Projet L3 FISA Système

"Parallélisation et optimisation des calculs matricielles"

Motivations :

Les applications modernes demandent de plus en plus de ressources de calcul performantes afin du supporter leur charge de traitement. Par exemple, les jeux vidéos ou les application basées sur l'intelligences artificielle qui nécessitent des plateformes spécifiques tels que les processeurs graphiques (*GPU* pour *Graphics Processing Unit)* afin d'accélérer l'exécution des opérations matricielles. Néanmoins, les GPU souffrent de deux problèmes principaux qui sont leur coût et leur consommation énergétique. Les CPU restent beaucoup moins couteux et économe en énergie mais beaucoup moins performants due à leur degré de parallélisme limité.

Ce projet a pour objectif d'étudier et d'analyser les degré de parallélisme lors de l'exécution des applications de calcul matriciel. Des concepts de bases sur les systèmes d'exploitation vont être abordés tel que la programmation parallèle et concurrente, la gestion des ressources critiques, ou l'étude de l'accélération *(speedup)*.

L'implémentation des blocs de base d'un réseau de neurones de convolution représentera un cas d'usage concret des solutions implémentés.

Modalités d'évaluation :

L'évaluation de ce projet est basé sur trois éléments :

1. Un rapport détaillant les différents aspects abordés, les problèmes rencontrés, ainsi que les solutions proposées.
2. Une courte présentation résumant le rapport en plus d'une démonstration.
3. Le code source des briques logicielles développées, bien commenté.

Travail demandé

Le travail demandé dans ce projet est d'implémenter une bibliothèque C (*fisa3_cnn.h* et *fisa_cnn.c*) qui facilite les opérations élémentaires des réseaux de neurones à convolution. Nous nous focalisons sur 3 opérations : 1) la convolution, 2) le pooling, et 3) la fonction d'activation.

La convolution (simplifié) :

L'opération de convolution (noté *) consiste à effectuer en produit élément par élément suivi d'une somme entre deux vecteurs (convolution 1D), ou entre une matrice (appelée Feature Map) et une autre (appelée filtre ou kernel). La figure suivante montre un exemple de calcul d'une convolution 1D avec une feature map (i_1, …,i_6) et un kernel (w_1, w_2, w_3) et qui donne une autre feature map (c_1,…,c_6) (source: https://www.analyticsvidhya.com).

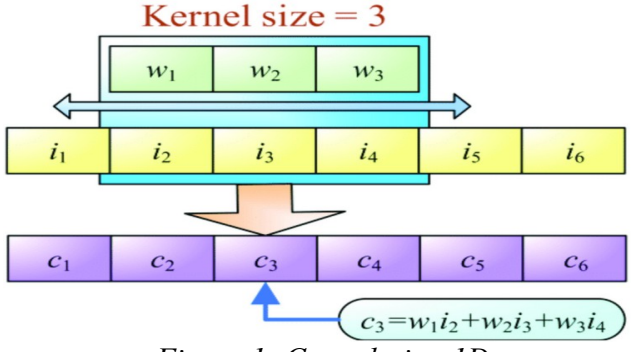

Figure 1: Convolution 1D

Le principe de la convolution 2D est similaire et montré dans la figure ci-dessous.

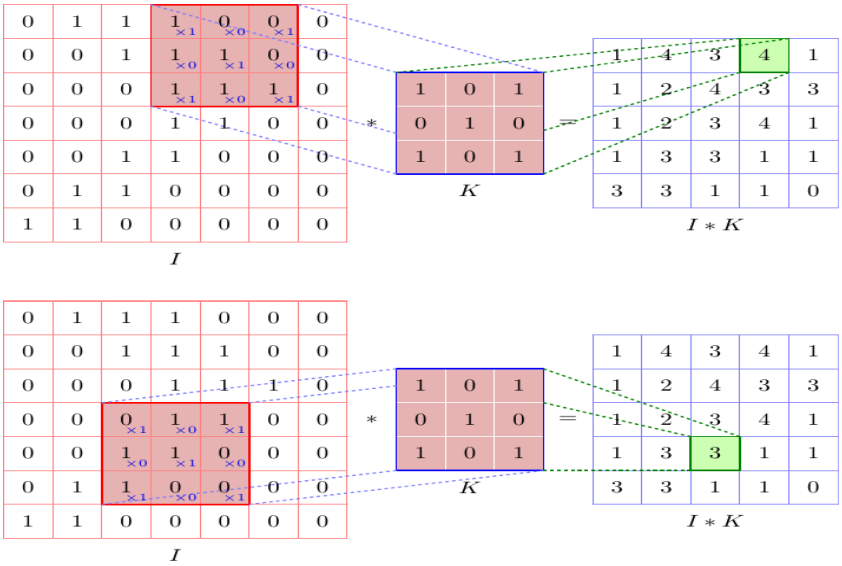

Figure 2: Convolution 2D

La taille de la feature map en sortie dépend de 4 paramètres :

1. La taille de la feature map en entré (nombre de lignes et de colonnes)
2. La taille du Kernel (nombre de lignes et de colonnes)
3. Le pas de glissement du Kernel sur le la feautre map en entrée (appelé stride)
4. Le nombre de lignes (=colonnes) contenant que des zéros rajoutées aux bords de la feature map en entrée (appelé padding. Voir annimation)

Pour plus de détails, voici un cours très complet du MIT [1].

Le travail demandé dans cette partie est de proposer une implémentation des deux convolutions 1D et 2D avec les prototypes suivants :

```
int conv1D(*in_feature_map,      // Le vecteur en entré
*kernel,                         // Le filtre
unsigned short stride,           // La valeur du pas de glissement
unsigned short padding,          // La valeur du padding (lignes et colonnes des zéros)
*in_feature_ma                   // Le vecteur résultant )

int conv2D(*in_feature_map,      // La matrice en entré
*kernel,                         // Le filtre
unsigned short stride,           // La valeur du pas de glissement
unsigned short padding,          // La valeur du padding (lignes et colonnes des zéros)
*in_feature_ma                   // Le vecteur résultant)
```

Le Max/Min/Average pooling :

L'opération du pooling est une opération fréquement utilisé dans les réseaux de neurones de convolution. Elle est souvent placée entre deux opérations de convolution afin de réduire la taille de la sortie d'une convolution avant de passer à la convolution suivante. Le pooling consiste à calculer la valeur maximale (dans le cas du max pooling), minimale (dans le cas du min pooling), ou moyenne (dans le cas du average pooling) de la feautre map. Le calcul se fait d'une manière semblable à la convolution mais en remplaçant le produit et la somme par le calcul du min, max, ou average. La figure suivante montre une opération d'un max pooling avec un filtre 2x2, un stride de 2 et sans padding.

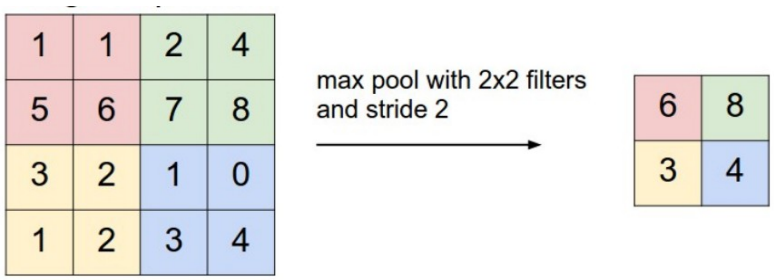

Figure 3: Max pooling 2x2 avec stride 2, et sans padding

Le travail demandé dans cette partie est de proposer une implémentation des trois types de pooling avec les prototypes suivants :

```
int maxPool(*in_feature_map,     // input feature map
unsigned short kernel_size,      // La taille filtre
unsigned short stride,           // La valeur du pas de glissement
unsigned short padding,          // La valeur du padding (lignes et colonnes des zéros)
*in_feature_map                  // output feature map)

int minPool(*in_feature_map,     // input feature map
*kernel,                         // Le filtre
unsigned short stride,           // La valeur du pas de glissement
unsigned short padding,          // La valeur du padding (lignes et colonnes des zéros)
*in_feature_map                  // output feature map)

int avgPool(*in_feature_map,     // input feature map
*kernel,                         // Le filtre
unsigned short stride,           // La valeur du pas de glissement
unsigned short padding,          // La valeur du padding (lignes et colonnes des zéros)
*in_feature_map                  // output feature map)
```

Les fonctions d'activation :

Les fonctions d'activation sont des opérations très utilisées dans les réseaux de neurones. Ces fonctions sont utilisées pour réduire les dispersion des valeurs obtenues après les opérations de convolution. Les fonctions d'activation ne changent pas la taille des feature map. Elle appliquent une fonction $y = f(x_{ij})$ où y est le résultat de la fonction d'activation f et x_{ij} est un élément de la feature map. Il existe 3 fonctions d'activation les plus fréquemment utilisées dans les réseaux de neurones : Sigmoid, Tanh, et ReLU.

Pour la fonction sigmoid $f(x)=\frac{1}{1+e^{-x}}$

Pour la fonction tanh $f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

Pour la fonction ReLU $f(x)=\begin{cases} x\ si\ x>0 \\ 0\ sinon \end{cases}$

Lorsqu'une fonction activation est appliquée à la sortie d'une convolution, la feautre map est parcourue et la fonction d'activation est appliquée à chaque élément ij de la feautre map.

Le travail demandé dans cette partie est d'implémenter les 3 fonctions d'activation mentionnées ci-dessus selon les prototypes suivants :

```
int sigmoid(*in_feature_map,     // input feature map
*in_feature_map                  // output feature map)

int tanh(*in_feature_map,        // input feature map
*in_feature_map                  // output feature map)

int relu(*in_feature_map,        // input feature map
*in_feature_map                  // output feature map)
```

Implémentation d'une partie de CNN séquentielle :

Après avoir implémenté et testé chaque opération, nous souhaitons implémenter dans un fichier *main* une partie d'un réseau de convolution comme suit :

[input] → [Conv2D] → [ReLU] → [MaxPool] → [output]

[input] → [Conv2D] → [Sigmoid] → [AvgPool] → [output]

- L'input et le kernel de convolution sont lus à partir de fichiers (un fichier chacun) donnés en paramètre lors de l'exécution du binaire. L'output est ecrit dans un fichier également.
- Les tailles de l'input sont variées comme suit : 32x32, 64x64, et 128x128.
- Les tailles du kernel de convolution sont variées comme suit : 3x3, 5x5, 7x7.
- La valeur du stride est variée comme suit : 1, 2, 3; sans et avec un padding de 1.
- Les valeurs des éléments de l'input et du kernel (fichiers en paramètres) sont générées aléatoirement.
- Le temps d'exécution de votre programme doit être mesuré (voir la fonction gettimeofday [2]) pour chaque configuration (taille d'input, taille du kernel, etc.). Les temps en fonction de chaque configuration doivent être presentés sous forme d'histogramme.

Parallélisation à l'aide des threads :

1. Reprenez votre programme développé dans la précédente partie et proposez une version multithreddée lors de l'exécution de la convolution et du pooling en faisant varier le nombre de threads utilisé : 2, 4, 6, et 8.
2. Mesurez le temps d'exécution de votre programme en fonction du nombre de threads utilisé. Les temps en fonction de chaque configuration et nombre de threads doivent être présentés sous forme d'histogramme.

NOTE : l'output d'une opération et forcement l'input de l'opération suivante. Il y aura donc un accès concurrent (section critiques) sur les résultats intermédiaires.

Références

[1] Convolutional Neural Networks (CNNs / ConvNets): *https://cs231n.github.io/convolutional-networks/*

[2] GETTIMEOFDAY: *manpagesfr.free.fr/man/man2/gettimeofday.2.html*