoaştere bazat pe scorul maxim.

- deoarece am antrenat ambele RNC fără exemple negative, ele nu pot fi utilizate direct pentru a detecta dacă o componentă conţine una, două sau mai multe cifre. Prin antrenarea unei RN doar cu exemple pozitive, reţeaua va încerca să clasifice orice imagine de intrare, chiar şi pe cele incorecte - care nu conţin reprezentări grafice ale niciuneia dintre clasele învăţate -, clasei cu care seamănă cel mai mult. Mai mult, probabil din cauza naturii convoluţionale a reţelelor, cele mai confuze cazuri apar la numerele formate din două cifre de tipul $\overline{XX}$. Este foarte dificil atât pentru RNC_1, cât şi pentru RNC_2 să distingă dacă o imagine conţine o singură cifră sau un număr format din două cifre. De exemplu, dacă imaginea conţine doar cifra 4, scorul ambelor RNC poate fi foarte mare şi foarte asemănător, RNC_2 detectând numărul 44!
- multe imagini conţinând cifrele 0, 1, 3 , 4, 7, 8, respectiv 9 sunt interpretate de RNC_2 ca fiind 01, 11, 31, 41, 71, 81, respectiv 91. Problema se datorează metodei de unire ([20]) a cifrelor pentru setul de antrenare al RNC_2, deoarece dacă cifra 1 este scrisă ca un segment de dreaptă şi este suprapusă parţial cu o cifră care are în partea dreaptă pixeli pe toată înălţimea cifrei, rezultatul este foarte confuz (Figura 5.7). Un argument similar se poate oferi şi pentru cazurile 10, 16, 18. Aceste cazuri sunt notate cu $\overline{1X}$ şi $\overline{X1}$.

Figura 5.7: Cazurile de unire defectuoasă de tipul $\overline{X1}$.

Evident, antrenarea ambelor RNC cu exemple negative ar fi rezolvat multe probleme, dar colectarea manuală a imaginilor cu exemple negative - imagini care nu conţin cifre sau perechi de cifre - din formularele NIST SD 19 este un proces foarte îndelungat şi neplăcut. Numărul de exemple negative trebuie să fie suficient de mare pentru a asigura o antrenare adecvată. În [44], autorii au antrenat clasificatorul la nivel de cifră cu 1000 şi apoi cu 4000 de exemple negative. În ambele cazuri rata de recunoaştere a clasificatorului la nivel de cifră s-a redus cu 2-4%, dar acesta a devenit rezistent la imagini ce nu conţin cifre. Metoda prezentată în această teză are doi clasificatori, deci ar trebui colectate exemple negative pentru ambii. Bineînţeles, scrierea unui algoritm de generare automată a exemplelor negative este un alt

BUPT

mod de a rezolva această problemă, dar ar trebui conceput un astfel de algoritm. Autorii din [44] au ajuns la concluzia că tot mai bune sunt datele colectate manual, astfel că scrierea algoritmului de generare ar putea reprezenta timp irosit. După ce am evaluat toate aceste aspecte, am decis să nu antrenez clasificatorii cu exemple negative şi să încerc să dezvolt o metodă care să atenueze problemele prezentate anterior într-o manieră cât mai simplă.

Când am studiat erorile, am observat că în majoritatea cazurilor RNC recunoşteau corect, dar scorul relativ al clasificatorilor genera clasificarea incorectă.

Am antrenat ambele RNC astfel încât să fie cât mai puţin influenţate de translaţii, dar dacă imaginea este perfect centrată în clasificator, atunci scorurile sunt mult mai bune. Considerând acest aspect, am încercat să translatez imaginea în câmpul de intrare al clasificatorilor. RNC_1 a fost antrenată cu baza de date LNIST, iar imaginile din ea sunt centrate în centrul de masă. Din această cauză am aplicat repetat RNC_1 asupra imaginii translatate cu ± 1 pixeli relativ la centrul de masă şi am păstrat cel mai mare scor din cele $3 \times 3 = 9$ aplicări. RNC_2 a fost antrenată cu imagini care au fost centrate prin centrarea dreptunghiului delimitator. Considerând această metodă de generare ([20]) a imaginilor de antrenare, am plasat imaginea de 18×10 pixeli în toate poziţiile posibile în câmpul de intrare de 21×13 neuroni ai RNC_2. Există $(21 - 18 + 1) \times (13 - 10 + 1) = 16$ posibilităţi. Din nou am reţinut scorul maxim pentru fiecare clasă. Din păcate, numărul de clasificări incorecte a crescut (Tabelul 5.6) şi timpul de execuţie s-a mărit din cauza aplicărilor repetate ale clasificatorilor. Se poate conclude că mutarea clasificatorilor deasupra imaginii (sau, echivalent, a imaginii în câmpul clasificatorilor) nu îmbunătăţeşte rata de recunoaştere a metodei ce foloseşte clasificarea bazată pe scor maxim.

Sistem de recunoaştere bazat pe diferenţa de scoruri

Tabelul 5.6: Rate de recunoaştere pentru setul de test 3DNS (1C - cu deplasarea RNC_1 deasupra imaginii, 2C - cu deplasarea RNC_2 deasupra imaginii, 1+2C - cu deplasarea ambelor RNC deasupra imaginii, simplu - fără deplasarea clasificatorilor).

	simplu(%)	1C(%)	2C(%)	1+2C(%)
scor maxim	91.46	71.54	69.85	71.54
diferenţe	76.49	76.76	92.75	93.36
scor maxim & diferenţe	83.40	83.74	93.63	**93.77**

Arătam înainte că nu putem să ne bazăm decizia selectării unui anumit clasificator pe scorul maxim oferit de acesta deoarece nu am antrenat clasificatorii cu exemple negative. Mai mult, prin antrenarea reţelelor numai cu exemple pozitive, RNC vor clasifica orice imagine de la intrare într-una din cele mai asemănătoare clase cunoscute. Aceasta înseamnă că dacă un clasificator primeşte la intrare o imagine care este reprezentarea uneia dintre clasele pe care le cunoaşte, atunci ieşirea neuronului corespunzător clasei va avea o valoare apropiată de 1, iar celelalte ieşiri vor fi considerabil mai mici, apropiate de -1. Putem utiliza acest aspect al reţelelor neuronale artificiale pentru a deriva o nouă metodă de a decide dacă o imagine aparţine sau nu vreuneia din clasele cu care a fost antrenată reţeaua. Astfel, diferenţa dintre cele mai mari două scoruri trebuie să fie considerabilă pentru ca imaginea să aparţină unei clase învăţate. Se pot imagina cazuri în care această metodă nu va funcţiona, de exemplu când imaginea de la intrare nu aparţine niciuneia din clase, dar seamănă destul de mult cu

BUPT

una din ele, iar cu restul nu. Un alt caz ar fi acela când există clase foarte asemănătoare, iar o imagine care seamănă cu una din ele va semăna, evident, şi cu celelalte clase, generând din acest motiv scoruri mari şi apropiate. Oricum, se poate presupune că aceste cazuri sunt excepţii de la regulă, fiind suficient de rare pentru ca metoda să fie eficientă.

Considerând observaţiile pe care le-am făcut în urma studierii erorilor generate de Algoritmul 4 pentru cazurile $\overline{X1}$, am implementat Algoritmul 5, întărind condiţia dacă cel puţin una din cifrele pe care le detectează ca posibil existente RNC_2 este 1.

Contrar cu primul sistem de recunoaştere care a folosit doar scorul maxim al fiecarei RNC, în acest caz mutarea clasificatorilor deasupra imaginilor de intrare a îmbunătăţit rata de recunoaştere cu mai bine de 6% până la 92.82%.

```
foreach componentă din imagine do
|  rez1[ ]=RNC1(componentă);
|  rez2[ ]=RNC2(componentă);
|  sortează descrescător rez1[ ] şi rez2[ ];
|  dif1 = rez1[0].score − rez1[1].score;
|  dif2 = rez2[0].score − rez2[1].score;
|  if u=1 or z=1 then
|  |  if dif1 ≥ dif2 then
|  |  |  return rez1[0].class;
|  |  end
|  |  else
|  |  |  return rez2[0].class;
|  |  end
|  end
|  else
|  |  if dif1 ≥ 2 · dif2 then
|  |  |  return rez1[0].class;
|  |  end
|  |  else
|  |  |  return rez2[0].class;
|  |  end
|  end
end
```

Algorithm 5: Sistem de recunoaştere bazat pe diferenţa de scoruri (*z* - cifra zecilor, *u* - cifra unităţilor).

Sistem de recunoaştere bazat pe scor maxim şi pe diferenţe de scoruri

Pentru a vedea dacă se pot obţine rezultate mai bune combinând metodele anterioare, am implementat Algoritmul 6. Pentru fiecare clasificator am adunat cel mai bun scor cu diferenţa dintre cele mai bune două scoruri. Cel mai bun rezultat, **93.77%**, reprezentând 92 de şiruri incorect clasificate, a fost obţinut când am aplicat translaţii asupra clasificatorilor. Acest rezultat este cu 3 până la 18% mai bun decât toate rezultatele din [44] obţinute cu numeroase arhitecturi de reţele neuronale, deşi acelea au fost antrenate şi cu exemple negative. Chiar dacă comparăm metoda prezentată cu cea mai bună din [44], diferenţa este de doar 3.05%,

BUPT

şi aceasta în contextul în care noua metodă nu foloseşte nici segmentare, nici antrenare cu exemple negative.

```
foreach componentă din imagine do
|   rez1[]=RNC1(componentă);
|   rez2[]=RNC2(componentă);
|   sortează descrescător rez1[] şi rez2[];
|   dif1 = rez1[0].score − rez1[1].score;
|   dif2 = rez2[0].score − rez2[1].score;
|   if u=1 or z=1 then
|   |   if rez1.score + dif1 ≥ rez2.score + dif2 then
|   |   |   return rez1[0].class;
|   |   end
|   |   else
|   |   |   return rez2[0].class;
|   |   end
|   end
|   else
|   |   if rez1 + dif1 ≥ rez2 + 2 · dif2 then
|   |   |   return rez1[0].class;
|   |   end
|   |   else
|   |   |   return rez2[0].class;
|   |   end
|   end
end
```

Algorithm 6: Sistem de recunoaştere bazat pe scor maxim şi pe diferenţe de scoruri (*z* - cifra zecilor, *u* - cifra unităţilor).

Algoritmul poate fi îmbunătăţit mai departe prin complicarea regulilor de selecţie a rezultatului unuia dintre clasificatori, dar işi va pierde simplitatea. Analiza erorilor arată că RNC sunt foarte bune în a recunoaşte îmaginile de la intrare, dar regulile care selectează rezultatul unuia dintre clasificatori nu oferă totdeauna rezultate corecte. Cele 92 de erori pot fi împărţite în cinci tipuri:

- 4 erori de segmentare, toate fiind cauzate de imagini scanate deficitar,
- 8 erori cu imagini care conţin trei cifre concatenate. Sistemul de recunoaţere nu tratează cazurile de imagini cu mai mult de două cifre parţial suprapuse.
- 23 erori ale RNC_1. Aproximativ jumătate dintre ele sunt cauzate de o subreprezentare a specimenului din clasa respectivă în setul de antrenare, iar celelalte de imagini foarte confuze.
- 26 erori ale RNC_2. Majoritatea lor se datorează faptului că cele două cifre parţial suprapuse sunt extrem de diferite în înălţime (chiar cu mai mult de 200%), iar antrenarea RNC_2 a fost efectuată cu perechi de cifre diferite în înălţime prin maximum 10%.
- 63 erori generate de alegerea defectuoasă a clasificatorului. Acestea ar putea fi corectate complicând regulile de selecţie sau, preferabil, prin antrenarea şi cu cu exemple negative.

BUPT

Tabelul 5.7: Rate de recunoaştere pentru setul de test 6DNS (1C - cu deplasarea RNC_1 deasupra imaginii, 2C - cu deplasarea RNC_2 deasupra imaginii, 1+2C - cu deplasarea ambelor RNC deasupra imaginii, simplu - fără deplasarea clasificatorilor).

	simplu(%)	1C(%)	2C(%)	1+2C(%)
scor maxim	88.72	89.73	47.93	50.58
diferenţe	56.08	56.83	91.91	92.45
scor maxim & diferenţe	66.42	67.51	91.50	**92.52**

Pentru a verifica dacă cel de-al doilea clasificator - cel la nivel de două cifre - contribuie la îmbunătăţirea ratei de recunoaştere, am rulat experimentele dezactivând 2RNC şi lasând activ doar clasificatorul la nivel de cifră. Au fost identificate corect 89.16% din imagini în cazul în care nu am deplasat imaginea în câmpul de intrare al clasificatotului, şi 89.30% dacă am utilizat deplasările. Se poate observa că ambele scoruri sunt cu peste 4% mai mici decât dacă s-a folosit şi 2RNC.

5.4 Experimente pe setul 6DNS

Cele mai bune rezultate (Tabelul 5.7) au fost obţinute şi de această dată aplicând translatarea imaginilor în câmpul de intrare al clasificatorilor. De asemenea, combinaţia dintre scorul maxim şi diferenţa dintre primele două scoruri s-a dovedit a fi cea mai performantă.

Din cele 1471 de imagini ale setului 6DNS, 110 imagini au fost incorect identificate, reprezentând 7.48% din setul de test 6DNS. Analiza rezultatelor a arătat că şi în cazul acestui set de test cele mai multe erori provin din alegerea eronată a clasificatorului. De asemenea, am observat că clasificatorul la nivel de două cifre este mai susceptibil la erori decât clasificatorul la nivel de cifră. Aceasta este normal deoarece complexitatea imaginilor ce trebuie învăţate este mult mai mare. Un alt motiv este dimensiunea mai mică a câmpului de intrare a clasificatorului la nivel de două cifre.

Faţă de rezultatele obţinute în [44] pe setul 6DNS, rata de 92.52% este net superioară celei de 63.7% cât au obţinut cu perceptronul multistrat antrenat fără exemple negative. Cel mai bun rezultat din [44] (obţinut cu Support Vector Classifier with Radial Basis Functions Gaussian kernel), 96.74%, este cu 4.22% mai bun decât cel obţinut cu metoda prezentată, dar foloseşte atât segmentare, cât şi antrenare cu exemple negative.

Am verificat şi în acest caz comportarea sistemului de recunoaştere în cazul dezactivării 2RNC. Lasând activ doar clasificatorul la nivel de cifră, au fost identificate corect 91.37% din imagini în cazul în care nu am deplasat imaginea în câmpul de intrare al clasificatotului, şi 91.91% dacă am utilizat deplasările. Faţă de setul 3DNS, unde îmbunătăţirea adusă de includerea 2RNC a fost de peste 4%, majorarea ratei de recunoaştere s-a diminuat la 0.61% în cazul setului 6DNS. O explicaţie posibilă ar fi că setul 6DNS conţine imagini mai dificil de recunoscut decât setul 3DNS.

BUPT

Tabelul 5.8: Rate de recunoaştere pentru setul de test 10DNS (1C - cu deplasarea RNC_1 deasupra imaginii, 2C - cu deplasarea RNC_2 deasupra imaginii, 1+2C - cu deplasarea ambelor RNC deasupra imaginii, simplu - fără deplasarea clasificatorilor).

	simplu(%)	1C(%)	2C(%)	1+2C(%)
scor maxim	82.69	85.31	20.27	23.23
diferenţe	33.83	34.05	91.80	**92.82**
scor maxim & diferenţe	49.09	49.77	89.64	91.80

5.5 Experimente pe setul 10DNS

Rezultatele obţinute pe acest set de test pot fi urmărite în Tabelul 5.8. Erorile mari obţinute când deplasarea imaginii în câmpul 2RNC a fost oprită arată că dimensiunea de 21x13 pixeli pentru imagini de două cifre este prea mică. Această observaţie este susţinută şi de faptul că deplasarea imaginilor în câmpul clasificatorului la nivel de cifră îmbunătăţeşte foarte puţin rata de recunoaştere. Probabil din cauză că în acest caz imaginile sunt mult mai mari, 29x29 pixeli, motiv pentru care clasificatorul este mai robust. Autorii din [44] nu au prezentat rezultatele pentru acest set pe motiv că niciunul din articolele anterioare nu prezenta, de asemenea, rezultatele pe acest set. Din acest motiv nu se poate compara rezultatul obţinut în cazul setului 10DNS.

5.6 Concluzii

Ratele de eroare obţinute au fost de 0.86% pentru o cifră, respectiv de 5.35% pentru două cifre, ceea ce arată că RNC reprezintă o bună alegere atât pentru clasificatorul dedicat cifrelor, cât şi pentru clasificatorul dedicat perechilor de cifre parţial suprapuse.

Deşi RNC nu au fost antrenate cu exemple negative, rezultatele obţinute pentru seturile de test 3DNS şi 6DNS sunt mult mai bune decât cele obţinute anterior în domeniu, cu reţele neuronale artificiale. Diferenţele ajung până la câteva zeci de procente în cazul reţelelor antrenate fără exemple negative.

Utilizarea combinaţiei de clasificatori s-a dovedit a fi eficientă, rata de recunoaştere crescând faţă de sistemul ce utilizează numai clasificatorul de cifre cu 4.47% pentru setul 3DNS, respectiv cu 0.61% pentru setul 6DNS.

Metoda propusă oferă rate de recunoaştere cu numai câteva procente (3.03% pentru 3DNS şi 4.42% pentru 6DNS) mai mici decât cele mai bune rezultate obţinute anterior [44]. Având în vedere că rezultatele sunt obţinute fără a aplica segmentarea şi fără antrenare cu exemple negative, ratele de recunoaştere pot fi considerate foarte bune.

Din analiza erorilor rezultă următoarele două observaţii: clasificatorul la nivel de perechi de cifre parţial suprapuse ar trebui îmbunătăţit şi ambii clasificatori ar trebui antrenaţi cu exemple negative. Pentru a îmbunătăţi setul TDNS trebuie detaliată metoda de unire automată a cifrelor, prin crearea de cazuri specifice pentru $\overline{1X}$ şi $\overline{X1}$. De asemenea, trebuie mărite imaginile deoarece dimensiunea de 21×13 este prea mică pentru reprezentarea suficient de clară a imaginilor. Antrenarea cu exemple negative este foarte importantă pentru ambii clasificatori, nemaiexistând apoi alegeri greşite de clasificator. Din analiza manuală a erorilor rezultă că dacă totdeauna s-ar alege clasificatorul potrivit imaginii (RNC_1 pentru cifre,

BUPT

respectiv RNC_2 pentru perechi de cifre), rata de recunoaştere ar deveni mai bună decât cea mai bună existentă în momentul de faţă în domeniu.

O mare parte a rezultatele au fost publicate în [17] şi au fost prezentate la SYNASC 2008.

BUPT

Capitolul 6

Optimizarea vitezei de execuţie

Având în vedere că plăcile grafice folosite pentru accelerarea calculelor în virgulă mobilă au apărut pe piaţă la sfârşitul lunii iunie 2008, precum şi dificultăţile legate de scrierea unor programe pe arhitecturi complet noi şi extrem de diferite faţă de arhitectura x86, acest capitol prezintă numai rezultate parţiale, pentru finalizarea portării tuturor aplicaţiilor pe plăci grafice fiind necesare cel puţin încă câteva luni. Chiar dacă acest capitol nu aduce nimic nou în privinţa performanţelor sistemului de clasificare, neavând legătură explicită cu rata de recunoaştere, am considerat că este util a prezenta şi aceste rezultate parţiale deoarece tehnologia folosită reprezintă viitorul pentru majoritatea domeniilor din ştiinţă care au nevoie de putere foarte mare de calcul, şi cu siguranţă în domeniul reţelelor neuronale. Mai multe detalii vor fi prezentate în secţiunile dedicate accelerării vitezei de execuţie cu ajutorul plăcilor grafice.

Clasificatorul 2RNC a fost antrenat cu o versiune modificată a programelor open source din [3]. Din păcate, acest cod a fost scris într-un mod mai mult "didactic", decât având ca ţintă performanţa. Oricâte modificări i-am făcut în ideea de a-i îmbunătăţi viteza, am ajuns la o limită sub care nu mai puteam scădea timpul de rulare decât dacă rescriam de la zero tot programul. Problema şi mai neplăcută era că dacă măream dimesiunea reţelei neuronale peste un anumit nivel, programul începea să funcţioneze haotic. Încercarea de a descoperi cauzele acestei comportări s-a soldat cu eşec, chiar după multe săptămâni de depanare (sursele programului au aproximativ 250KB de cod C++). Modul în care se aloca memoria în toată ierarhia de clase, precum şi cantitatea imensă care era alocată - mult peste un gigabyte de RAM -, multă din ea chiar pe stivă şi nu în heap, generau acest comportament. Un alt factor care a îngreunat mult depanarea a fost arhitectura implementată multi-threading (cu mai multe fire de execuţie). Cum scrierea unui program de acest gen este foarte complexă şi durează un timp apreciabil, am considerat că utilizarea unor imagini de dimensiuni mai reduse este compromisul optim. Astfel au rezultat toate experimentele şi rezultatele din capitolul precedent.

Una din concluziile din capitolului anterior evidenţia faptul că pentru a obţine rezultate mai bune este nevoie de un clasificator mai performant pentru perechi de cifre parţial suprapuse. Pe lângă antrenarea cu exemple negative, performanţele clasificatorului pot fi crescute prin mărirea dimensiunii imaginilor cu care este antrenat. O dimensiune adecvată pentru a obţine rezultate mai bune pentru 2RNC ar fi 41×29 de pixeli. Aceasta înseamnă 1189 de neuroni pentru stratul de intrare al reţelei, şi, implicit, numere crescute de neuroni şi de hărţi de convoluţie pe celelalte straturi. Faţă de imaginile de 21×13 pixeli cu care a fost antrenat

BUPT

2RNC, imagini care necesită numai 293 de neuroni pentru stratul de intrare, imaginile cu dimensiunea de 41×29 pixeli ar necesita o reţea de peste patru ori mai mare. Timpul de antrenare ar creşte proporţional cu mărimea reţelei. Cum cea mai bună reţea pentru 2RNC a necesitat câteva zile pentru a fi antrenată, timpul pentru noua dimensiune a imaginilor ar ajunge la aproximativ două săptămâni, ceea ce este total nepractic.

Întrucât precizam că portarea programelor nu este finalizată, pentru a putea face comparaţii de viteză este necesară alegerea unei porţiuni de cod reprezentativă pentru reţelele neuronale, cod care apoi să fie implementat în diverse moduri. Reţelele neuronale, ca toţi clasificatorii, au două părţi, una de antrenare şi una de testare. Cum algoritmul de propagare înainte (forward propagation) apare atât la antrenare, cât şi la testare, unde reprezintă chiar întreg pasul de testare, am decis să aleg o parte a acestui algoritm pentru a fi implementată pe diverse arhitecturi. Am ales cel mai complex strat al reţelei neuronale de convoluţie, stratul L3. Acesta este un strat complet conectat, şi nu de convoluţie, dar conţine de departe cele mai multe operaţii, necesitând cea mai mare parte a timpului de rulare a algoritmului de propagare înainte. Am ales stratul L3 al clasificatorului la nivel de cifră, 1RNC, deoarece este mai mare decât cel de la 2RNC. Este bine ca dimensiunea stratului să fie cât mai mare deoarece scopul final al acestei optimizări este acela de a permite antrenarea unor reţele de dimensiuni mai mari.

În toate experimentele următoare timpul a fost măsurat şi raportat pentru 10000 de rulări pentru a putea fi măsurat corect şi, în acelaşi timp, pentru a putea fi raportat în secunde, nu în microsecunde. De asemenea, setul de test al MNIST are 10000 de imagini, testele fiind efectuate pe întreg setul de test.

Stratul L3 conţine 100 de neuroni fiecare din ei având 1251 de conexiuni, una pentru valoarea de prag (bias) şi 1250 pentru cei 1250 de neuroni de pe stratul L2. În total există 125100 de conexiuni şi tot atâtea ponderi. Calculul acestui strat este format din evaluarea a 100 de produse scalare dintre vectori cu 1251 de componente, urmată de aplicarea funcţiei de activare asupra rezultatului fiecărui produs scalar.

6.1 Variante CPU ale optimizării algoritmului de propagare înainte

În aceast subcapitol vor fi prezentate rezultatele obţinute prin optimizarea codului ce rulează numai pe procesor, fără plăci grafice. Întâi se va prezenta algoritmul iniţial, cel care foloseşte multe clase şi vectori STL (Standard Template Library). Timpul său de rulare va fi folosit ca etalon pentru comparaţiile următoare. Apoi vor fi prezentate două variante care nu folosesc ierarhii de clase şi, mai important, vectori STL, ci memorie alocată dinamic cu new/malloc. Ultima variantă este implementată în limbaj de asamblare şi foloseşte instrucţiuni SSE3.

6.1.1 Varianta open source modificată

Această variantă este cea cu care am efectuat experimentele, dar care s-a dovedit a fi imposibil de extins la reţele mai mari din cauza timpului mare de rulare şi din cauza comportării eronate în cazul creşterii dimensiuni reţelei neuronale. Codul iniţial este preluat din [3] şi modificat pentru a putea fi utilizat pentru experimentele pe care le-am efectuat. În Figura 6.1 este prezentat codul sursă al algoritmului de propagare înainte. Se poate observa cât de

BUPT

încărcat este codul şi cât de mult face apel la vectori STL. Timpul necesar pentru cele 10000 de imagini ale setului de test a fost de 11.67 secunde. Ca observaţie, timpul necesar pentru întreg algoritmul de propagare înainte pentru toate cele 10000 de imagini ale setului de test din MNIST a fost de 27 de secunde.

```
void NNLayer::Calculate()
{
    ASSERT( m_pPrevLayer != NULL );

    VectorNeurons::iterator nit;
    VectorConnections::iterator cit;

    double dSum;

    for( nit=m_Neurons.begin(); nit<m_Neurons.end(); nit++ )
    {
        NNNeuron& n = *(*nit);  // to ease the terminology

        cit = n.m_Connections.begin();

        ASSERT( (*cit).WeightIndex < m_Weights.size() );

        // weight of the first connection is the bias; neuron is ignored
        dSum = m_Weights[ (*cit).WeightIndex ]->value;

        for ( cit++ ; cit<n.m_Connections.end(); cit++ )
        {
            ASSERT( (*cit).WeightIndex < m_Weights.size() );
            ASSERT( (*cit).NeuronIndex < m_pPrevLayer->m_Neurons.size() );

            dSum += ( m_Weights[ (*cit).WeightIndex ]->value ) *
                ( m_pPrevLayer->m_Neurons[ (*cit).NeuronIndex ]->output );
        }

        n.output = SIGMOID( dSum );
    }
}
```

Figura 6.1: Codul sursă al algoritmului de propagare înainte pentru varianta C++ cu vectori STL.

6.1.2 Varianta C++ fară vectori STL

Această variantă (Figura 6.2) reprezintă prima tentativă de optimizare prin rescrierea programului de antrenare a reţelelor neuronale artificiale pe procesoare x86. Scopul ei a fost acela de a verifica gradul de accelerare a execuţiei faţă de varianta open source modificată, care se baza excesiv pe clase şi pe vectori STL. Datele referitoare la ieşirile neuronilor, precum şi la ponderi au fost restructurate astfel încât accesele la memorie să fie cât mai grupate. Pentru creşterea vitezei acceselor la memorie, toţi vectorii au fost alocaţi la adrese de memorie aliniate la 64 octeţi [5]. S-a încercat eliminarea indirectărilor inutile prin precalcularea valorilor

BUPT

```
for(unsigned l= ;l<nLayers- ;l++)
    {
        double sum =  ;
        double *output, *pondere;
        unsigned2 *conex;

        unsigned nr_conexiuni,nr_neuroni;
        unsigned int idx= , indice= ;

        nr_neuroni=L[l].nrn;
        output=L[l- ].output;
        conex = L[l].c;
        pondere = L[l].p;

        unsigned *pc=(unsigned*)conex;
        for(unsigned i= ;i<nr_neuroni;i++)
        {
            if(pc[ ] ==              ) // adun bias la suma
                sum = pondere[pc[ ]];
            nr_conexiuni=L[l].nrc[i];
            pc+= ;
            nr_conexiuni--;
            for(unsigned j= ;j<nr_conexiuni;j++)
            {
                sum +=  output[pc[ ]] * pondere[pc[ ]];
                pc+= ;
            }
            nr_conexiuni++;
            L[l].outputSum[i]=sum;
            L[l].output[i]=SIGMOID(sum);
            idx+=nr_conexiuni;
        }
    }
```

Figura 6.2: Codul sursă al algoritmului de propagare înainte pentru varianta C++.

necesare. După toate aceste optimizări timpul de rulare pentru stratul L3 a scăzut la 4.39 secunde. Aceasta înseamnă o creştere a vitezei de 2,65 ori faţă de varianta originală.

Ca observaţie, timpul necesar pentru întreg algoritmul de propagare înainte pentru toate cele 10000 de imagini ale setului de test din MNIST a scăzut de la 27 de secunde la 9.97 secunde.

6.1.3 Varianta SSE2

Căutând noi metode pentru a creşte viteza algoritmului, am recurs la limbajul de asamblare. Mai mult de atât, întrucât procesoarele pe care le foloseam, Intel Core2 Duo, dispuneau şi de seturi de instrucţiuni paralele, am scris codul folosind aceste instrucţiuni.

Procesoarele x86 au începând cu Intel Pentium MMX si AMD K6 instrucţiuni SIMD (Single Instruction Multiple Data). Dacă prima extensie adăugată, MMX, a fost pentru numere întregi,

BUPT

au urmat apoi seturi de instrucţiuni pentru calcule cu numere reale: SSE, SSE2, SSE3, SSE4, SSE4.1, SSE4.2, 3DNow! şi variantele sale. Detalii despre aceste seturi de instrucţiuni paralele pot fi găsite în [25].

```
double dotproduct(double *a, double *b, int count)
{
        //xmm0 xmm1 - registri SSE pe 128 de biti (pot lucra cu 4 float sau 2 double sau ...)
        //pentru a stoca rezultatul (pointerul "pr" e necesar in __asm{} deoarece pe "r" nu il accepta desi e pointer si el)
        double r[ ],*pr=r;
        __asm{
                //salvam registrii principali (s-ar putea sa fie inutil)
                pusha
                //incarcam ecx cu valoarea contorului
                mov             ecx,count
                //incarcam adresele vectorilor
                mov             eax,a
                mov             ebx,b
                //xmm0 <- 0 (pt ca A xor A = 0), il vom folosi pe post de acumulator al rezultatelor (doi acumulatori !)
                pxor            xmm0,xmm0       //paralel XOR
        repeta:
                //incarcam in xmm1 doua valori double din tabloul a[]
                movaps          xmm1,XMMWORD ptr [eax]  //transfer din memorie aliniata
                //incarcam in xmm2 doua valori double din tabloul b[]
                movaps          xmm2,XMMWORD ptr [ebx]
                //inmulteste elementele corespunzatoare din xmm1 si xmm2 (doua inmultiri simultane)
                mulpd           xmm1,xmm2               //inmultire paralela
                //acumulam rezultatele inmultirii anterioare
                haddpd          xmm0,xmm1               //adunare orizontala
                //sarim peste doua valori double in a[]
                add             eax,
                //sarim peste doua valori double in b[]
                add             ebx,
                //decrementam contorul (loopnz il mai decrementeaza o data !)
                dec             ecx
                //repetam bucla
                loopnz          repeta
                mov             eax,pr
                //salvam rezultatul xmm0 in vectorul r
                movups          [eax],xmm0              //transfer la adresa nealiniata
                //restauram registrii principali
                popa
        }
        //[illegible] rezultatul final
        return r[ ]+r[ ];
}
```

Figura 6.3: Codul sursă al algoritmului de propagare înainte pentru varianta C++ cu SSE II.

Procesoarele pe care le-am avut la dispoziţie, Intel Core2 Duo 6600 şi Intel Core2 Quad 6600 suportă maxim SSE3, astfel că a trebuit să limitez optimizarea codului la acest set de instrucţiuni. Modelele noi de procesoare Core2 de la Intel, cele în tehnologie de 45nm, suportă setul de instrucţiuni SSE4 care are o instrucţiune foarte importantă, MAD (Multiply and Accumulate), foarte utilă pentru calculul produsului scalar. Dacă ar fi fost disponibil un astfel de procesor, probabil că viteza ar fi fost şi mai mare, dar nu cu mult deoarece aşa cum

BUPT

se poate vedea din Figura 6.3 în bucla de calcul a produsului scalar există şi alte instrucţiuni care nu mai pot fi optimizate.

Deoarece a scrie cod în limbaj de asamblare nu e un lucru facil şi durează destul de mult, am încercat să evit scrierea în asamblor a întregului algoritm de calcul a stratului L3, scriind o rutină care calculează produsul scalar a doi vectori de mari dimensiuni. Întrucât stratul L3 se bazează pe calculul a 100 de produse scalare de vectori de 1251 de dimensiuni, acest mod de evaluare a performanţelor este adecvat şi corect. Figura 6.3 prezintă codul optimizat pentru instructiuni SSE3. Pentru documentare am folosit [52]. Codul sursă este comentat în Figura 6.3, astfel că nu va mai fi explicat din nou. Am implementat aceeaşi rutină pentru produsul scalar şi în C++, activând toate optimizările pentru viteză ale compilatorului VS C++ 2008, inclusiv folosirea de instrucţiuni SSE3. În urma măsurării timpilor de execuţie am constatat că varianta scrisă în limbaj de asamblare este de 1.3 ori mai rapidă decât cea scrisă în C++. Extrapolând acest rezultat la problema calculului stratului L3, s-ar obţine 3.37s, adică o accelerare de 3.46 ori faţă de algoritmul iniţial.

Trebuie menţionat că am utilizat valori de tip double, astfel că instrucţiunile SSE au prelucrat câte două valori simultan. Puteam folosi şi valori de tip float, caz în care calculele propriu-zise s-ar fi accelerat de două ori deoarece instrucţiunile SSE ar fi prelucrat câte patru valori float simultan. Pe ansamblu, însă, accelerarea nu ar fi fost mult mai mare deoarece algoritmul de calcul al produsului scalar este limitat de accesele la memorie.

Optimizările pe procesor ar putea continua cu scrierea de cod paralel, care să folosească mai multe fire de execuţie. Nu am evaluat această posibiltate deoarece numărul de nuclee dintr-un procesor este de maxim 4, iar plăcile grafice au sute de nuclee de procesare, astfel că este imposibil ca orice optimizare pe procesor să fie mai rapidă decât cea pentru plăcile grafice.

6.2 Variante GPU ale optimizării algoritmului de propagare înainte

În ultimii ani viteza de procesare a plăcilor grafice a crescut extrem de mult faţă de viteza procesoarelor de uz general. Iniţial plăcile grafice au fost proiectate pentru a efectua foarte multe operaţii grafice în paralel. Deşi aveau o putere de calcul destul de mare, aceasta nu putea fi folosită decât foarte dificil - prin transformarea problemelor respective în operaţii grafice - şi în alte domenii. Jocurile şi grafica 3D sunt cele care vând cele mai multe plăci grafice performante, astfel că firmele producătoare au proiectat cipuri capabile de a efectua un număr din ce în ce mai mare de operaţii. Grafica 3D foloseste intensiv operaţii cu numere reale, de obicei în simplă precizie. Operaţiile sunt simple, de obicei adunări şi înmulţiri, şi mult mai rar complicate: radical, funcţii trigonometrice etc. În timp, cipurile grafice s-au trasformat în procesoare grafice (GPU - Graphics Processing Unit), devenind mult mai flexibile, putând fi programate pentru a executa şi alte operaţii decât cele dedicate graficii.

Pentru a putea efectua un număr atât de mare de operaţii există două metode, un procesor unic la o frecvenţă foarte mare sau multe procesoare la frecvenţe moderate. Prima variantă este folosită de procesoarele de uz general, iar tactul acestora nu a depăşit niciodată limita de 4GHz. Puterea consumată creşte rapid cu creşterea frecvenţei, tehnologia actuală nefiind capabilă de a oferi frecvenţe mult mai mari la consumuri rezonabile de curent, de exemplu un maxim de 150W. A doua variantă s-a dovedit a fi mult mai practică, zeci, chiar

BUPT

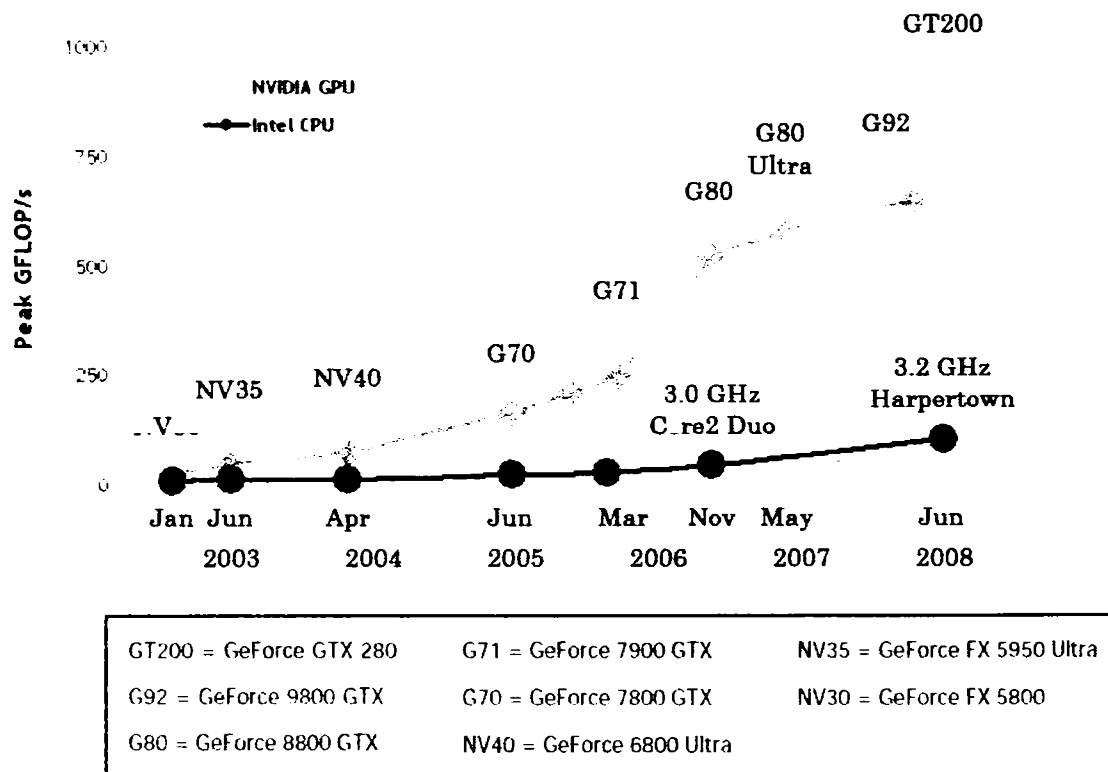


Figura 6.4: Evoluţia performanţelor procesoarelor şi a plăcilor grafice. Preluare după [54].

sute de procesoare simple ce rulează la frecvenţă redusă pot efectua împreună un număr impresionant de calcule în virgulă mobilă. În Figura 6.4 sunt prezentate vitezele în GFLOPS ale plăcilor grafice de la NVIDIA, precum şi a procesoarelor de la Intel. Se poate observa foarte clar că pentru calcule masive plăcile grafice sunt înaintea procesoarelor cu câteva generaţii, ceea ce se traduce în 5-10 ani. Cealaltă firmă producătoare de plăci grafice, AMD-ATI, a scos pe piaţă în ultimele luni plăci grafice chiar şi mai performante: ATI Radeon HD 4850 atinge 1 TFLOPS, ATI Radeon HD 4870 - 1.2TFLOPS, şi cea mai nouă, ATI Radeon HD 4870X2, - 2.4TFLOPS. Un alt aspect important în care plăcile grafice excelează este lăţimea de bandă la memoria plăcii grafice, care este cu aproximativ un ordin de mărime mai mare decât cea a procesoarelor la memoria RAM a sistemului. Trebuie avut în vedere că absolut toate cifrele prezentate în graficele anterioare reprezintă maxime teoretice - atât pentru GPU-uri, cât şi pentru CPU-uri -, în practică performanţele obţinute depinzând mult de problemă şi de abilitatea progamatorului. Însă, chiar şi ineficient programate, GPU-urile sunt în marea majoritate a cazurilor de câteva ori mai rapide decât CPU-urile.

Avantajele şi dezavantajele GPU faţă de CPU pot fi observate analizând Figura 6.6. Procesoare moderne dedică un spaţiu foarte mare memoriei cache (60-90% din numărul de tranzistoare) şi relativ puţine tranzistoare nucleelor de procesare a datelor care sunt în număr de maxim patru (în viitorul apropiat firma Intel va scoate pe piaţă procesoare cu şase nuclee). În schimb, GPU-urile folosesc cele mai multe tranzistoare pentru unităţile de calcul şi foarte puţine pentru memoria cache. Sumarizând, CPU-urile au puţine nuclee de execuţie, dar care rulează la frecvenţe ridicate, au un număr mare de instrucţiuni şi sunt foarte flexibile,

BUPT

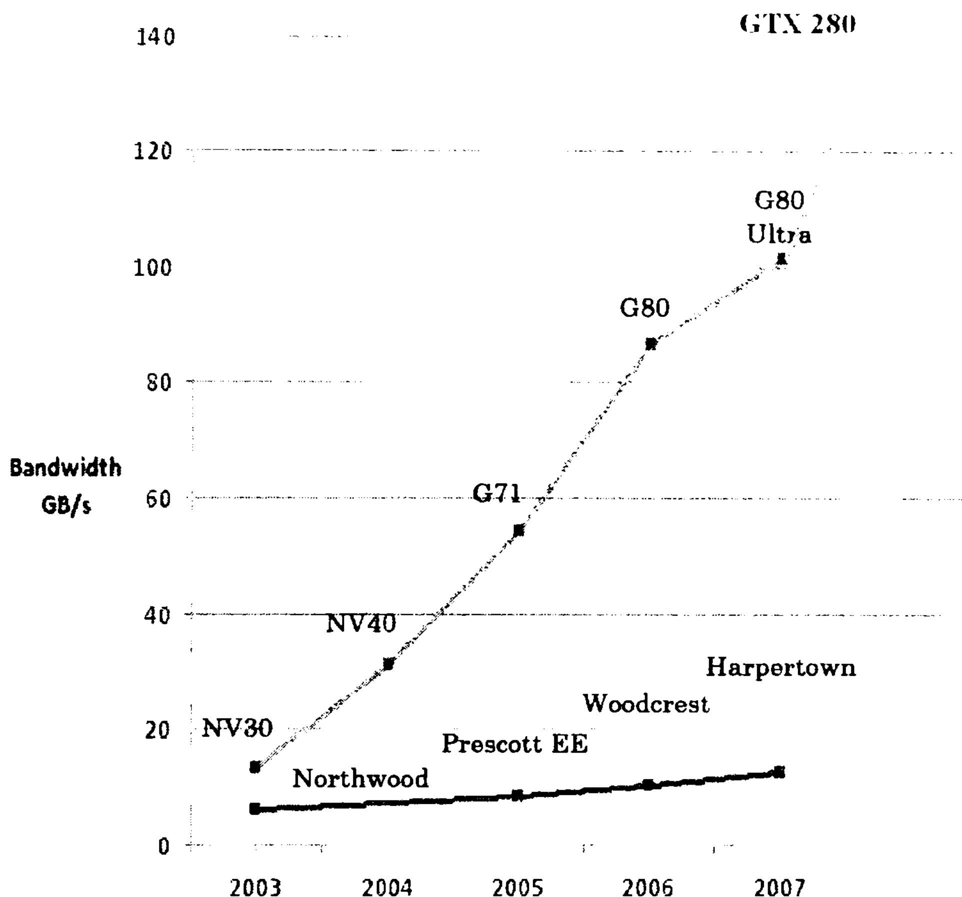


Figura 6.5: Comparaţie între evoluţiile lăţimii de bandă pentru memoria sistem (RAM) şi pentru memoria globală a plăcilor grafice. Adaptare după [54].

au o memorie cache considerabilă -ultimele procesoare de la Intel au 12MB cache L2- ce le oferă acces rapid la date aflate la adrese aleatoare de memorie. GPU-urile au foarte multe nuclee de execuţie - ATI 4870X2 are 2×800 nuclee dispuse în două cipuri - care rulează la frecvenţe scăzute, nucleele sunt simple şi nu oferă flexibilitatea celor din CPU, memoria cache este foarte puţină, accesul la memoria globală fiind foarte rapid numai dacă se face la adrese consecutive de memorie.

O observaţie importantă este că atât pentru CPU, cât şi pentru GPU, raportul dintre numărul de operaţii pe secundă şi lăţimea de bandă este aproximativ acelaşi, 7. Acest lucru arată că pentru a folosi la maxim puterea de calcul a unui CPU sau a unui GPU, accesele la memorie trebuie să fie mult mai reduse decât numărul de operaţii asupra datelor. Din acest motiv se spune că problemele sunt limitate de memorie (memory bound) sau limitate de CPU/GPU (CPU/GPU bound).

Odată cu dezvoltarea GPU-urilor, acestea au început să fie denumite GPGPU (General-

BUPT

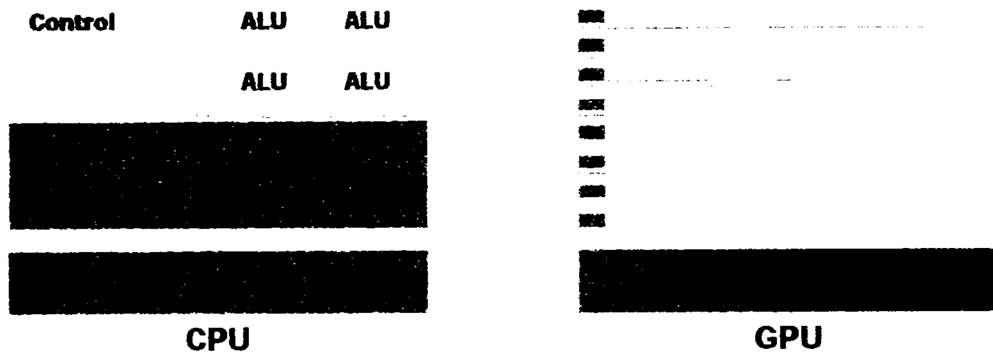


Figura 6.6: Comparaţie între alocarea tranzistoarelor pe suprafaţa de siliciu pentru GPU şi CPU. Preluare după [54].

Purpose GPU) pentru a evidenţia că au devenit mult mai flexibile şi pot fi utilizate şi la alte aplicaţii decât cele grafice. Există şi o organizaţie şi un site dedicat programării GPGPU-urilor. Primele utilizări majore au fost în supercalculatoare distribuite la nivel planetar şi au fost folosite pentru studiul plierii proteinelor - Folding@home - [2], pentru detectarea stelelor neutronice rotitoare - Einstein@Home [1].

Programarea GPU-urilor se face folosind stream computing (programarea fluxurilor de date) şi este un mod particular de paralelism care impune anumite limite felului în care sunt create, executate şi sincronizate firele de execuţie. Toate aceste particularizări fac ideal acest mod de paralelism pentru probleme care au următoarele proprietăţi:

- densitate mare de calcule pentru fiecare element care este prelucrat,
- paralelism la nivelul datelor: aceeaşi rutină (kernel) poate fi aplicată pentru multe date diferite,
- datele sunt consistent localizate: informaţiile necesare unei instanţe a kernel-ului se află distribuite în memorie într-un mod similar cu datele celorlalte instanţe ale kernel-ului. Aceasta constrângere a devenit din ce în ce mai laxă odată cu creşterea flexibilităţii GPU-urilor.

6.2.1 Varianta BROOK+

Limbajul BROOK+ [7] este derivat din limbajul BROOK dezvoltat la Stanford [66] pentru programarea fluxurilor de date (stream computing). BROOK+ este specific plăcilor grafice AMD/ATI şi reprezintă un nivel de abstractizare peste nivelul limbajului de asamblare al procesoarelor grafice.

În secţiunea anterioară precizam că performanţele ultimelor plăci de la ATI sunt impresionante, între 1TFLOPS şi 2.4 TFLOPS, toate având acelaşi cip grafic, RV770 [6], diferind prin frecvenţa acestuia şi prin tipul şi frecvenţa memoriei utilizate (GDDR3 sau GDDR5). Placa pe care am avut-o la dispoziţie este ATI Radeon HD 4850, având 800 de procesoare rulând la 625MHz, pentru o viteză maximă de $625MHz \times 800 processors \times 2FLOP/Hz/processor = 1TFLOPS$. Deşi este cea mai puţin performantă dintre plăcile cu GPU RV770 (seria 48xx),

BUPT

```
kernel void compute_layer(double intr[], double idx_p[][], [illegible] idx_s[][],
                                                    unsigned nrc, out double stari<>)
{
    float2 idx = { , };
    int i;
    double sum =     ;
    float2 step = { , };

    idx.y = indexof(stari).x;
    sum = idx_p[idx.xy];
    idx.xy += step;
    for(i= ; i<nrc; i++)
    {
        sum += idx_p[idx.xy]*intr[idx_s[idx.xy]];
        idx.xy += step;
    }
    stari = SIGMOID(sum);
}
```

Figura 6.7: Codul sursă al algoritmului de propagare înainte pentru varianta BROOK+.

pentru preţul de 199USD oferă o viteză maximă de calcul care este mult superioară faţă de orice CPU existent.

Am început optimizarea algoritmului de propagare înainte pe GPU-uri folosind această placă şi limbajul BROOK+. Firma AMD-ATI are o mare problemă în partea de software a stream computing. Dacă plăcile grafice la nivel hardware sunt excelente, softul (SDK-ul) lasă foarte mult de dorit. Este documentat foarte puţin, implementarea limbajului BROOK+ nu este finalizată, implicând multe limitări în modul în care trebuie scrise programele. Unicele surse de documentare găsite au fost [7] şi [66]. Pe site-ul AMD există un forum dedicat stream computing, dar cei care se ocupă de dezvoltarea limbajului BROOK+ răspund, dacă o fac, cu întârziere destul de mare. Cum nu există alte surse de informaţii, iar a experimenta mai mult sau mai puţin haotic în încercarea de afla cum funcţionează unele aspecte ale limbajului BROOK+ nu este deloc eficient, timpul necesar învăţării acestui limbaj pentru a-l putea aplica cu succes este îndelungat, iar rezultatele pot fi mediocre. La momentul când au apărut plăcile din seria 48xx (iunie-iulie 2008) SDK-ul era la versiunea 1.1. Profiler-ul nu funcţiona pe plăcile din noua serie. Cel mai descurajant aspect este lipsa posibilităţii de depanare automatizată, neexistând un debugger! Astfel că orice depanare trebuie făcută terminând toate firele de execuţie pentru că numai din codul scris pentru CPU se pot face afişări.

Un alt aspect negativ al limbajului BROOK+ este nivelul înalt al acestuia, nivel care nu permite accesul la nivel scăzut asupra hardware-ului. Având în vedere complexitatea unui program cu foarte multe fire de execuţie, accesul la nivel scăzut este obligatoriu pentru a putea scrie programe performante. Compilatorul nu poate decât în puţine cazuri să producă optimizările necesare. Există, într-adevăr, şi un limbaj de asamblare pentru GPU, dar este la fel de slab documentat. Se va vedea în secţiunea referitoare la limbajul CUDA ce diferenţă există între BROOK+ şi CUDA în privinţa documentaţiei, a accesului la nivel scăzut al hardului, uşurinţei de a scrie şi depana programe etc.

Cu toate aceste limitări am încercat scrierea de cod BROOK+ pentru algoritmul de propagare înainte. După îndelungate încercări nereuşite, am ajuns la algoritmul din Figura 6.7.

Nu voi detalia modul de funcţionare al limbajului BROOK+, însă câteva explicaţii asupra

BUPT

codului din Figura 6.7 sunt necesare. Cuvântul cheie **kernel** specifică faptul că această rutină va fi executată în paralel pentru fiecare element al stream-ului de ieşire *stari*. Parantezele <> specifică natura de stream a parametrului *stari*. Se poate conceptualiza în felul următor: dacă stream-ul (vector-ul) are n elemente, atunci pentru fiecare element al său se va executa logic, în paralel câte o instranţă a kernel-lului, pentru un total de n fire de execuţie. Din considerente fizice, numărul de instanţe executate în paralel este limitat de numărul de procesoare şi de alte aspecte ale arhitecturii procesorului grafic. Dacă un stream este precizat folosind parantezele <>, atunci în instanţa curentă poate fi utilizat numai elementul corespunzător instanţei. Dacă este necesar ca un kernel să folosească date de oriunde dintr-un parametru de tip stream, atunci se folosesc parantezele drepte pentru respectivul stream. Recapitulând, <> înseamnă acces numai asupra elementului curent, iar [] - acces la orice element al stream-ului. Este bine a se evita accesul cu paranteze [], deoarece accesele la locaţii aleatoare de memorie este foarte lent.

Funcţia nu este optimizată pentru un strat anume al reţelei, putând fi utilizată pentru orice tip de reţea cu propagare înainte. Parametrul *intr* este un stream unidimensional care conţine ieşirile neuronilor de pe stratul anterior, *idx_p* este un stream bidimensional ce conţine ponderile tuturor conexiunilor neuronilor de pe stratul curent, *idx_s* este un stream bidimensional ce conţine indici ai neuronilor stratului anterior, iar *nrc* este numărul de conexiuni ale fiecărui neuron, considerat în această implementare a fi acelaşi pentru toţi neuronii unui strat.

Tipul **float2** împachetează două valori de tip float, accesibile cu .x şi .y. Variabila *step* este iniţializată cu valoarea {1,0}, deoarece 1 reprezintă pasul necesar trecerii la conexiunea următoare. Este de tip **float2** deoarece tabloul în care se foloseşte acest pas pentru incrementarea indicelui *idx* este bidimensional. Pe rândul 9 se selectează neuronul corespunzător instanţei curente a kernel-ului. Funcţia **indexof()** returnează indicele elementului care este prelucrat de instanţa curentă a kernel-ului. Pe rândul 10 se încarcă bias-ul şi apoi se trece la următoarea conexiune pe rândul următor. Bucla de la rândurile 12-16 calculează restul produsului scalar. La final se aplică funcţia de activare asupra rezultatului produsului scalar şi se stochează această valoare în elementul din stream-ul de ieşire, element corespunzător instanţei kernel-ului.

Ca performanţă, această rutină lasă foarte mult de dorit. Timpul necesar pentru cele 10000 de imagini din setul de test al MNIST a fost de peste 15 secunde pentru L3, deci de câteva ori mai mare decât varianta pe procesor. Acest lucru demonstrează cât de slab este compilatorul de BROOK+ pentru anumite probleme. În cazul de faţă nu este folosită nici măcar 0.5% din puterea procesorului grafic.

În data de 10 septembrie 2008 a apărut noua versiune a SDK-ului de la AMD-ATI. Versiunea 1.2 are o documentaţie îmbunătăţită şi limbajul BROOK+ a fost şi el puţin dezvoltat. Având în vedere aceste aspecte şi faptul că în ultimele luni am început să înteleg mai bine (scriind programe în CUDA pentru GPU-uri NVIDIA - detalii în secţiunea următoare) modul în care trebuie scrise programele pentru a fi eficiente pe arhitecturi GPU, în viitor voi reevalua posibilitatea scrierii algoritmilor de învăţare ai reţelelor neuronale artificiale în limbajul BROOK. Însă, atâta timp cât nu vor fi documentate detaliile arhitecturale ale RV770 şi nu vor fi puse la dispoziţia programatorului prin extensii BROOK+, este total improbabil că se va ajunge la performanţe apropiate de cele de care este capabil la nivel hard GPU-ul RV770.

BUPT

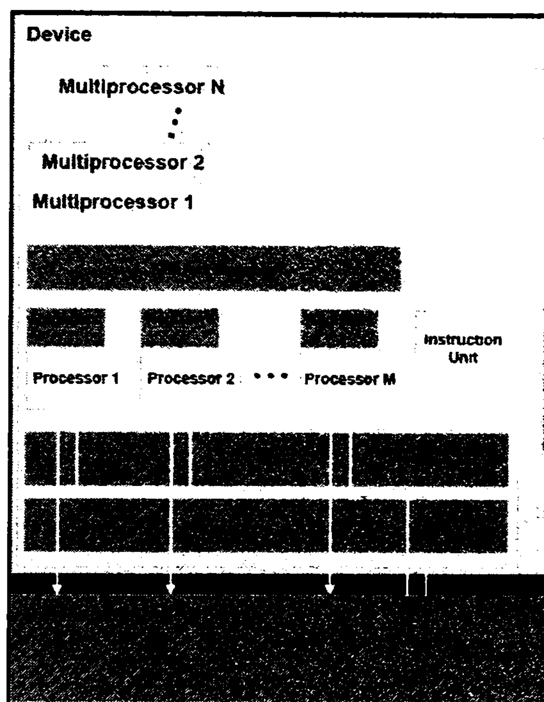


Figura 6.8: Arhitectura GPU-urilor de la NVIDIA (seriile GeForce 8 şi 200). Preluare după [65].

6.2.2 Variantele CUDA

Celălalt mare producător de plăci grafice este NVIDIA. Este şi cel mai mare, având o cotă de piaţă de peste 60%. În anul 2006 a introdus limbajul CUDA (Compute Unified Device Architecture) [55] special conceput pentru GP-GPU (General Purpose GPU). Acesta funcţionează pe toate plăcile grafice din seriile GeForce 8 si GTX 200, în lume existând aproape 100 de milioane de astfel de mini supercomputere, cum mai sunt numite datorită puterii mari de calcul. Spre deosebire de limbajul BROOK+ de la AMD-ATI, CUDA este mult mai dezvoltat şi permite mult mai facil accesul la resursele hardware ale GPU-urilor. De asemenea, există mod debug şi depanarea se face aproape ca în orice compilator de C. În plus, există şi un program de analiză a performanţelor programelor (profiler) scrise în CUDA.

De la NVIDIA am avut la dispoziţie modelul de vârf, GTX 280. Acesta a apărut în luna iunie 2008. Este un procesor care conţine foarte multe tranzistoare, peste 1.4 miliarde, gravate în tehnologie de 65nm. Frecvenţa procesorului este 602MHz, iar a ALU-urilor (unităţi aritmetico-logice) 1297MHz. Conţine 30 de multiprocesoare fiecare având 8 procesoare, pentru un total de 240 de procesoare. Viteza maximă de procesare este $1296MHz \times 240processors \times 3FLOP/Hz/processor = 933GFLOPS$. Memoria folosită este de tipul GDDR3, rulând la 1107MHz şi fiind conectată la GPU printr-un bus foarte lat, de 512 biţi (8 canale a 64 de biţi). Lăţimea de bandă maximă este de $1107MHz \times 64bytes/transfer \times 2transfers/Hz = 141GB/s$. În configuraţie standard, memoria este de 1GB şi este împărţită în 8 canale de 128MB fiecare.

În Figura 6.8 este descrisă la modul general arhitectura ultimelor două generaţii de pro-

BUPT

cesoare grafice, cele din seriile GeForce 8 şi GTX 200. În cazul lui GTX 280 N=30, iar M=8. Fiecare multiprocesor are 16KBytes de memorie partajată ce poate fi accesată de toate cele 8 unităţi de calcul pe care le conţine multiprocesorul.

Memoria partajată este foarte rapidă, apropiată de viteza regiştrilor, scrierile şi citirile durând numai 4 impulsuri de tact. Un aspect important al memoriei partajate este acela că este împărţită în 8 bancuri de câte 1KBytes, accesele la aceste bancuri putându-se efectua simultan. Adresele bancurilor sunt întreţesute la nivel de float, astfel că bancul 0 are adresele 0, 32, 64 ..., bancul 1 - 1, 33, 65 ... etc. Nerespectarea accesării în acest mod a memoriei duce la conflicte de banc, care generează întârzieri în execuţie. GTX 280 are o facilitate în plus, aceea de a permite tuturor celor 8 procesoare dintr-un multiprocesor să citească simultan aceeaşi locaţie de memorie partajată. În literatură această memorie mai este denumită şi memorie cache gestionată de programator (user managed cache). Având în vedere că memoria globală - 1GB GDDR3 - este mult mai lentă decât memoria partajată, unul din modurile în care se pot accelera calculele este acela de a aduce în memoria partajată datele care sunt accesate de mai multe ori.

Fiecare multiprocesor are propriul set de regiştri pe care îi distribuie celor opt procesoare pe care le conţine. Numărul maxim de regiştri disponibili unui multiprocesor este 8192.

Oricare din cele 8 procesoare poate citi din memoria globală, sau din memoria pentru texturi sau memoria constantelor. În cazul ultimelor două tipuri de memorie, accesele se fac prin intermediul unei memorii cache de 8KBytes.

Accesele la memoria globală nu sunt intermediate de nicio memorie cache. Din acest motiv, accesele la memoria globală sunt eficiente numai dacă se fac la adrese consecutive. În plus, e bine ca prima adresă să fie aliniată la nivel de 64 de bytes, cât este dimensiunea magistralei, altfel efectuându-se cel puţin un acces suplimentar. Timpul de acces aleator la memoria globală este de 400-600ns. Accesele consecutive se fac apoi la câte un tact distanţa. În cazul cel mai nefavorabil, când toate accesele sunt aleatoare, neexistând localizare spaţială sau temporală, lăţimea de bandă a memoriei scade de la 141GB/s la aproximativ 8MB/s! Din acest motiv este obligatoriu ca datele să fie accesate astfel încât să existe localizare spaţială şi temporală. Profiler-ul care există în SDK analizează şi aceste aspecte, oferind informaţii utile în păstrarea coeziunii acceselor la memorie.

În continuare va fi descris pe scurt cum funcţionează un program CUDA. Fiecare program este împărţit în foarte multe fire de execuţie (thread-uri) care apoi sunt rulate logic în paralel. De fapt, thread-urile sunt organizate într-un grid bidimensional care este format din mai multe blocuri de thread-uri (Figura 6.9). Fiecare bloc este executat de către un singur multiprocesor, toate procesoarele dintr-un multiprocesor executând aceaşi instrucţiune la un anumit moment de timp. Pe un multiprocesor pot rula mai multe blocuri logic în paralel, dar fizic iterativ. Pentru detalii poate fi consultată documentaţia din SDK [54]. Detalii interesante pot fi găsite şi în seria de articole din [26]. Există foarte multe articole şi prezentări despre modul de funcţionare al CUDA, precum şi despre cum trebuie scrise programele pentru a rula cât mai optim din prisma vitezei, dar nu vor fi înşirate aici pentru a nu aglomera explicaţiile. Întrucât limbajul CUDA este predat şi în cadrul universităţilor, un mod bun de a găsi documentaţie este de a căuta cursurile afişate pe pagina www.nvidia.com/object/cuda_university_courses.html.

Deoarece acest capitol are ca scop optimizarea performanţelor pe diverse arhitecturi, se vor descrie în continuare aspecte ce trebuie avute în vedere pentru a obţine programe performante. Informaţiile oferite reprezintă o sinteză a articolelor şi prezentărilor de pe pagina www.nvidia.com/object/cuda_university_courses.html. Există şi alte aspecte mai puţin importante şi nu atât de eficiente pentru procesul de optimizare, dar nu vor fi expuse deoarece nu

BUPT

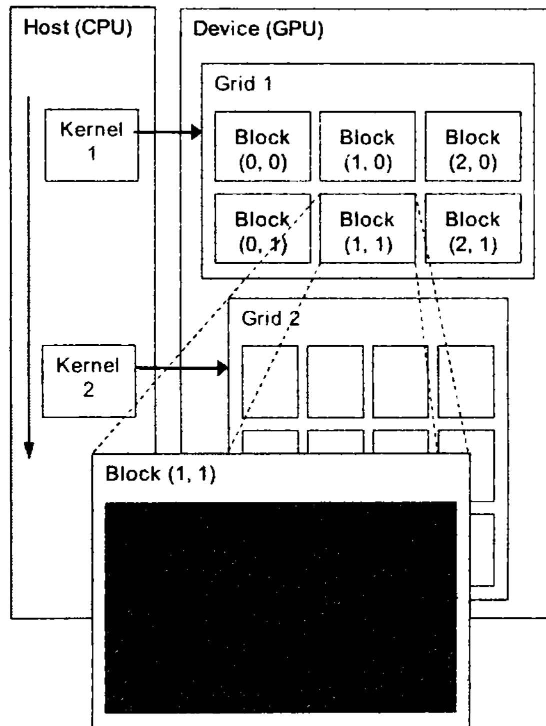


Figura 6.9: Organizarea thread-urilor. Adaptare după [56].

le-am folosit în rutinele ce urmează.

- eliminarea buclelor
  - se face cu directiva de compilare #pragma unroll n, unde n este o constantă care specifică numărul de iteraţii care vor fi efectuate
  - utilă doar dacă numărul de iteraţii este cunoscut aprioric, altfel trebuie efectuate alte artificii de programare
  - am obţinut o reducere a timpilor de execuţie cu aproximativ 40% pentru bucle mari ce calculau produse scalare (de exepmlu în Figura 6.11 rândurile 198-200)
- memoria globală
  - accesele trebuie să fie localizate spaţial şi temporal
  - este bine a fi accesată cât mai puţin
  - accesele aleatoare sunt de peste 100 de ori mai lente decât cele la memoria partajată
- memoria partajată
  - trebuie folosită de fiecare dată când se citesc repetat aceleaşi locaţii de memorie
  - trebuie evitate conflictele de banc

BUPT

  - thread-urile trebuie sincronizate înainte de a citi din memoria partajată
  - fiecare bloc este bine să folosescă cât mai puţină memorie partajată pentru a putea fi rulate logic în paralel mai multe blocuri de thread-uri pe fiecare multiprocesor

- numărul de blocuri ale gridului
  - trebuie să fie suficient de mare pentru ca multiprocesoarele să aibă ce rula
  - cu cât sunt mai multe blocuri alocate unui multiprocesor, cu atât mai bine sunt ascunse latenţele la memoria globală: când un bloc aşteaptă după memoria globală, este pus în coada de aşteptare şi controlul este dat următorului bloc
- numărul de thread-uri din fiecare bloc e bine să fie multiplu de 32 pentru a fi trimise spre execuţie cât mai eficient

Optimizarea kernel-urilor pentru straturile reţelei neuronale

Din multitudinea de variante încercate vor fi prezentate două dintre ele, cele mai performate. Prima variantă este prezentată în figurile 6.10 şi 6.11 şi implementează întreg algoritmul de propagare înainte. În prima figură sunt implementate straturile de convoluţie L1 şi L2.

Stratul L1 conţine 6 hărţi de convoluţie de dimensiunea 13×13. Este creat un grid cu 6 blocuri a 169 de thread-uri fiecare. Specificatorul **__global__** arată că funcţia (kernel-ul) este apelată de către CPU, dar rulează pe GPU. Întrucât ieşirile stratului anterior vor fi folosite la calculul mai multor produse scalare, au fost încărcate în memoria partajată (declaraţia este la rândul 97, iar încărcarea la rândurile 102-104). Deoarece memoria partajată are 16 bancuri, pentru a nu genera conflicte de banc doar thread-urile corepunzătoare primilor 16 neuroni se ocupă de încărcarea datelor în memoria partajată. Variabila *btidx* conţine indicele thread-ului în blocul curent, iar *idx* - indicele în întregul grid. La încărcarea ieşirilor stratului anterior se foloseşte *btidx* deoarece fiecare bloc trebuie să încarce datele întrucât memoria partajată este la nivel de bloc, iar ieşirile stratului anterior sunt necesare în toate blocurile.

Specificatorul **__shared__** precizează că respectiva variabilă este alocată în memoria partajată. Pentru acest strat, şi numărul ponderilor este redus, existând numai 156 de ponderi, câte 26 pentru fiecare hartă de convoluţie. Fiecare bloc trebuie să încarce (rândurile 106-108) numai 26 de ponderi deoarece execută calcule doar pentru una din hărţi. Deoarece sunt numai 26 de ponderi, am lăsat un singur thread să se ocupe de încărcare, thread-ul 0 al fiecărui bloc. Apelul *__syncthreads()* asigură că fiecare bloc are toate datele încărcate în memoria partajată. În acest moment poate începe calculul produselor scalare. La rândul 112 fiecare thread iniţializează valoarea produsului scalar cu valoarea bias-ului, urmând ca pe rândurile 114-117 să fie adunate produsele dintre ponderi şi ieşirile corespunzătoare ale stratului anterior. Cele două for-uri aplică calculul pe un nucleu de 5x5 neuroni. La final (rândul 118) este calculată ieşirea neuronului curent prin aplicarea funcţiei de activare asupra valorii produsului scalat. Rezultatul este memorat în memoria globală.

Stratul L2 conţine 50 de hărţi de convoluţie fiecare a 5×5 neuroni. Fiecare neuron al fiecărei hărţi este conectat cu câte 25 de neuroni din fiecare hartă de pe L1. Se crează un grid cu 50 de blocuri, fiecare bloc calculând valorile neuronilor dintr-o hartă, având, deci, 25 de thread-uri. Ca şi la stratul precedent, ieşirile de pe stratul anterior, precum şi ponderile dintre straturi sunt încărcate în memoria partajată. Avantajul straturilor de convoluţie este că folosesc un număr restrâns de ponderi pe care le reutilizează. Rândurile 129-135 produc

BUPT

încărcarea datelor în memoria partajată şi asigură sincronizarea thread-urilor la final. Rândurile 137-145 calculează valorile neuronilor de pe stratul L2 într-un mod similar cu algoritmul stratului L1. Al treilea for, cel mai interior, selectează hărţile din stratul L1.

```
__global__ void computeConvolutionLayer1(float *prevLN, float *currLW, float *currLN)
{
    float sum= ;
    __shared__ float output[ ][ ];
    __shared__ float w[ ];
    unsigned int btidx=threadIdx.y* + threadIdx.x;
    unsigned int idx = blockIdx.x* * + btidx;

    if(btidx< )
        for(int i=btidx;i< * ;i+= )
            output[i/ ][i% ]=prevLN[i];

    if(threadIdx.x== && threadIdx.y== )
        for(int i= ;i< ;i++)
            w[i]=currLW[blockIdx.x* +i];

    __syncthreads();

    sum = w[ ];
    #pragma unroll 5
    for(int y= ;y< ;y++)
        #pragma unroll 5
        for(int x= ;x< ;x++)
            sum += output[ *threadIdx.y+y][ *threadIdx.x+x] * w[ +y* +x];
    currLN[idx] = SIGMOIDF(sum);
}

__global__ void computeConvolutionLayer2(float *prevLN, float *currLW, float *currLN)
{
    float sum= ;
    __shared__ float output[ ][ ][ ];
    __shared__ float w[ + * ];
    unsigned int btidx=threadIdx.y* + threadIdx.x;
    unsigned int idx = blockIdx.x* * + btidx;

    if(btidx< ){
        for(int i=btidx;i< * * ;i+= )
            output[i/ ][(i% )/ ][(i% )% ]=prevLN[i];
        for(int i=btidx;i< + * ;i+= )
            w[i]=currLW[blockIdx.x*( + * )+i];
    }
    __syncthreads();

    sum = w[ ];
    #pragma unroll 5
    for(int y= ;y< ;y++)
        #pragma unroll 5
        for(int x= ;x< ;x++)
            #pragma unroll 6
            for(int m= ;m< ;m++)
                sum += output[m][ *threadIdx.y+y][ *threadIdx.x+x] * w[ + y* * + x* + m];
    currLN[idx] = SIGMOIDF(sum);
}
```

Figura 6.10: Kernel-urile pentru straturile de convoluţie.

Stratul L3 (Figura 6.11) este total conectat cu stratul anterior. Fiecare din cei 100 de neuroni ai stratului este conectat cu fiecare din cei 1250 de neuroni de pe L2, deci fiecare neuron are 1251 de conexiuni, una fiind pentru bias. Există $100 \times 1251 = 125100$ conexiuni şi tot atâtea ponderi. La acest caz ponderile nu se refolosesc, ci sunt utilizate o singură dată. Oricum, sunt mult prea multe pentru a putea fi încărcate în memoria partajată. Din aceste motive ele vor ramâne în memoria globală şi vor fi citite numai când este nevoie de ele. În schimb, cele 1250 de ieşiri ale neuronilor de pe stratul L2 sunt utilizate fiecare de circa 100 de ori. Fiind puţine şi des accesate este foarte indicat a fi încărcate în memoria partajată. La

BUPT

execuţie este trimis un grid cu un singur bloc a 100 de thread-uri.

Rândurile 185-189 din Figura 6.11 declară variabilele locale necesare şi calculează indicele thread-urilor la nivel de grid. Urmează (rândurile 191-194) încărcarea ieşirilor neuronilor de pe stratul L2 în memoria partajată a blocului, finalizată cu sincronizarea thread-urilor. Este obligatorie această sincronizare deoarece numai primele 16 thread-uri ale blocului de 100 de thread-uri participă la încărcarea memoriei partajate. Fără sincronizare thread-urile ce nu efectuează încărcarea ar trece la execuţia instrucţiunilor următoare care citesc date din memoria partajată care încă nu ar fi într-o stare coerentă, procesul de încărcare nefiind finalizat.

```
__global__ void computeFullConnectedLayer3(float *prevLN, float *currLW, float *currLN)
{
    unsigned int ic= ;
    float sum= ;
    __shared__ float output[ ];
    unsigned int tidx = threadIdx.x;
    unsigned int idx = blockIdx.x*MAX_THREADS + threadIdx.x;

    //loading previous layer outputs in shared memory
    for(int i=tidx;i< ;i+= )
        output[i]=prevLN[i];
    __syncthreads();

    //compute output
    sum = currLW[idx];
    #pragma unroll 1150
    for (ic= ; ic<= ; ic++)
        sum += output[ic- ] * currLW[ic* +idx];
    currLN[idx] = SIGMOIDF(sum);
}

__global__ void computeFullConnectedLayer4(float *prevLN, float *currLW, float *currLN)
{
    float sum= ;
    __shared__ float output[ ];
    unsigned int idx = threadIdx.x;

    //loading previous layer outputs in shared memory
    for(int i=idx;i< ;i+= )
        output[i]=prevLN[i];
    __syncthreads();

    //compute output
    sum = currLW[idx];
    #pragma unroll 100
    for (int ic= ; ic<= ; ic++)
        sum += output[ic- ] * currLW[ic* +idx];
    currLN[idx] = SIGMOIDF(sum);
}
```

Figura 6.11: Kernel-urile pentru straturile complet conectate.

În continuare (rândurile 196-201) se calculează produsul scalar corespunzător şi se aplică funcţia de activare asupra rezultatului. La final are loc scrierea ieşirii neuronului în memoria globală. Sunt necesare câteva precizări pentru operaţiile de la rândul 200. Ca şi la celelalte straturi, *output* se alfă în memoria partajată. În schimb, ponderile *currLW* se află în memoria globală care pentru a fi rapidă trebuie accesată începând de la adrese multiplu de 64bytes,

BUPT

cât este lăţimea magistralei, iar adresele locaţiilor accesate este bine să prezinte localizare spaţială şi temporală. Acest lucru înseamnă că citirile pe care le efectuează thread-urile gridului trebuie să fie concentrate, nu răsfirate aleator prin memorie, iar aceste citiri (accesări, la modul general) trebuie să fie făcute cvasisimultan. Dacă sunt respectate aceste principii, atunci accesele vor fi grupate de către GPU şi executate în paralel. În mod normal, ponderile erau stocate în memorie în felul următor: cele 1251 de ponderi ale neuronului 0, urmate de cele 1251 de ponderi ale neuronului 1, şi tot aşa până la neuronul 99. Considerând că thread-urile rulează în paralel, la un moment dat, fiecare thread accesează ponderea cu acelaşi indice din lista de ponderi corespunzătoare lui. Aceasta înseamnă accese la adresele: $1251 \times 4 \times 0$, $1251 \times 4 \times 1$, $1251 \times 4 \times 2$ etc (4 apare din cauză că sizeof(float)=4). Evident, nu este respectată regula localizării spaţiale. Din acest motiv vectorul de ponderi a fost reordonat în felul următor: primele 100 de valori reprezintă cele 100 de bias-uri ale neuronilor, următoarele 100 de valori reprezintă ponderile spre neuronul 0 al stratului L2 şi tot aşa. În acest mod se respectă regula localizării spaţiale. Numărul 128 apare în loc de valoarea 100 pe rândul 200 deoarece am dorit ca accesele să înceapă la adrese multiplu de 64 bytes. În acest mod nu sunt făcute două accese în loc de unul în cazurile în care datele se întind în două blocuri diferite chiar dacă dimensiunea lor cumulată este de cel mult 64 bytes. Bineînţeles, tabelul de ponderi este creat astfel încât ponderile corespunzătoare unui neuron de pe stratul anterior să se afle la multiplii de 128. Aceasta înseamnă că tabelul conţine câte 28 de elemente neutilizate pentru fiecare 100 de ponderi. Se "pierd" astfel $28 \times 1251 \times sizeof(float) = 140112$ bytes, ceea ce este nesemnificativ având în vedere că memoria globală este de 1GB.

Pentru GTX 280 alinierea la multiplii de 128 nu aduce nicio creştere de viteză deoarece circuitele hard care accesează memoria globală au fost mult îmbunătăţite faţă de seria GeForce 8 unde acestea erau mult mai simple. La seria GeForce 8 alinierea datelor la multiplii de 128 se manifestă într-o creştere observabilă a vitezei de execuţie (10-40%), în funcţie de problemă).

Stratul L4 nu va mai fi detaliat fiind foarte asemănător cu stratul L3. Codul din Figura 6.11 este suficient pentru înţelegerea funcţionării sale.

Cu această variantă am obţinut un timp de rulare pentru kernel-ul stratului L3 pentru cele 1000 de imagini ale setului de test MNIST de 1.60 secunde, de 2.75 ori mai rapid decât algoritmul optimizat pentru procesor. Este o accelerare semnificativă, dar totuşi mică având în vedere diferenţa de performanţă dintre placa grafică si CPU. Era evident că se poate mult mai bine din următoarele două motive:

- am testat algoritmul pe două plăci grafice diferite GTX 280 şi Quadro NVS 290. Deşi ultima are performanţe teoretice de peste 30 de ori mai slabe decât GTX 280, timpii obtinuţi cu ea au fost de numai 3-4 ori mai mari. Aceasta arăta că nu foloseam decât puţin din potenţialul plăcii GTX 280,

- întrucât problema este limitată de viteza memoriei (memory bound problem) am calculat pentru stratul L3 lăţimea de bandă utilizată. Citirea ieşirilor stratului L2, a ponderilor stratului L3, precum şi scrierea ieşirilor stratului L3 totalizează $1250 + 1251 \times 100 + 100 = 126450$ float-uri, adică 505800 octeţi citiţi/scrişi pentru o imagine (acesta ignorând octeţii citiţi inutil pentru păstrarea alinierii datelor). Pentru toate cele 10000 de imagini, înseamnă 5058000000 octeţi, adică 4.71 GB accesaţi în 1.6 secunde. Lăţimea de bandă folosită este de 2.94 GB/s. Având în vedere că GTX 280 are o lăţime de bandă de 141GB/s se poate observa cât de ineficient este utilizată. În schimb, 2.94GB/s este aproape de limita maximă a plăcii NVS 290, arătând încă o dată că modul în care am

BUPT

scris kernel-ul nu este unul scalabil, nefiind capabil de a utiliza resursele plăcii GTX 280.

```
void wrapperComputeFullConnectedLayer3(float *prevLN,
                              float *currLW, float *currLN, float *partial)
{
    dim3 dimGrid( , );                          //30 de blocuri
    dim3 dimBlock(   , );                       //a 100 de thread-uri fiecare
    computeFullConnectedLayer3A<<<dimGrid,dimBlock>>>
                                  (prevLN,currLW,currLN,partial);
    dimGrid=dim3( , );                          //4 blocuri
    dimBlock=dim3(   , );                       //a 25 de thread-uri
    computeFullConnectedLayer3B<<<dimGrid,dimBlock>>>
                                  (prevLN,currLW,currLN,partial);
}

__global__ void computeFullConnectedLayer3A(float *prevLN, float *currLW,
                                  float *currLN, float *partialsum)
{
    #define nrp 42                              //numarul de ponderi dintr-o suma partiala
    float sum=   ;                              //suma partiala
    __shared__ float output[nrp];               //iesiri ale stratului anterior
    unsigned int tidx = threadIdx.x;            //indice thread in bloc
    unsigned int bidx = blockIdx.x;             //indice bloc in grid
    //incarcarea iesirilor stratului anterior in memoria partajata
    if (tidx < nrp)
        output[tidx]=prevLN[bidx*nrp+tidx];
    __syncthreads();
    //calculeaza sumele partiale
    #pragma unroll nrp
    for (unsigned int ic= ; ic<nrp; ic++)
        sum += output[ic] * currLW[nrp*   *bidx+(ic+ )*    +tidx];
    partialsum[   *bidx+tidx]=sum;              //scrierea sumei partiale
}

__global__ void computeFullConnectedLayer3B(float *prevLN, float *currLW,
                                  float *currLN, float *partialsum)
{
    unsigned int idx = blockIdx.x*    + threadIdx.x; //indicele thread-ului
    float sum=currLW[idx];                           //incarcarea bias-ului
    #pragma unroll 30
    for (unsigned int ic= ; ic<  ; ic++)             //adunarea sumelor partiale
        sum += partialsum[ic*  +idx];
    currLN[idx] = SIGMOIDF(sum);
}
```

Figura 6.12: Kernel-urile pentru stratul L3.

În a doua variantă am continuat optimizarea kernel-ului pentru stratul L3. Varianta anterioară era ineficientă deoarece pentru a nu încărca repetat în memoria partajată ieşirile neuronilor de pe stratul L2, am ales o configuraţie cu un singur bloc a 100 de thread-uri. Cum un bloc este trimis la execuţie la un singur multiprocesor, se poate observa cât de ineficient am utilizat cele 30 de multiprocesoare din GTX 280. Ar trebui să avem cel puţin 30 de blocuri pentru a ocupa GPU-ul, asta dacă ignorăm latenţele la memorie pentru ascunderea cărora se precizează că este bine să existe mai multe blocuri disponibile pentru fiecare multiprocesor,

BUPT

în timp ce unul aşteaptă după memoria globală, altul poate deveni activ şi executa operaţii.

Soluţia simplă ar fi fost împărţirea blocului de 100 de thread-uri în 34 de blocuri a 3 thread-uri fiecare, ultimul bloc conţinând numai un thread. Sunt trei dezavantaje. Primul este numărul mic de thread-uri dintr-un bloc, numai 3, faţă de cel puţin 32 cât e recomandat. Apoi, fiecare thread are mult de calculat - un produs scalar între doi vectori a 1251 de elemente. La final, memoria partajată a fiecăruia din cele 34 de blocuri ar fi trebuit umplută cu date din memoria globală. Se observă că această soluţie are ceva comun cu cea anterioară: numărul de 100 de thread-uri.

Soluţia mai complicată rezolvă problema numărului mic de thread-uri prin divizarea calculului produselor scalare în mai multe părţi. Din acest motiv avem nevoie de două kernel-uri, unul care calculează sumele parţiale, şi unul care formează rezultatul final din aceste sume parţiale (Figura 6.12). Comunicaţia dintre kernel-uri se face prin memoria globală folosind parametrul *partialsum*. Deoarece dorim să ocupăm toate cele 30 de multiprocesoare ale GTX 280, vom crea 30 de blocuri. Avem 1250 de produse ce trebuie apoi adunate, deoarece pentru bias nu trebuie să efectuăm nicio înmulţire. Numărul 1250 nu se divide cu 30. Ar fi indicat să folosim o repartizare uniformă deoarece fiecare bloc ar avea exact acelaşi număr de thread-uri şi nu ar trebui să mai punem instrucţiuni condiţionale în kernel. Pentru a rezolva acest aspect, completăm cu neuroni cu ieşire nulă stratul L2. Avem nevoie de $\lceil 1250\ div\ 30 \rceil \times 30 = 1260$ neuroni, deci mai adăugăm 10 neuroni fictivi pe stratul L2. Deci, vom avea 40 de blocuri a 100 de thread-uri fiecare (rândurile 3 şi 4), iar fiecare thread va calcula suma parţială formată din $\lceil 1250\ div\ 30 \rceil = 42$ de produse. Dimensiunea necesară pentru *partialsum* este de $30 \times 100 \times sizeof(float)$, dar vom aloca $30 \times 112 \times sizeof(float)$ pentru a avea datele aliniate la 64 de octeţi.

Kernel-ul 3A calculează sumele parţiale. Începe prin a încărca în memoria partajată cele 42 de ieşiri de pe stratul anterior (rândurile 23-25). Numai primele 42 de thread-uri efectuează încărcarea, restul aşteaptă la bariera de sincronizare (rândul 25). Urmează calculul sumelor parţiale şi scrierea rezultatelor în *partialsum*.

Kernel-ul 3B foloseşte sumele parţiale calculate de 3A pentru a obţine rezultatul final. În urma mai multor teste, am găsit drept configuraţie optimă 4 blocuri a 25 de thread-uri fiecare (rândurile 8-10). La rândul 37 se iniţializează produsul scalar cu valoarea de prag (bias), iar la rândurile 39 şi 40 se adună cele 30 de sume parţiale, urmând ca la rândul 41 să se aplice funcţia de activare şi să se memoreze rezultatul în variabila de ieşire.

Acest mod de a scrie algoritmul de calcul pentru L3 a dus la scăderea timpului de rulare pentru stratul L3 până la 0.2 secunde, ceea ce este de 21.95 ori mai rapid ca varianta C++ optimizată. Folosind calcule similare cu cele de la prima variantă scrisă în CUDA, am obţinut pentru lăţimea de bandă utilizată de acest strat valoarea de 38 GB/s.

6.3 Concluzii

În capitolul dedicat experimentelor am ajuns la concluzia că unul din modurile în care rata de recunoaştere poate fi îmbunătăţită este antrenarea clasificatorilor cu imagini mai mari. Acest lucru implică reţele neuronale mai mari, antrenarea cărora va dura mult mai mult timp. Folosind programele existente deja, timpul de antrenare s-ar întinde pe mai multe săptămâni. De aceea devine necesară căutarea unor soluţii mai rapide.

Pentru că scrierea tuturor programelor pentru fiecare tip de optimizare ar fi durat multe luni de zile, am ales un mod mai simplu de a testa diferitele moduri de optimizare. Mai exact, am

BUPT

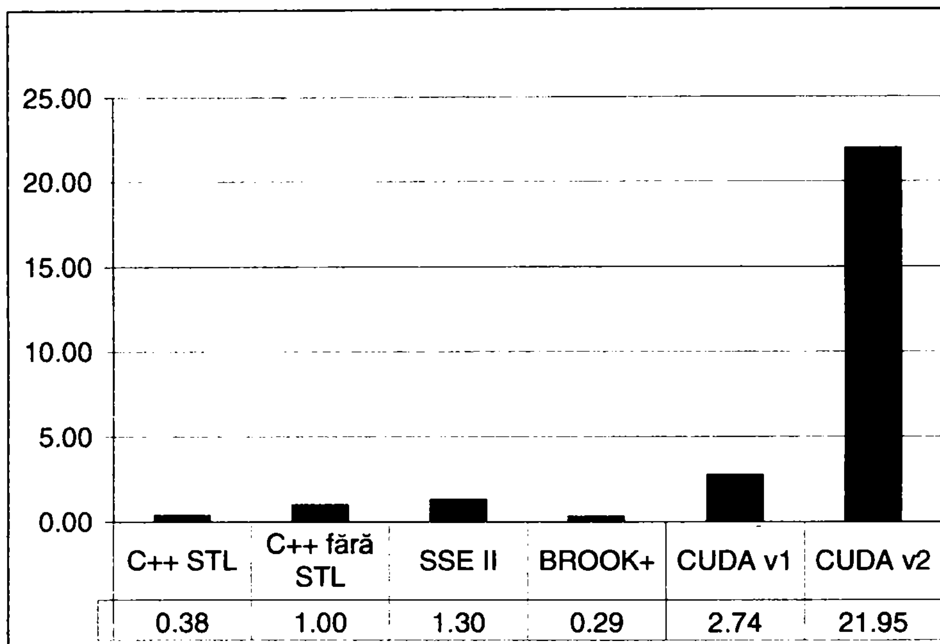


Figura 6.13: Gradul de accelerare a diferitelor variante de kernel pentru stratul L3. Referinţa este considerată varianta C++ optimizată, fără vectori STL.

ales o parte reprezentativă a algoritmului reţelelor neuronale şi am implementat-o în diverse moduri. Pentru experimentele de optimizare am ales stratul L3 al 1RNC (reţeaua neuronală la nivel de cifră). Acest strat este cel mai mare din ambii clasificatori folosiţi.

Am efectuat două tipuri de optimizări folosind arhitectura x86 şi apoi folosind GPU-urile. Prima variantă optimizată folosind numai procesorul a implicat rescrierea în C++ a stratului L3 şi aranjarea datelor de intrare astfel încât să fie cât mai uşor accesibile. S-a obţinut o îmbunătăţire semnificativă faţă de algoritmul iniţial, de 2.63 ori. Această versiune de optimizare a fost folosită drept referinţă pentru celelalte tipuri de optimizări. Deşi ar fi fost avantajos din prisma mărimii cifrelor, nu am folosit drept referinţă varianta iniţială deoarece este un algoritm open source care este scris într-un mod didactic, nu având ca principiu un timp cât mai mic de rulare. Al doilea motiv pentru care am ales algoritmul C++ optimizat pentru a fi referinţă este că doream să compar arhitecturi diferite şi nu este normal a compara o variantă neoptimizată scrisă pe o arhitectură cu una optimizată scrisă pe cealaltă arhitectură.

Pe arhitectura x86 am mai scris şi o variantă în limbaj de asamblare ce foloseşte instrucţiuni SIMD SSE3. Aceasta s-a dovedit a fi mai rapidă ca varianta C++ optimizată, dar nu am folosit-o ca referinţă deoarece nu intenţionam să scriu în limbaj de asamblare variantele pentru plăcile grafice şi s-ar fi ajuns la comparaţii nefondate. În Figura 6.13 sunt prezentate

BUPT

accelerările obţinute cu diferitele variante implementate.

Celălalt tip de arhitectură folosit a fost reprezentat de plăci grafice, aşa numitele GPU-uri. Am efectuat experimente cu GPU-uri atât de la AMD-ATI, cât şi de la NVIDIA. Varianta scrisă în BROOK+ pe placa AMD-ATI este foarte lentă deoarece nu este optimizată, documentaţia oferită de AMD-ATI fiind mediocră. De asemenea, BROOK+ nu oferă acces de nivel scăzut la arhitectura GPU, optimizările fiind lăsate exclusiv în seama compilatorului. Din aceste motive şi din cauza inexistenţei unui debugger, am renunţat la a mai optimiza varianta BROOK+.

GPU-ul de la NVIDIA este mult mai uşor de programat pentru a obţine performanţă. Toate deficienţele întâlnite la BROOK+ nu există la limbajul CUDA folosit pentru GPU-urile NVIDIA. Am implementat două variante de optimizare, cea mai bună atingând o creştere a vitezei de 21.95 ori. În cazul comparării timpului de rulare cu cel obţinut de varianta iniţială, open source, obţinem o creştere de performanţă şi mai mare, de 58.35 ori.

Optimizările efectuate cu ajutorul GPU-urilor au arătat că gradul de accelerare poate fi şi mai mare dacă reţeaua neuronală are dimensiuni mai mari. Accelerarea de 21.95 ori a fost limitată de dimensiunea mică a reţelei simulate.

Pentru a putea implementa eficient o reţea neuronală cu ajutorul GPU-urilor, este obligatorie cunoaşterea şi accesul prin software a arhitecturii interne a procesorului grafic. În caz contrar, performanţele obţinute sunt comparabile (sau chiar mai slabe) cu cele obţinute utilizând CPU-uri.

Concluzia cea mai importantă a acestui capitol este că plăcile grafice reprezintă cea mai performantă soluţie de optimizare a algoritmilor reţelelor neuronale ce poate fi aplicată. Algoritmii implementaţi pe GPU sunt de departe mult mai rapizi decât cei implementaţi pe CPU, o accelerare de peste 20 de ori permiţând antrenarea unor reţele mult mai complexe.

BUPT

Capitolul 7

Concluzii

În acest capitol vor fi prezentate contribuţiile autorului, urmate de sumarizarea concluziilor tezei. La final vor fi expuse principalele direcţii de continuare a cercetării.

7.1 Contribuţii

- **Crearea unui clasificator pentru perechi de cifre parţial suprapuse** şi demonstrarea eficienţei acestuia [20]. Acest clasificator pune bazele construirii unui sistem de recunoaştere a şirurilor numerice cu număr nedefinit de cifre, unite sau nu.

- **Crearea unui nou algoritm de recunoaştere** [17] a şirurilor numerice scrise de mână. Metoda propusă evită folosirea segmentării prin utilizarea a doi clasificatori, unul la nivel de cifră şi unul la nivel de două cifre parţial suprapuse. Ratele de recunoaştere obţinute sunt foarte apropiate de cele mai bune existente în domeniu, cu toate că metoda nu foloseşte nici segmentare, nici antrenare cu exemple negative. Pentru evitarea antrenării cu exemple negative, au fost concepute trei metode simple şi eficiente de selecţie a clasificatorului potrivit conţinutului imaginii. Analiza erorilor arată că în cazul utilizării antrenării cu exemple negative, sistemul de recunoaştere va ajunge la rate de recunoaştere comparabile, probabil chiar mai bune decât cele mai eficiente metode actuale, păstrând, însă, simplitatea dată de neutilizarea segmentării.

- **Accelerarea pasului de propagare înainte** a reţelelor neuronale de peste 20 de ori folosind arhitecturi GPU, cu perspectiva măririi acestui factor pentru reţele şi mai mari. Punerea bazelor creării unui algoritm de antrenare pe GPU pentru reţele neuronale practic imposibil de antrenat pe CPU.

- Crearea unui metode ce generează automat perechi de cifre parţial suprapuse.

- Crearea seturilor de imagini cu perechi de cifre parţial suprapuse TDNS.

- Crearea seturilor de imagini cu cifre LNIST.

- Crearea seturilor de imagini cu şiruri numerice 3DNS, 6DNS şi 10DNS.

BUPT

- Dezvoltarea unui algoritm de extragere automată a câmpurilor din formulare. Algoritmul poate fi adaptat pentru a extrage câmpuri rectangulare din orice tip de formular.
- Conceperea unui algoritm de clusterizare (concatenare) a imaginilor cu părţi de cifre în imagini de cifre sau de perechi de cifre.

7.2 Sumarul tezei

Teza propune o metodă eficientă pentru recunoaşterea şirurilor numerice formate dintr-un număr nedefinit de cifre izolate sau parţial suprapuse.

Întrucât orice metodă trebuie validată, prezentarea începe cu **construirea unor seturi de imagini pentru antrenarea şi testarea clasificatorilor**.

Pentru antrenarea clasificatorului la nivel de cifră am creat seturile din LNIST (66214 imagini pentru setul de antrenare şi 45398 de imagini pentru setul de test) prin extragerea de imagini din NIST SD 19. Acestea sunt construite identic cu cele folosite de autorii din [44, 45] pentru a permite o comparaţie cât mai riguroasă a rezultatelor.

Pentru antrenarea clasificatorului la nivel de două cifre parţial suprapuse (2RNC) am creat seturile de antrenare şi de test TDNS. Din NIST SD 19 au fost extrase 20000 de imagini de cifre (10000 pentru antrenare şi 10000 pentru test) din care s-au creat câte 200000 de imagini cu perechi de cifre parţial suprapuse atât pentru setul de antrenare, cât şi pentru setul de test. Pentru formarea imaginilor cu perechi de cifre parţial suprapuse am dezvoltat un algoritm care uneşte cifrele în funcţie de morfologia acestora.

Dacă imaginile setului LNIST au fost extrase pentru a fi folosite de către clasificatorul la nivel de cifră, pentru cazul general, cel al şirurilor numerice formate din mai multe cifre, au fost construite seturile de test 3DNS, 6DNS şi 10DNS. Şi în construirea acestor seturi am urmărit obţinerea unor seturi cât mai apropiate de cele folosite de alţi cercetători pentru evaluarea performanţelor metodelor de recunoaştere. Cele trei seturi sunt identice cu cele folosite în domeniu, în limita informaţiilor furnizate de autorii care le-au utilizat anterior.

Pentru extragerea din NIST SD 19 a imaginilor necesare construirii seturilor 3DNS, 6DNS şi 10DNS am elaborat un algoritm care foloseşte o variantă a transformatei Hough pentru detectarea liniilor din formularele NIST. După detecţia acestor linii, sunt detectate dreptunghiurile ce încadrează câmpurile dorite. Apoi are loc extragerea propriu-zisă a imaginilor şi se aplică un algoritm de înlăturare automată a posibilelor părţi din dreptunghiul delimitator care au fost extrase în acelaşi timp cu imaginile şirurilor numerice. La final a fost aplicat un proces de inspectare şi curăţare manuală a imaginilor. Algoritmul de extragere automată a imaginilor din câmpurile formularelor a diminuat considerabil timpul necesar construirii pe cale manuală a celor trei seturi de test. De asemenea, algoritmul este destul de general, putând fi adaptat facil pentru alte tipuri de formulare.

După ce seturile pentru antrenare şi testare au fost create, se trece la prezentarea **sistemului de recunoaştere**. Se începe cu antrenarea celor doi clasificatori. Ratele de eroare obţinute sunt de 0.86% pentru o cifră, respectiv de 5.35% pentru două cifre, ceea ce arată că RNC reprezintă o bună alegere atât pentru clasificatorul dedicat cifrelor, cât şi pentru clasificatorul dedicat perechilor de cifre parţial suprapuse.

BUPT

Folosind cei doi clasificatori şi trei metode distincte de seleţie a lor în funcţie de conţinutul imaginii, am implementat sistemul de recunoaştere a şirurilor numerice.

Ratele de recunoaştere obţinute sunt: 93.77% pentru şiruri de trei cifre (setul 3DNS), 92.52% pentru şiruri de şase cifre (setul 6DNS) şi 92.82% pentru şiruri de zece cifre (setul 10DNS).

Deşi RNC nu au fost antrenate cu exemple negative, rezultatele obţinute pentru seturile de test 3DNS şi 6DNS (pentru 10DNS nu există date de comparaţie în literatură) sunt mult mai bune decât cele obţinute anterior în domeniu, cu reţele neuronale artificiale. Diferenţele ajung până la câteva zeci de procente în cazul reţelelor antrenate fără exemple negative.

Utilizarea combinaţiei de clasificatori s-a dovedit a fi eficientă, rata de recunoaştere crescând faţă de sistemul ce utilizează numai clasificatorul de cifre cu 4.47% pentru setul 3DNS, respectiv cu 0.61% pentru setul 6DNS.

Metoda propusă oferă rate de recunoaştere cu numai câteva procente (3.03% pentru 3DNS şi 4.42% pentru 6DNS) mai mici decât cele mai bune rezultate obţinute anterior [44]. Având în vedere că rezultatele sunt obţinute fără a aplica segmentarea şi fără antrenare cu exemple negative, ratele de recunoaştere pot fi considerate foarte bune.

Concluzia experimentelor efectuate este că unul din modurile în care rata de recunoaştere poate fi îmbunătăţită este antrenarea clasificatorilor cu imagini mai mari. Acest lucru implică reţele neuronale mai mari, antrenarea cărora va dura mult mai mult timp. Folosind programele existente deja, timpul de antrenare s-ar întinde pe mai multe săptămâni. Din acest motiv devine necesară căutarea unor **soluţii mai rapide**.

Pentru că scrierea tuturor programelor pentru fiecare tip de optimizare ar fi durat multe luni de zile, am ales un mod mai simplu de a testa diferitele moduri de optimizare. Mai exact, am ales o parte reprezentativă a algoritmului reţelelor neuronale şi am implementat-o în diverse moduri. Pentru experimentele de optimizare am ales stratul L3 al 1RNC (reţeaua neuronală la nivel de cifră). Acest strat este cel mai mare din ambii clasificatori folosiţi.

Am efectuat două tipuri de optimizări folosind arhitectura x86 şi apoi folosind GPU-urile. Prima variantă optimizată folosind numai procesorul a implicat rescrierea în C++ a stratului L3 şi aranjarea datelor de intrare astfel încât să fie cât mai uşor accesibile. S-a obţinut o îmbunătăţire semnificativă faţă de algoritmul iniţial, de 2.63 ori. Această versiune de optimizare a fost folosită drept referinţă pentru celelalte tipuri de optimizări. Deşi ar fi fost avantajos din prisma mărimii cifrelor, nu am folosit drept referinţă varianta iniţială deoarece este un algoritm open source care este scris într-un mod didactic, nu având ca principiu un timp cât mai mic de rulare. Al doilea motiv pentru care am ales algoritmul C++ optimizat pentru a fi referinţă este că doream să compar arhitecturi diferite şi nu este normal a compara o variantă neoptimizată scrisă pe o arhitectură cu una optimizată scrisă pe cealaltă arhitectură.

Pe arhitectura x86 am mai scris şi o variantă în limbaj de asamblare ce foloseşte instrucţiuni SIMD SSE3. Aceasta s-a dovedit a fi mai rapidă ca varianta C++ optimizată, dar nu am folosit-o ca referinţă deoarece nu intenţionam să scriu în limbaj de asamblare variantele pentru plăcile grafice şi s-ar fi ajuns la comparaţii nefondate.

Celălalt tip de arhitectură folosit a fost reprezentat de plăci grafice, aşa numitele GPU-uri. Am efectuat experimente cu GPU-uri atât de la AMD-ATI, cât şi de la NVIDIA. Varianta scrisă în BROOK+ pe placa AMD-ATI este foarte lentă deoarece nu este optimizată, documentaţia oferită de AMD-ATI fiind mediocră. De asemenea, BROOK+ nu oferă acces de nivel scăzut la arhitectura GPU, optimizările fiind lăsate exclusiv în seama compilatorului. Din aceste motive

BUPT

şi din cauza inexistenţei unui debugger, am renunţat la a mai optimiza varianta BROOK+.

GPU-ul de la NVIDIA este mult mai uşor de programat pentru a obţine performanţă. Toate deficienţele întâlnite la BROOK+ nu există la limbajul CUDA folosit pentru GPU-urile NVIDIA. Am implementat două variante de optimizare, cea mai bună atingând o creştere a vitezei de 21.95 ori. În cazul comparării timpului de rulare cu cel obţinut de varianta iniţială, open source, obţinem o creştere de performanţă şi mai mare, de 58.35 ori.

Optimizările efectuate cu ajutorul GPU-urilor au arătat că gradul de accelerare poate fi şi mai mare dacă reţeaua neuronală are dimensiuni mai mari. Accelerarea de 21.95 ori a fost limitată de dimensiunea mică a reţelei simulate.

Pentru a putea implementa eficient o reţea neuronală cu ajutorul GPU-urilor, este obligatorie cunoaşterea şi accesul prin software a arhitecturii interne a procesorului grafic. În caz contrar, performanţele obţinute sunt comparabile (sau chiar mai slabe) cu cele obţinute utilizând CPU-uri.

Observaţia cea mai importantă este că plăcile grafice reprezintă cea mai performantă soluţie de optimizare a algoritmilor reţelelor neuronale. Algoritmii implementaţi pe GPU sunt de departe mult mai rapizi decât cei implementaţi pe CPU, o accelerare de peste 20 de ori permiţând antrenarea unor reţele mult mai complexe.

7.3 Continuări posibile

Pentru mărirea ratei de recunoaştere trebuie îmbunătăţiţi clasificatorii:

- 1RNC:
  - adăugarea în setul de antrenare de exemplare ale cifrei 1 cu linie orizontală în partea de jos. În prezent setul de antrenare nu conţine niciun astfel de exemplar şi toate cifrele 1 scrise în acest mod sunt recunoscute ca fiind 2. Verificarea existenţei a suficiente exemplare din fiecare stil de scriere a cifrelor [42] şi completarea setului de antrenare, dacă este cazul
  - generarea sau colectarea de exemple negative
  - reantrenarea clasificatorului
- 2RNC:
  - mărirea dimensiunii imaginilor setului de antrenare de la 21×13 pixeli la 41×29 pixeli
  - rafinarea algoritmului de formare automată a perechilor de cifre parţial suprapuse prin implementarea cazurilor particulare $\overline{1X}$ şi $\overline{X1}$
  - studierea efectului aplicării deformărilor înaintea procesului de unire a cifrelor
  - generarea sau colectarea de exemple negative
  - reantrenarea clasificatorului

Reantrenarea clasificatorilor (sau cel puţin a unuia dintre ei) cu exemple negative este foarte importantă deoarece ar îmbunătăţi drastic acurateţea alegerii clasificatorului corect. Din analiza manuală a erorilor rezultă că dacă totdeauna s-ar alege clasificatorul potrivit

BUPT

imaginii (RNC_1 pentru cifre, respectiv RNC_2 pentru perechi de cifre), rata de recunoaştere ar deveni mai bună decât cea mai bună existentă în momentul de faţă în domeniu.

Pentru a putea antrena în timp rezonabil (câteva ore, maxim o zi, şi nu săptămâni) clasificatorul 2RNC cu imagini cu dimensiunea de 41×29 pixeli este necesară finalizarea implementării pe GPU a algoritmului de antrenare a reţelei neuronale.

Întrucât există, deşi într-un număr destul de redus, şi imagini care conţin mai mult de două cifre unite, trebuie dezvoltată o metodă de rezolvare a acestui caz.

Algoritmul de extragere a câmpurilor din formulare poate fi îmbunătăţit pentru a diminua cazurile în care trebuie rezolvate manual. Acest aspect este important numai dacă sistemul de clasificare va trebui în viitor să extragă imagini de test din alte formulare. Bineînteles, etapele de extragere a câmpurilor şi cea de recunoaştere a informaţiilor ar putea fi concatenate şi introduse într-un sistem unitar care să proceseze automat formulare cu informaţii numerice.

Studierea implementării clasificatorilor cu SVM (Support Vector Machine) sau alte metode de clasificare. Sunt importante atât rata de recunoştere, cât şi gradul de paralelism al algoritmului clasificatorului în ideea implementării pe GPU.

O continuare interesantă şi foarte utilă ar fi extinderea metodelor de clasificare şi la literele scrise de mână. Întrucât literele sunt de aproximativ trei ori mai multe decât cifrele, clasificatorul la nivel de literă ar avea de trei ori mai multe ieşiri, devenind mai complex. Cel mai mult va creşte gradul de complexitate al clasificatorului la nivel de două litere parţial suprapuse (sau scrise legat) deoarece numărul claselor va ajunge la 900. Din fericire, gradul acesta de complexitate este potrivit cu capacitatea de procesare a plăcilor grafice, şi problema recunoaşterii literelor şi cuvintelor scrise de mână ar reuşi să utilizeze mult mai eficient resursele care acum sunt doar parţial folosite din cauza dimensiunilor prea mici ale problemei recunoaşterii cifrelor şi a perechilor de cifre parţial suprapuse.

Multe din aspectele prezentate se alfă deja în faza de dezvoltare.

BUPT

Anexa 1 - Formatul fişierelor IDX

Acest format este utilizat pentru a stoca tablouri unidimensionale şi multidimensionale. Tipul elementelor poate fi ales din cele mai uzuale tipuri numerice. Elementele dintr-un fişier trebuie să aibă toate acelaşi tip.

Fişierul este structurat în modul următor:

- cod,
- numărul de elemente din dimensiunea 0,
- numărul de elemente din dimensiunea 1,
- numărul de elemente din dimensiunea 2,
- ...
- numărul de elemente din dimensiunea N,
- date.

Codul este un întreg pe 4 octeţi. Primii doi octeţi sunt nuli. Al treilea octet codifică tipul de date memorate:

- 0x08: unsigned byte,
- 0x09: signed byte,
- 0x0B: short (2 bytes),
- 0x0C: int (4 bytes),
- 0x0D: float (4 bytes),
- 0x0E: double (8 bytes).

Al patrulea octet specifică numărul de dimensiuni ale tabloului.

Datele sunt stocate în forma tablourilor C, cu indicele ultimei dimensiuni schimbându-se cel mai des.

Toate numerele întregi sunt reprezentate pe patru octeţi în format MSB (non Intel).

Pentru detalii se poate consulta [40].

BUPT

Anexa 2 - Formatul fişierelor MNIST

Este un caz particular al tipului de fişier idx descris în anexa precedentă. Toate numerele întregi sunt reprezentate în format MSB first (cel mai semnificativ octet este stocat la adresa cea mai mică). Cele patru fişiere din MNIST sunt structurate în felul următor:

- train-images-idx3-ubyte

offset	tip	valoare	descriere
0000	32 bit integer	0x00000803(2051)	tip date
0004	32 bit integer	60000	număr de imagini
0008	32 bit integer	28	număr de rânduri
0012	32 bit integer	28	număr de coloane
0016	unsigned byte	??	valoare pixel
0017	unsigned byte	??	valoare pixel
....			
xxxx	unsigned byte	??	valoare pixel

- train-labels-idx1-ubyte

offset	tip	valoare	descriere
0000	32 bit integer	0x00000801(2049)	tip date
0004	32 bit integer	60000	număr de elemente
0008	unsigned byte	??	etichetă
0009	unsigned byte	??	etichetă
....			
xxxx	unsigned byte	??	etichetă

BUPT

- t10k-images-idx3-ubyte

offset	tip	valoare	descriere
0000	32 bit integer	0x00000803(2051)	tip date
0004	32 bit integer	10000	număr de imagini
0008	32 bit integer	28	număr de rânduri
0012	32 bit integer	28	număr de coloane
0016	unsigned byte	??	valoare pixel
0017	unsigned byte	??	valoare pixel
....			
xxxx	unsigned byte	??	valoare pixel

- t10k-labels-idx1-ubyte

offset	tip	valoare	descriere
0000	32 bit integer	0x00000801(2049)	tip date
0004	32 bit integer	10000	număr de elemente
0008	unsigned byte	??	etichetă
0009	unsigned byte	??	etichetă
....			
xxxx	unsigned byte	??	etichetă

Etichetele au valori între 0 şi 9, iar pixelii între 0 (alb) şi 255 (negru).

BUPT

Bibliografie

[1] Einstein@home - distributed computing. *http://einstein.phys.uwm.edu/.*

[2] Folding@home - distributed computing. *http://folding.stanford.edu/.*

[3] Neuralnet recognition project on the codeproject page, www.codeproject.com/kb/library/neuralnetrecognition.aspx.

[4] Nist special database 19 - nist handprinted forms and characters database. *www.nist.gov/srd/nistsd19.htm.*

[5] MSDN Run-Time Library Reference: _aligned_malloc. http://msdn.microsoft.com/en-us/library/8z34s9c6(vs.80).aspx. *Visual C++ Developer Center.*

[6] AMD. Ati radeon hd 4800 series - gpu specifications. *http://ati.amd.com/products/Radeonhd4800/specs.html.*

[7] AMD. Brook+. *http://ati.amd.com/technology/streamcomputing/AMD-Brookplus.pdf.*

[8] Sven Behnke. *Hierarchical Neural Networks for Image Interpretation*, volume LNCS 2766. Springer, 2002.

[9] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 13th edition, 2005.

[10] Christopher M. Bishop. *Pattern recognition and machine learning.* Springer, 2006.

[11] R. G. Casey and E. Lecolinet. A survey of methods and strategies in character segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7):690–706, July 1996.

[12] K Chellapilla, S. Puri, and Simard P. High performance convolutional neural networks for document processing. In *International Workshop on Frontiers in Handwriting Recognition*, 2006.

[13] K Chellapilla, M. Shilman, and Simard P. Combining multiple classifiers for faster optical character recognition. In *Proc. of the 7th International Workshop on Document Analysis Systems*, volume LNCS 3872, pages 358–367, 2006.

[14] C.E. Cheong, K. Ho-Yon, S. Jang-Won, and J.H. Kim. Handwritten numeral string recognition with stroke grouping. *Proc. of the Fifth International Conference on Document Analysis and Recognition (ICDAR'99)*, (1):745–748, September 1999.

BUPT

[15] Dan Ciresan. Analiza şi prelucrarea imaginii în vederea procesului de recunoaştere cu reţele neuronale artificiale. Technical report, Referat 1, 2004.

[16] Dan Ciresan. Image segmentation methods based on natural clustering algorithms, usable for unconstrained handwriting recognition. *Proc. of the 6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 131–140, 2004.

[17] Dan Ciresan. Avoiding segmentation in multi-digit numeral string recognition by combining single and two-digit classifiers trained without negative examples. *Proc. of 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 197–203, 2008.

[18] Dan Ciresan and Cosmin Cernazanu. Linguistic modelling for automatic speech recognition in romanian language. *Proc. of the 7th International Conference on Development and Application Systems*, pages 490–497, 2004.

[19] Dan Ciresan and Dan Pescaru. Using character moment based invariant features to improve off-line handwriting recognition. *Proc. of the 8th International Conference on Development and Application Systems*, pages 471–476, 2006.

[20] Dan Ciresan and Dan Pescaru. Off-line recognition of handwritten numeral strings composed from two-digits partially overlapped using convolutional neural networks. *Proc. of IEEE 4th International Conference on Intelligent Computer Communication and Processing*, pages 53–60, 2008.

[21] L.P. Cordella and M. Vento. Symbol recognition in documents: a collection of techniques? *International Journal on Document Analysis and Recognition*, 3(2):73–88, December 2000.

[22] CEDAR DATABASE. http://www.cedar.buffalo.edu/databases/cdrom1/. *USPS Office of Advanced Technology Database of Handwritten Cities, States, ZIP Codes, Digits, and Alphabetic Characters.*

[23] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15:11–15, January 1972.

[24] G. Dzuba, A. Filatov, and A. Volgunin. Handwritten zip code recognition. *Proc. of the 4th International Conference on Document Analysis and Recognition (ICDAR'97)*, 2:766–770, August 1997.

[25] Streaming SIMD Extensions. http://en.wikipedia.org/wiki/streaming_simd_ extensions. *Wikipedia.*

[26] Rob Farber. Cuda, supercomputing for the masses. *http://www.ddj.com /architect/207200659.*

[27] Jurgen Franke. Expanding the performance of polynomial classifiers by iterative learning. *Advances in Handwriting Recognition*, 34:378–386, 1999.

[28] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, April 1980.

BUPT

[29] T. M. Ha, M. Zimmermann, and H. Bunke. Off-line handwritten numeral string recognition by combining segmentation-based and segmentation-free methods. *Journal of Pattern Recognition*, 31(3):257–272, 1998.

[30] T.M. HA, J. Zimmermann, and H. Bunke. Off-line handwritten numeral string recognition by combining segmentation-based and segmentattion-free methods. *Pattern Recognition*, 31(3):257–272, 1998.

[31] K. Kim, Y. Chung, J. Kim, and C.Y. Suen. Recognition of unconstrained handwritten numeral strings using decision value generator. *Proc. Sixth Int'l Conf. Document Analysis and Recognition*, pages 14–17, 2001.

[32] Kye Kyung Kim, Yun Koo Chung, and C.Y. Suen. Post-processing scheme for improving recognition performance of touching handwritten numeral strings. *Proc. of the 16th International Conference on Pattern Recognition*, 3:327–330, November 2002.

[33] Kye Kyung Kim, Yun Koo Chung, and C.Y. Suen. Recognition of unconstrained handwritten numeral strings by composite segmentation method. *Proc. of the 15th International Conference on Pattern Recognition*, 2:594–597, September 2002.

[34] G. Koch, Heutte L., and Paquet T. Automatic extraction of numerical sequences in handwritten incoming mail documents. *Pattern Recognition Letters*, 26(8):1118–1127, June 2005.

[35] F. Bortolozzi L. S. Oliveira, R. Sabourin and C. Y. Suen. Impacts of verification on a numeral string recognition system. *Pattern Recognition Letters*, 27(7):1023–1031, April 2003.

[36] F. Lauer, C.Y. Suen, and G. Bloch. A trainable feature extractor for handwritten digit recognition. *Journal of Pattern Recognition*, 40(6):1816–1824, June 2007.

[37] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.

[38] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[39] Y. LeCun, O. Matan, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, and H. S. Baird. Handwritten zip code recognition with multilayer networks. In IAPR, editor, *Proc. of the International Conference on Pattern Recognition*, volume II, pages 35–40, Atlantic City, 1990. IEEE. invited paper.

[40] Yann LeCun. The mnist database of handwritten digits http://yann.lecun.com/exdb/mnist/.

[41] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *The Quarterly of Applied Mathematics*, (2):164–168, 1944.

[42] Z. C. Li and C. Y. Suen. Distinctiveness and similarities of handwritten numerals. *Advances in Handwriting Recognition*, 34:387–396, 1999.

BUPT

[43] Cheng-Lin Liu and K. Marukawa. Handwritten numeral string recognition: character-level vs string-level classifier training. *Proc. of the 17th International Conference on Pattern Recognition*, 1:405–408, August 2004.

[44] Cheng-Lin Liu, Hiroshi Sako, and Hiromichi Fujisawa. Effects of classifier structures and training regimes on integrated segmentation and recognition of handwritten numeral string. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 26(11):1395–1407, November 2004.

[45] L.C. Liu, K. Nakashima, H. Sako, and H. Fujisawa. Integrated segmentation and recognition of handwritten numerals: Comparison of classification algorithms. *Proc. of the International Workshop on Frontiers in Handwritten Recognition (IWFHR'02)*, 8:303–308, 2002.

[46] George F. Luger. *Artificial Intelligence - Structures and strategies for complex problem solving.* Addison-Wesley, fifth edition, 2005.

[47] W. Ohyama M. Shi, T. Wakabayashi and F. Kimura. Mirror image learning for handwritten numeral recognition. *Proc. of the Second International Workshop on Machine Learning and Data Mining in Pattern Recognition*, LNCS 2123:239–248, 2001.

[48] Ranzato Marc'Aurelio, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In J. Platt et al., editor, *Advances in Neural Information Processing Systems (NIPS 2006).* MIT Press, 2006.

[49] O. Matan, J. Bromley, C. Burges, J. Denker, L. Jackel, Y. LeCun, E Pednault, W. Satterfield, C Stenard, and T. Thompson. Reading handwritten digits: A zip code recognition system. *IEEE Computer*, 25(7):59–63, July 1992.

[50] Ofer Matan, Christopher J. C. Burges, Yann LeCun, and John S. Denker. Multi-digit recognition using a space displacement neural network. In J. M. Moody, S. J. Hanson, and R. P. Lippman, editors, *Neural Information Processing Systems*, volume 4. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[51] Michael Negnevitsky. *Artificial Intelligence - A Guide to Intelligent Systems.* Addison-Wesley, second edition, 2005.

[52] Intel Software Network. Using streaming simd extensions 3 in algorithms with complex arithmetic. *http://cache-www.intel.com/cd/00/00/06/67 /66715_66715.pdf.*

[53] Daisuke Nishiwaki and Keiji Yamada. A new numeral string recognition method using character touching type verification. *Advances in Handwriting Recognition*, 34:416–425, 1999.

[54] NVIDIA. Cuda compute unified device architecture - programming guide. *www.nvidia.com.*

[55] NVIDIA. Cuda home. *http://www.nvidia.com/object/cuda_home.html#.*

[56] NVIDIA. Cuda overview. *http://www.isi.edu/~ddavis/GPU/Course/Slides/Mono Chrome/02 - Introduction to CUDA.pdf.*

BUPT

[57] L.S. Oliveira and R. Sabourin. Support vector machine for handwritten numerical string recognition. *9th International Workshop on Frontiers in Handwriting Recognition (IWFHR-9)*, pages 39–44, October 2004.

[58] L.S. Oliveira, R. Sabourin, F. Bortolozzi, and Suen C.Y. Automatic recognition of handwritten numerical strings: A recognition and verification strategy. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 24(11):1438–1454, November 2002.

[59] S. Ouchtati, M. Bedda, and A. Lachouri. Segmentation and recognition of handwritten numeric chains. *Journal of Computer Science*, 3(4):242–248, 2007.

[60] Marc'Aurelio Ranzato, Fu-Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'07)*. IEEE Press, 2007.

[61] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach*. Prentice Hall, second edition, 2003.

[62] Stuart C. Shapiro, editor. *Encyclopedia of artificial intelligence*, volume 1. John Wiley & Sons, 1990.

[63] Z. Shi, N. Srihari, C. Y. Shin, and A. V. Ramanaprasad. A system for segmentation and recognition of totally unconstrained handwritten numeral strings. *Proc. of the 4th International Conference on Document Analysis and Recognition (ICDAR'97)*, 2:445–458, August 1997.

[64] Patrice Y. Simard, Dave Steinkraus, and John Platt. Best practices for convolutional neural networks applied to visual document analysis. *Proc. of International Conference on Document Analysis and Recognition*, pages 958–962, 2003.

[65] ASPLOS 2008 Tutorial. Gpu architecture and applications. *http://www.gpgpu.org/asplos2008/ASPLOS08-4-GPU-architecture.pdf*.

[66] Stanford University. Brookgpu. *http://graphics.stanford.edu/projects/brookgpu/*.

[67] Xian Wang, Venu Govindaraju, and Sargur Srihari. Holistic recognition of touching digits. *Advances in Handwriting Recognition*, 34:359–367, 1999.

[68] R. Allen Wilkinson, Michael D. Garris, and Jon Geist. Machine-Assisted Human Classification of Segmented Characters for OCR Testing and Training. volume 1906. SPIE, San Jose, 1993.

[69] J.F. Wang Y.K.Chen. Segmentation of single- or multiple-touching handwritten numeral string using background and foreground analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1304–1317, November 2000.

[70] Jie Zhou, Qiang Gan, Adam Krzyzak, and Ghing Y. Suen. Quantum neural network in recognition of handwritten numerals. *Advances in Handwriting Recognition*, 34:368–377, 1999.

BUPT

Titluri recent publicate în colecţia „TEZE DE DOCTORAT" seria 10: Ştiinţa Calculatoarelor

1. **Rodica Ţirtea** – *Contribuţii la îmbunătăţirea dependabilităţii şi securităţii informaţiei, ISBN 978-973-625-422-2*, (2007);

2. **Ionel Muscalagiu** – *Contribuţii la implementarea, evaluarea şi îmbunătăţirea performanţelor tehnicilor de căutare asincrone în cadrul programării bazate pe constrângeri distribuite, ISBN 978-973-625-592-2,* (2007);

3. **Daniel Cioi** – *Contribuţii la utilizarea realităţii virtuale în proiectarea asistată de calculator, ISBN 978-973-625-613-4,* (2008);

4. **Sorin Babii** – *Cercetări privind creşterea performanţelor reţelelor neuronale într-un mediu de calcul distribuit, ISBN 978-973-625-559-5,* (2008);

5. **Norbert Neidenbach** - *Das Service-Management eines IT-Outsourcing-Projektes durch ITIL-Best-Practices, IT-Outsourcing kostenoptimiert planen und steuern, ISBN 978-973-625-660-8*, (2008);

6. **Edwin Hans Wolf** - *Das Geschäftsmodell (Business model) MDS (Managed Desktop Support) im IT-Outsourcing, Leistungserbringung im Rahmen des MDS-Geschäftsmodells, ISBN 978-973-625-661-5,* (2008);

7. **Adrian Zafiu** – *Minimizarea sistemelor decizionale multivalente deterministe şi nedeterministe, ISBN 978-973-625-678-3,* (2008);

8. **Daniel Iercan** – *Contributions to the Development of Real-Time Programming Techniques and Technologies, ISBN 978-973-625-719-3,* (2008);

9. **Laurenţia Timar** – *Contribuţii referitoare la configurarea optimală prin prisma performanţă-fiabilitate a unor reţele de dispozitive de achiziţia datelor cu aplicabilitate la excavatoarele cu cupe, ISBN 978-973-625-775-9,* (2008).

EDITURA POLITEHNICA

BUPT