Neural Network using OpenMp

In vederea elaborarii teme s-a ales ca punct de plecare proiectul profesorului Sven Koenig propus pentru dezvoltare in cadrul departamentului de Computer Science de la University of Southern California in anul 2010. Proiectul se numeste “Gesture Recognition with Neural Networks” si are aplicabilitate cu preponderenta in domeniul jocurilor video. Proiectul sta la baza unui nou trend in materie de tehnologi folosite in jocurile video si a ajutat la dezvoltarea unor sisteme considerate revolutionare (WII / Xbox Kinect / VR).

Paralelizare se va executa prin intermediul OpenMP

OpenMP este o interfatǎ de programare a aplicatiilor (API – Application Program Interface) care poate fi utilizatǎ în paralelismul multifir cu memorie partajatǎ (în indiviziune). Contine trei componente API primare: directive de compilare, rutine de bibliotecǎ la executie, variabile de mediu (environment). Portabilitate: Aceastǎ API este realizatǎ în C/C++ si Fortran. S-a implementat pe platforme variate, inclusiv pe cele mai multe platforme Unix si Windows NT. Standardizare: Este definitǎ si aprobatǎ în comun de un grup de producǎtori majori de hardware si software. Este de asteptat a deveni în viitor un standard ANSI (American National Standards Institute). Ce înseamnǎ OpenMP? Înseamnǎ Open specifications for Multi Processing prin lucrǎri în colaborare care întrunesc interese din industria de software si de hardware, guvernamentale si academice.

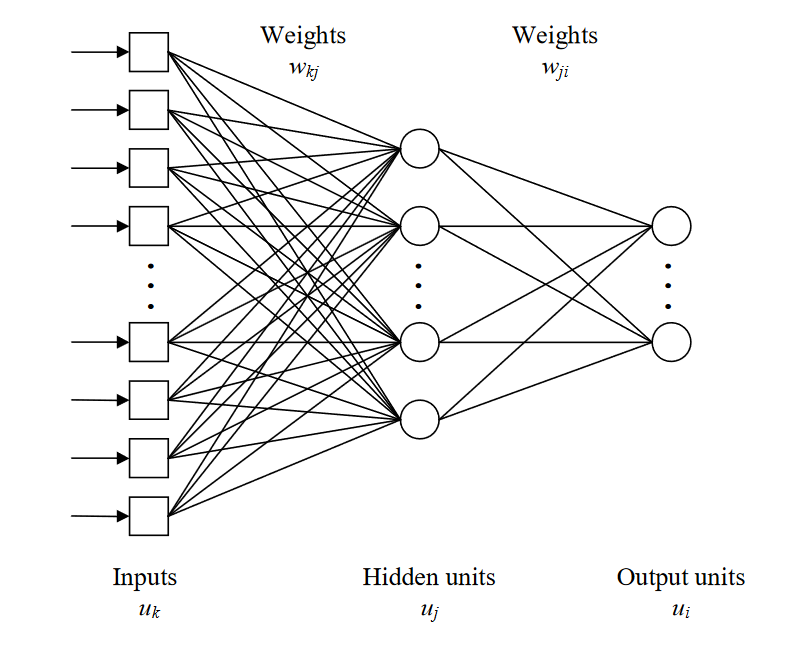
Pentru inceput, trebuie specificat ca in cadrul Universitatii din California de Sud exista un program de licenta numit “Games” si un program de master, “Games Development” cu scop primordial in pregatirea studentilor astfel incat la terminarea ciclului 2 de studii sa isi poata puna amprenta pe industria jocurilor.

Inca din anul 2010, odata cu aparitia numeroaselor librarii grafice ce au facut posibila implementarea graficii de mare calitate, jocurile nu s-au mai putut distanta unul fata de celalate pe acest criteriu iar pe primul loc a trecut inteligenta artificiala. Noul trend este reprezentat de folosirea algoritmilor de inteligenta artificiala pentru a creea elemente cat mai complexe de trajectory-palnning (algoritmi de cautare, de cele mai multe ori variante ale algoritmului A\*).

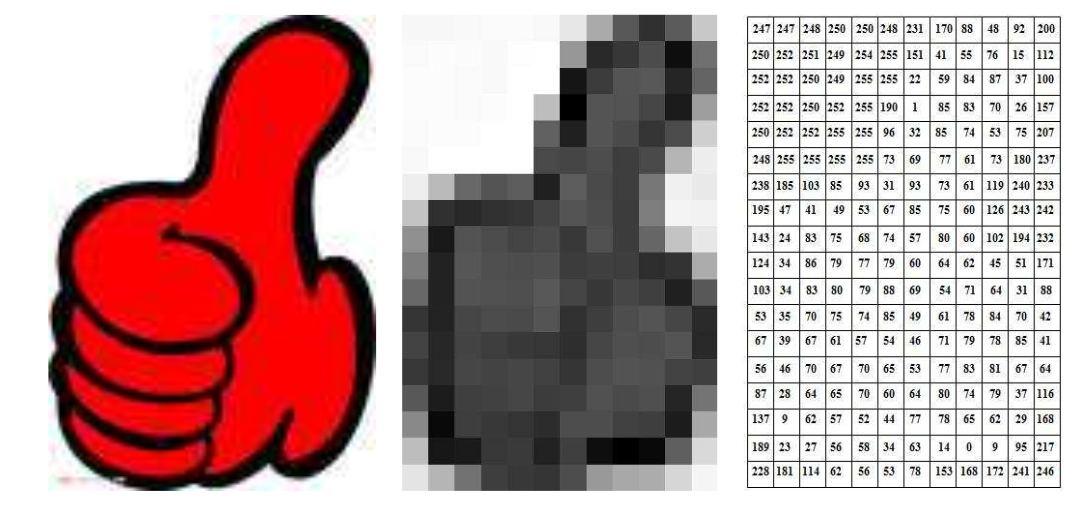
Un joc inovator in acest domeniu a fost “Black and White”, lansat in 2001, este considerat pionierul intergrarii Machine Learning in mediul jocurilor. S-a folosit o combinatie de arbori de decizie si retele neuronale pentru a conferi utilizatorului o experienta ce va rezona cu gusurile si caracterul sau.

Retelele neuronale reprezinta una din cele mai importante tehnici de machine learning. In proiectul propus, s-a adaptat o retea neuronala pusa la dispozitie de Tom Mitchel in cartea intitulata “Machine learning” si adaptat de Sven Koenig in cadrul proiectului sau pentru a se plia pe cerintele interpretarii gesturilor in vederea folosirii in industria jocurilor video. In urma analizarii codului s-a ajuns la concluzia ca s-ar putea imbunatatii prin paralelizarea. Se observa ca a simpla paralelizare a regiunii de antrenare a retelei neuronal va conduce la o imbunatatire considerabila a timpului in care dispozitivul antreneaza reteaua. Un timp micsorat de antrenare duce la o performanta mai buna si posibilitatea de a folosi reteaua neuronala pentru a lua decizii cat mai bune intr-un timp cat mai scurt(cu cat antrenarea este mai complexa cu atat testarea va avea rezultate mai bune).

Informatii utile pentru viiitori dezvoltatori: Un alt element in care s-a putut observa necesitatea existentei paralelizarii este detectia in timp real de gesturi. Trebuie mentionat ca nu s-a implementat aceasta functionalitate in cadrul proiectului dar prin paralelizarea efectuata asupra regiuni de test s-a ajuns la concluzia ca s-a redus timpul necesar incardraii unei poze in gama de poze ce reprezinta gestul cautat. Paralelizarea este o conditie Sine qua non pentru a fi posibila integrarea aplicatiei in cadrul dispozitivelor ce vor procesa imagini si oferi utilizatorilor o experienta bazata pe gesturile acestora.

 In elaborarea proiectului s-a folosit o retea neuronala cu un singur layer ascuns si un layer de output asa cum se poate observa din figura urmatoare:

Imaginile au fost subimpartite si apoi reprezentate in functie de intensitatea lor, intr-o matrice, cu valori cuprinse de la 0(negru) la 255(alb). In figura urmatoare se poate observa procesul de impartire (format .pgm).

 In folderul “gestures” se gasesc foldere ce contin gesturi in format (down/hold/stop/up) ce provin din mai multe surse( A,B,C, …, M) in format .pgm. In acest proiect se va face

Fiecare iesire a retelei corespunde unei combinatii de valori ale datelor de intrare. Se va folosi un singur neuron in layer-ul de output si se va raspunde cu “da” daca valoarea din neuron este mai mare decat 0.5 respectiv “nu” in caz contrar.

**Rularea programului gesturetrain**

* -n < network file> - aceasta optiune incarca o retea neuronala existenta sau creeaza una cu numele dat.Reteaua neuronala este salvata in fisierul respectiv la finalul datelor de antrenament
* -e < number of epoch> - Aceasta optiune specfica numarul de “epoci” pentru antrenare.Optiunea implicita este 100
* -s < seed> Aceasta optiune seteaza baza pentru generatorul de numere aleatoare.Baza implicita este 102194.Aceasta optiune permite flexibilitate in alegerea generarii numerelor aleatoare.
* -S < number of epochs between saves > Aceasta optiune specifica numarul de “epoci” intre doua salvari ale retelei neuronale.Valoarea implicita este 100, ceea ce inseamna ca reteaua neuronala este salvata doar la finalul antrenamentului.
* -t <train set list> Aceasta optiune specifica numele fisierului care contine o lista de nume de imagini foloste ca antrenament.
* -1 < test set 1 list> - aceasta optiune specifica numele unui fisier care contine o lista de nume de imagini (calea catre imagini), o cale pe fiecare linie, folosite ca exemple de validare ( cross validation set ) .
* -2 <test set 2 list> aceasta optiune specifica numele unui fisier care contine o lista de nume de imagini (calea catre imagini), o cale pe fiecare linie, folosite ca exemple de test.

Rezultatul lui **gesturetrain**:

Programul gesturetrain prima data citeste toate datele si afiseaza cateva linii despre aceste operatii.Apoi incepe antrenarea si raporteaza performantele antrenarii, validarii si testarii pe cate o linie diferita penru fiecare epoca.

<epoch> <delta> <trainperf> <trainerr> <tes1perf> <test1err> <test2perf> <test2err>

Aceste valori inseamna:

epoch – numarul “epoci” ( iteratiei )

delta - suma tuturor valorilor lui al layerelor ascuns si al celui final peste toate exemplele

trainperf – procentajul exemplelor de antrenare care au fost clasificate corect

trainerr – media erorilor pentru toate exemplele de antrenare

test1perf - procentajul exemplelor de validare care au fost clasificate corect

test1err – media erorilor pentru toate exemplele de validare

test2perf - procentajul exemplelor de test care au fost clasificate corect

test2err – media erorilor pentru toate exemplele de test

Detalii de implementare:

*gesturetrain.c* reprezinta top-level-ul tuturor modulelor in implementarea functionalitatii de recunoastere a gesturilor.

*backprop.c* si *backprop.h* sunt modulele care permit initializarea antrenarea si folosirea reteleleor neuronale

*imagenet.c* reprezinta modulul de interfata care genereaza datele de intrare pentru reteaua neuronala si seteaza output-ul dorit pentru antrenare.

*hidtopgm.c* modulul de vizualizare a neuronilor din layerul intern.

*pgmimage.c si pgmimage.h* sunt modulele care permit citirea si scrierea fisierelor de tip pgm.

In cele ce urmeaza se este descris codul aferent retelei neuronale in mod detaliat.

***void bpnn\_initialize(int seed) –*** Initializeaza reteaua neuronala .Trebuie apelata inaintea oricarei alte functii.In momentul acesta functia initializeaza doar generatorul de numere aleatoare.

***BPNN\* bpnn\_create(int n\_in,int n\_hidden,int n\_out) –*** Aceasta rutina creaza o noua structura de tip BPNN cu valorile date.Toate *weight-urile* sunt initializate cu valori aleatoare in intervalul -1.0 - 1.0.Subrutina intoarce un pointer catre structura creata sau NULL in caz de eroare

***void bpnn\_free(BPNN\* net)*** - Elibereaza toata memoria asociata cu structura BPNN.

***void bpnn\_train(BPNN\* net,double learning\_rate,double momentum,double \*erro double\* errh) –*** Aceatsa metoda ruleaza algoritmul back-propagation pentru strucura oferita ca intrare.Aceasta functie presupune ca datele de iesire si de intrare au fost setate in prealabil.***learning\_rate*** si ***momentum*** trebuie sa fie valori din intervalul 0.0 si 1.0.***errh*** si ***erro*** sunt pointeri catre double care sunt setate cu suma tuturor valorilor al neuronilor ascunsi si al celor din layerul final peste toate exemplele de antrenament.

***void bpnn\_feedforward(BPNN\* net)*** - Aceasta functie determina output-ul actual pentru valoarea de input din cadrul retelei neuronale.

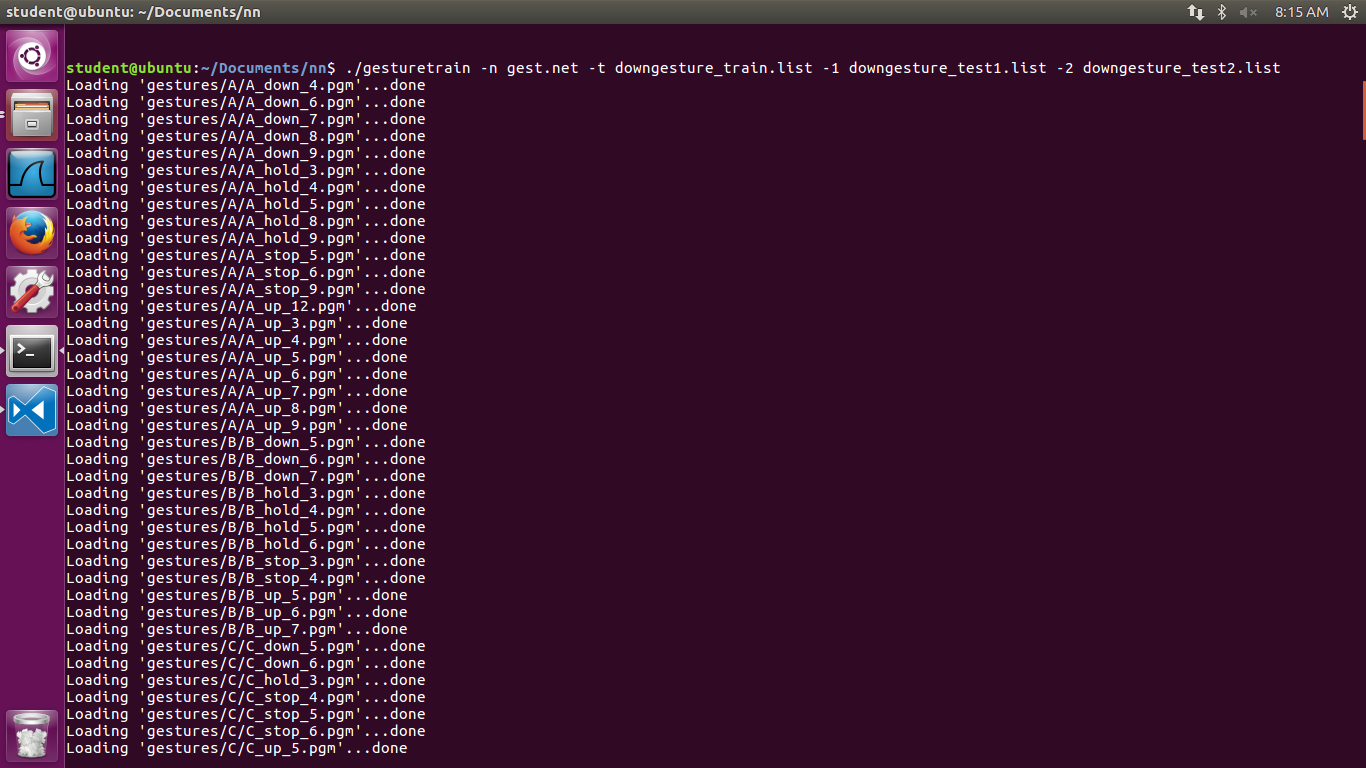
***BPNN\* bpnn\_read(char\* filename) –*** Aceasta functie aloca memorie pentru BPNN, o initializeaza cu weight-urile stocate in fisier si returneaza un pointer catre noua structura creata sau NULL in caz de eroare.

***void bpnn\_save(BPNN\* net,char\* filename) –*** Aceasta rutina salveaza reteaua data in fisierul oferit ca parametru.

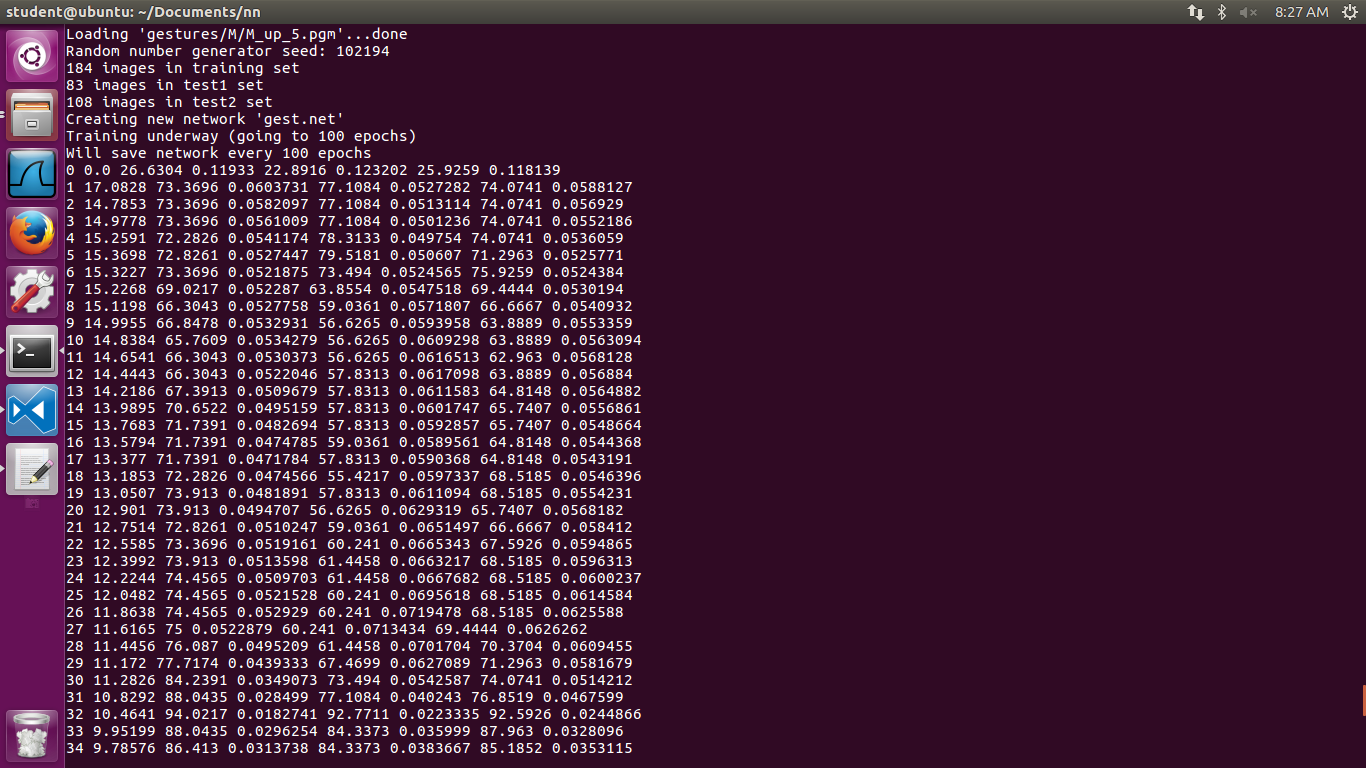
Un exemplu de rulare a programului:

-se foloseste comanda **./gesturetrain -n gest.net -t downgesture\_train.list -1 downgesture\_test1.list -2 downgesture\_test2.list**

Unde: gest.net reprezinta reteaua neuronala si daca aceasta nu exista se va creea una noua cu acest nume, downgesture\_train.list . In acest exemplu am ales ce de-a doua situatie.

Primul pas esete cel de incarcare al imaginilor. 

Pasul 2 il reprezinta afisarea numarului imaginilor ce vor face parte din antrenarea respectiv testarea retelei apoi se vor afisa datele pentru fiecare iteratie (epoca).

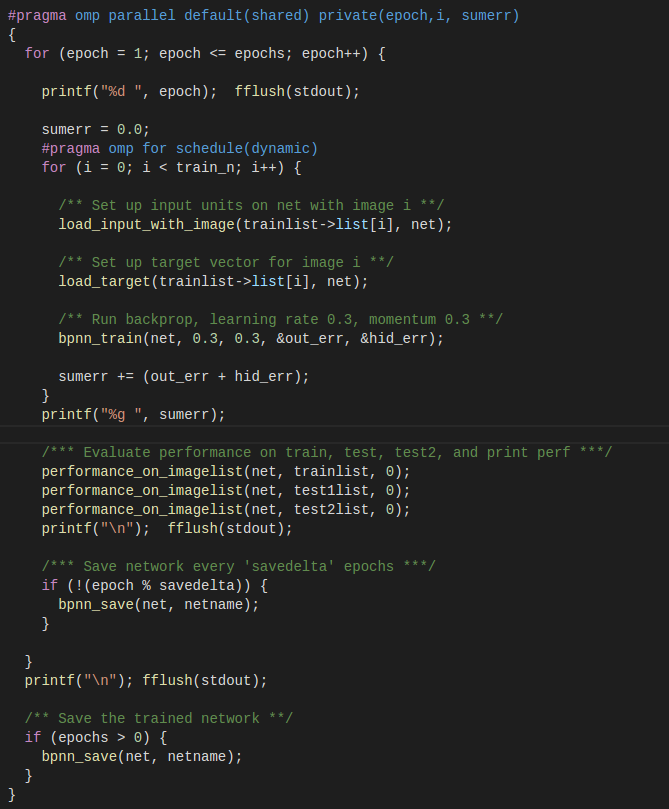


Se observa ca cu cat creste numarul iteratiilor, procentajul exemplelor de antrenare ce au fost clasificate corect creste iar media erorilor pentru toate exemplele de test scade.

Prima justificare a paralelizarii se gasest la nivelul incarcarii datelor din fisiere. S-a folosit directiva sections deoarece fisierele sunt de dimensiuni relativ egale iar fiecare incarcare va fi atribuita cate unui thread in parte.



Urmatoarea sectiune a proiectului paralelizata va fi bucata de cod ce sa ocupa cu antrenarea si testarea retelei neuronale. Deoarece paralelizarea codului retelei ar duce la alterarea rezultatelor s-a ales metoda impartirii iteratiilor (epoci) pe numarul maxim de thread-uri.



Pentru a testa diferenta vitezei de executie a programului in cele 2 cazuri s-a folosit functia omp\_get\_wtime() din omp.h. Pentru a avea acces la aceasta functie a trebuit inclusa in proiect prin directiva #include <omp.h> si adaugat la compilare –fopenmp. Pentru folosirea exclusiva de directive #pragma pentru paralelizare nu este nevoie de includerea <omp.h>.

start = omp\_get\_wtime();

end = omp\_get\_wtime();

1)Timpul de executare al programului folosind codul paralelizat:



2)Timpul de executare al programului folosind codul serial:

C:\Users\Sorin\AppData\Local\Microsoft\Windows\INetCache\Content.Word\dovada2.jpg

In concluzie, Paralelizarea retelei neuronale nu are un efect foarte mare asupra unei retele cu putine iteratii(epoci, 100) dar este vitala in cazul implementarii acesteia intr-un produs ce se cere a recunoaste gesturile in mod automat si in timp util.

Codul adaptat se poate gasi pe pagina profesorului Tom Mitchell <http://www.cs.cmu.edu/~tom/>

<http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-8/faceimages/code/>

modificat de Sven Koenig <http://idm-lab.org/project-m/project3.html>