

Hardware for Signal Processing

Implémentation d'un CNN - LeNet-5 sur GPU

Abdelhamied OMAR YOUSIF

Partie 1 – Prise en main de CUDA: multiplication de matrices

Performance comparée des résultats CPU et GPU :

pour N=1000 :

```
jaheergoulam@jaheergoulam:~$ ssh -p 7718 jaheergoulam@d261-routeur.ensea.fr -t 118x41
[jaheergoulam@d261-pc8:~$ nvcc part1.cu -o part1
[jaheergoulam@d261-pc8:~$ nvprof ./part1
Matrice M1 :

Matrice M2 :

Résultat addition CPU :
Temps addition CPU : 0.0062 secondes
==7437== NVPROF is profiling process 7437, command: ./part1

Résultat addition GPU :

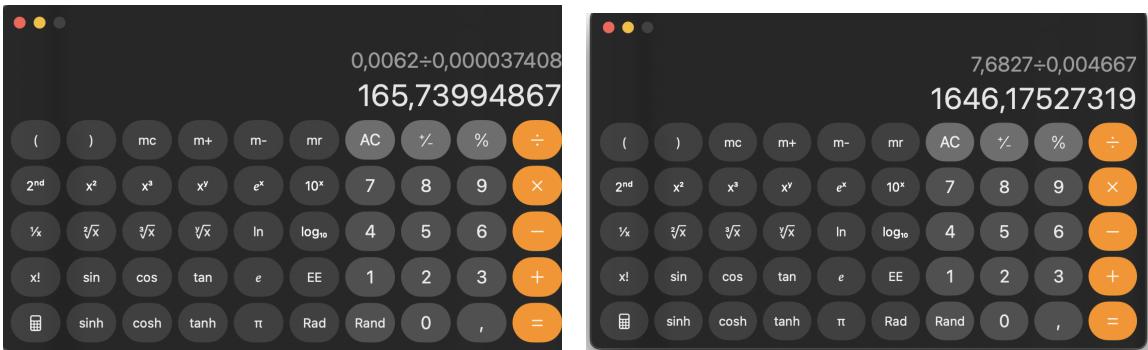
Résultat multiplication CPU :
Temps multiplication CPU : 7.6827 secondes

Résultat multiplication GPU :
==7437== Profiling application: ./part1
==7437== Profiling result:
      Type  Time(%)    Time    Calls      Avg      Min      Max  Name
GPU activities:  56.37%  4.6679ms     1  4.6679ms  4.6679ms  4.6679ms  cudaMatrixMult(float*, float*, float*, int)
                26.68%  2.2094ms     2  1.1047ms  490.08us  1.7193ms  [CUDA memcpy DtoH]
                16.50%  1.3668ms     2  683.39us  654.66us  712.13us  [CUDA memcpy HtoD]
                0.45%  37.408us     1  37.408us  37.408us  37.408us  cudaMatrixAdd(float*, float*, float*, int, int)
API calls:   93.75%  169.28ms     3  56.42ms  110.00us  169.05ms  cudaMalloc
            5.74%  10.362ms     4  2.5905ms  858.64us  5.5255ms  cudaMemcpy
            0.28%  505.80us     3  168.60us  162.59us  174.88us  cudaFree
            0.16%  280.84us    101  2.7800us  1.2570us  67.887us  cuDeviceGetAttribute
            0.05%  94.776us     2  47.388us  41.137us  53.639us  cudaLaunchKernel
            0.01%  15.784us     1  15.784us  15.784us  15.784us  cuDeviceGetName
            0.00%  8.2410us     1  8.2410us  8.2410us  8.2410us  cuDeviceGetPCIBusID
            0.00%  5.3080us     3  1.7690us  1.3970us  2.4450us  cuDeviceGetCount
            0.00%  3.0030us     2  1.5010us  1.1170us  1.8860us  cuDeviceGet
            0.00%  1.8850us     1  1.8850us  1.8850us  1.8850us  cuDeviceTotalMem
            0.00%  1.4660us     1  1.4660us  1.4660us  1.4660us  cuDeviceGetUuid
jaheergoulam@d261-pc8:~$
```

Multiplication :

- **CPU** : la multiplication a pris 7,6827s sur le CPU.
- **GPU** : la multiplication sur le GPU a pris un total de 4,6679ms, ce qui démontre une efficacité nettement supérieure. La décomposition des activités GPU est la suivante :
 - `cudaMatrixMult(float*, float*, float*, int)` : 4,6679 ms (56,37 %)
 - `[CUDA memcpy DtoH]`: 2,2094 ms (26,68 %)
 - `[CUDA memcpy HtoD]`: 1,3668 ms (16,50 %)
 - `cudaMatrixAdd(float*, float*, float*, int, int)` : 37,408 microsecondes (0,45 %)

$$\text{Accélération} = \frac{\text{Temps d'exécution sans mécanisme d'accélération}}{\text{Temps d'exécution avec mécanisme d'accélération}}$$



Si on met blockDim = 1 (pour x et y),

Pour N = 10 000

blocDim = (1, 1)

```

jaheergoulam - jahegou10@d261-pc8: ~ -- ssh -p 7718 jahegou10@d261-routeur.ensea.fr — 118x41
    0.00% 3.6330us      2  1.8160us  1.4670us  2.1660us cuDeviceGet
    0.00% 2.0960us      1  2.0960us  2.0960us  2.0960us cuDeviceTotalMem
    0.00% 1.6770us      1  1.6770us  1.6770us  1.6770us cuDeviceGetUuid
[jah...@d261-pc8:~$ nvcc part1.cu -o part1
[jah...@d261-pc8:~$ nvprof ./part1
Matrice M1 :

Matrice M2 :
==9710== NVPROF is profiling process 9710, command: ./part1

Résultat addition GPU :

Résultat multiplication GPU :
==9710== Profiling application: ./part1
==9710== Profiling result:
      Type  Time(%)     Time      Calls      Avg       Min       Max   Name
GPU activities:  61.50% 286.84ms      2 143.42ms  59.432ms 227.41ms [CUDA memcpy DtoH]
                27.74% 129.39ms      2 64.696ms  64.642ms 64.751ms [CUDA memcpy HtoD]
                10.74% 50.087ms      1 50.087ms  50.087ms 50.087ms cudaMemcpyMatrixMult(float*, float*, float*, int
t)
                0.02% 87.361us      1 87.361us  87.361us 87.361us cudaMemcpyAdd(float*, float*, float*, int
, int)
      API calls:  71.07% 467.79ms      4 116.95ms  64.807ms 228.33ms cudaMemcpy
                28.28% 186.12ms      3 62.039ms  131.44us 185.83ms cudaMalloc
                0.58% 3.8227ms      3 1.2742ms  407.67us 2.0834ms cudaFree
                0.05% 331.89us     101 3.2860us  1.0480us 81.016us cuDeviceGetAttribute
                0.01% 78.293us      2 39.146us  32.337us 45.956us cudaLaunchKernel
                0.00% 17.321us      1 17.321us  17.321us 17.321us cuDeviceGetName
                0.00% 7.3330us      1 7.3330us  7.3330us 7.3330us cuDeviceGetPCIBusId
                0.00% 6.2860us      3 2.0950us  1.6670us 2.8630us cuDeviceGetCount
                0.00% 3.7720us      2 1.8860us  1.5370us 2.2350us cuDeviceGet
                0.00% 2.0950us      1 2.0950us  2.0950us 2.0950us cuDeviceTotalMem
                0.00% 1.6770us      1 1.6770us  1.6770us 1.6770us cuDeviceGetUuid
[jah...@d261-pc8:~$ nvcc part1.cu -o part1
[jah...@d261-pc8:~$ nvprof ./part1
Matrice M1 :

Matrice M2 :
==9812== NVPROF is profiling process 9812, command: ./part1

Résultat addition GPU :

```

blocDim = (64, 64)

```

jaheergoulam — jahegou10@d261-pc8: ~ — ssh -p 7718 jahegou10@d261-routeur.ensea.fr — 118x41
      28.28% 186.12ms    3  62.039ms 131.44us 185.83ms  cudaMalloc
      0.58% 3.8227ms     3  1.2742ms 407.67us 2.0834ms  cudaFree
      0.05% 331.89us    101 3.2860us 1.0480us 81.016us  cuDeviceGetAttribute
      0.01% 78.293us     2  39.146us 32.337us 45.956us  cuLaunchKernel
      0.00% 17.321us     1  17.321us 17.321us 17.321us  cuDeviceGetName
      0.00% 7.3330us     1  7.3330us 7.3330us 7.3330us  cuDeviceGetPCIBusId
      0.00% 6.2860us     3  2.0950us 1.6070us 2.8630us  cuDeviceGetCount
      0.00% 3.7720us     2  1.8860us 1.5370us 2.2350us  cuDeviceGet
      0.00% 2.0950us     1  2.0950us 2.0950us 2.0950us  cuDeviceTotalMem
      0.00% 1.6770us     1  1.6770us 1.6770us 1.6770us  cuDeviceGetUuid
[jahegou10@d261-pc8:~$ nvcc part1.cu -o part1
[jahegou10@d261-pc8:~$ nvprof ./part1
Matrice M1 :

Matrice M2 :
==9812== NVPROF is profiling process 9812, command: ./part1

Résultat addition GPU :

Résultat multiplication GPU :
==9812== Profiling application: ./part1
==9812== Profiling result:
      Type Time(%)    Time   Calls    Avg     Min     Max  Name
GPU activities:  69.23% 287.95ms    2 143.98ms 60.266ms 227.69ms [CUDA memcpy DtoH]
                  30.66% 127.54ms    2 63.769ms 63.729ms 63.808ms [CUDA memcpy HtoD]
                  0.11% 445.06us    1 445.06us 445.06us 445.06us  cudaMatrixMult(float*, float*, float*, int)
t)
                  0.00% 4.3840us    1 4.3840us 4.3840us 4.3840us  cudaMatrixAdd(float*, float*, float*, int
, int)
      API calls:  73.74% 417.32ms    4 104.33ms 60.905ms 228.56ms  cudaMemcpy
                  25.33% 143.36ms    3 47.788ms 106.51us 143.13ms  cudaMalloc
                  0.86% 4.8657ms    3 1.6219ms 399.57us 2.3931ms  cudaFree
                  0.05% 283.28us    101 2.8040us 1.1170us 71.797us  cuDeviceGetAttribute
                  0.01% 71.168us    2 35.584us 20.952us 50.216us  cuLaunchKernel
                  0.00% 13.759us    1 13.759us 13.759us 13.759us  cuDeviceGetName
                  0.00% 7.2630us    1 7.2630us 7.2630us 7.2630us  cuDeviceGetPCIBusId
                  0.00% 5.7260us    3 1.9080us 1.5360us 2.5840us  cuDeviceGetCount
                  0.00% 3.2930us    2 1.6410us 1.1880us 2.0950us  cuDeviceGet
                  0.00% 1.6760us    1 1.6760us 1.6760us 1.6760us  cuDeviceTotalMem
                  0.00% 1.5370us    1 1.5370us 1.5370us 1.5370us  cuDeviceGetUuid
jahegou10@d261-pc8:~$
```

Pour N = 1000,

```

Temps addition CPU : 7.4995 secondes
==9572== Profiling application: ./prog
==9572== Profiling result:
      Type Time(%)    Time   Calls    Avg     Min     Max  Name
GPU activities:  82.14% 16.923ms    1 16.923ms 16.923ms 16.923ms  cudaMatrixMult(float*, float*, float*, int)
                  10.91% 2.2470ms    2 1.1235ms 546.66us 1.7004ms  [CUDA memcpy DtoH]
                  6.65% 1.3698ms    2 684.92us 667.33us 702.50us  [CUDA memcpy HtoD]
                  0.31% 62.880us    1 62.880us 62.880us 62.880us  cudaMatrixAdd(float*, float*, float*, int, int)
      API calls:  87.58% 166.88ms    3 55.627ms 116.78us 166.64ms  cudaMalloc
                  11.87% 22.612ms    4 5.6530ms 874.91us 17.757ms  cudaMemcpy
                  0.30% 569.14us    3 189.71us 169.30us 207.08us  cudaFree
                  0.18% 352.21us    101 3.4870us 1.3960us 83.392us  cuDeviceGetAttribute
                  0.03% 60.832us    2 30.416us 20.184us 40.648us  cuLaunchKernel
                  0.01% 28.006us    1 28.006us 28.006us 28.006us  cuDeviceGetPCIBusId
                  0.01% 21.441us    3 7.1470us 1.8160us 16.063us  cuDeviceGetCount
                  0.01% 18.368us    1 18.368us 18.368us 18.368us  cuDeviceGetName
                  0.00% 4.1200us    2 2.0600us 1.5360us 2.5840us  cuDeviceGet
                  0.00% 2.0960us    1 2.0960us 2.0960us 2.0960us  cuDeviceGetUuid
                  0.00% 2.0950us    1 2.0950us 2.0950us 2.0950us  cuDeviceTotalMem
```

Pour N = 1000, avec une grille de (1,1)

```

Temps addition CPU : 7.3717 secondes
==9438== Profiling application: ./prog
==9438== Profiling result:
      Type Time(%)    Time   Calls    Avg     Min     Max  Name
GPU activities:  60.38% 2.2485ms    2 1.1242ms 543.23us 1.7053ms  [CUDA memcpy DtoH]
                  36.84% 1.3717ms    2 685.83us 650.95us 720.71us  [CUDA memcpy HtoD]
                  2.67% 99.585us    1 99.585us 99.585us 99.585us  cudaMatrixMult(float*, float*, float*, int)
                  0.10% 3.9040us    1 3.9040us 3.9040us 3.9040us  cudaMatrixAdd(float*, float*, float*, int, int)
      API calls:  96.16% 167.08ms    3 55.692ms 115.10us 166.85ms  cudaMalloc
                  3.29% 5.7103ms    4 1.4276ms 853.68us 3.0422ms  cudaMemcpy
                  0.32% 559.02us    3 186.34us 178.87us 200.80us  cudaFree
                  0.17% 295.22us    101 2.9230us 908ns 80.109us  cuDeviceGetAttribute
                  0.04% 70.122us    2 35.061us 22.210us 47.912us  cuLaunchKernel
                  0.01% 15.714us    1 15.714us 15.714us 15.714us  cuDeviceGetName
                  0.01% 12.082us    1 12.082us 12.082us 12.082us  cuDeviceGetPCIBusId
                  0.00% 5.4480us    3 1.8160us 1.3970us 2.6540us  cuDeviceGetCount
                  0.00% 3.4220us    2 1.7110us 1.3270us 2.0950us  cuDeviceGet
                  0.00% 1.9550us    1 1.9550us 1.9550us 1.9550us  cuDeviceTotalMem
                  0.00% 1.4660us    1 1.4660us 1.4660us 1.4660us  cuDeviceGetUuid
```

Pour N = 10 000

```
[jahegouli0@d261-pc8:~$ ./part1
n = 10000, p = 10000
Matrice M1 :

Matrice M2 :
==12756== NVPROF is profiling process 12756, command: ./part1
blockDim = 16 16
grid dim: 625 625

Résultat addition GPU :

Résultat multiplication GPU :
==12756== Profiling application: ./part1
==12756== Profiling result:
      Type  Time(%)    Time     Calls    Avg     Min     Max   Name
GPU activities:  96.52% 11.8270s      1 11.8270s 11.8270s 11.8270s  cudaMatrixMult(float*, float*, float*, int)
                2.37% 289.86ms      2 144.93ms 59.780ms 230.08ms [CUDA memcpy DtoH]
                1.05% 128.29ms      2 64.146ms 63.882ms 64.490ms [CUDA memcpy HtoD]
                0.07% 8.3888ms      1 8.3888ms 8.3888ms 8.3888ms  cudaMatrixAdd(float*, float*, float*, int
, int)
    API calls:  98.79% 12.2550s      4 3.06375s 63.973ms 11.8870s  cudaMemcpy
                1.16% 144.35ms      3 48.118ms 122.08us 144.10ms  cudaMemcpy
                0.04% 4.8504ms      3 1.6168ms 400.06us 2.3446ms  cudaFree
                0.00% 268.26us      101 2.6560us 1.1170us 63.835us  cuDeviceGetAttribute
                0.00% 89.049us      2 44.524us 43.302us 45.747us  cuLaunchKernel
                0.00% 12.920us      1 12.920us 12.920us 12.920us  cuDeviceGetName
                0.00% 10.756us      1 10.756us 10.756us 10.756us  cuDeviceGetPCIBusId
                0.00% 5.3780us      3 1.7920us 1.3970us 2.4449us  cuDeviceGetCount
                0.00% 3.1430us      2 1.5710us 1.3270us 1.8160us  cuDeviceGet
                0.00% 1.8860us      1 1.8860us 1.8860us 1.8860us  cuDeviceTotalMem
                0.00% 1.4670us      1 1.4670us 1.4670us 1.4670us  cuDeviceGetUuid
[jahegouli0@d261-pc8:~$ ]
```

```
n = 10000, p = 10000
Matrice M1 :

Matrice M2 :
==12654== NVPROF is profiling process 12654, command: ./part1
blockDim = 16 16
grid dim: 1 1

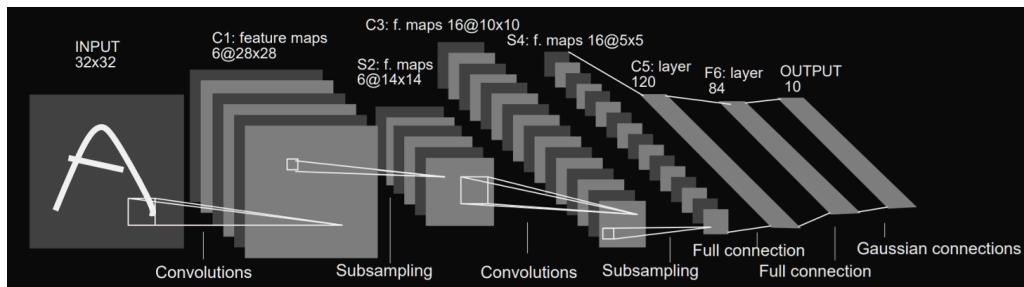
Résultat addition GPU :

Résultat multiplication GPU :
==12654== Profiling application: ./part1
==12654== Profiling result:
      Type  Time(%)    Time     Calls    Avg     Min     Max   Name
GPU activities:  68.86% 287.23ms      2 143.61ms 59.475ms 227.75ms  [CUDA memcpy DtoH]
                30.86% 128.73ms      2 64.366ms 64.318ms 64.414ms  [CUDA memcpy HtoD]
                0.28% 1.1825ms      1 1.1825ms 1.1825ms 1.1825ms  cudaMatrixMult(float*, float*, float*, int
, int)
                0.00% 4.4160us      1 4.4160us 4.4160us 4.4160us  cudaMatrixAdd(float*, float*, float*, int
, int)
    API calls:  73.38% 418.54ms      4 104.64ms 60.850ms 228.64ms  cudaMemcpy
                25.88% 147.59ms      3 49.197ms 186.86us 147.36ms  cudaMemcpy
                0.67% 3.8167ms      3 1.2722ms 407.18us 2.1044ms  cudaFree
                0.05% 307.58us      101 3.0450us 1.1170us 83.043us  cuDeviceGetAttribute
                0.02% 103.71us      2 51.857us 33.883us 69.911us  cuLaunchKernel
                0.00% 13.549us      1 13.549us 13.549us 13.549us  cuDeviceGetName
                0.00% 6.9140us      1 6.9140us 6.9140us 6.9140us  cuDeviceGetPCIBusId
                0.00% 6.0060us      3 2.0020us 1.3270us 3.0030us  cuDeviceGetCount
                0.00% 3.3520us      2 1.6760us 1.1870us 2.1650us  cuDeviceGet
                0.00% 1.8160us      1 1.8160us 1.8160us 1.8160us  cuDeviceGetUuid
                0.00% 1.6770us      1 1.6770us 1.6770us 1.6770us  cuDeviceTotalMem
```

Bien que théoriquement une configuration avec de nombreux threads et blocs (blockDim(624, 624)) devrait permettre d'exécuter des calculs plus rapidement grâce au parallélisme, dans la pratique, une configuration mal optimisée peut entraîner un temps d'exécution bien plus long. Voici pourquoi, dans votre cas, la configuration séquentielle blockDim(1,1) est plus rapide que blockDim(624, 624) .

Partie 2. Premières couches du réseau de neurones LeNet-5 :

Cette partie du travail pratique vise à implémenter les premières couches du réseau de neurones convolutifs LeNet-5 en exploitant les capacités de calcul parallèle des GPU via CUDA.



Matrices utilisées :

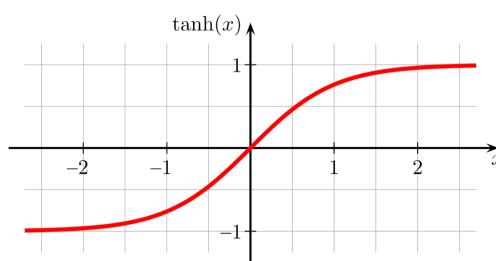
- **h_raw_data** : données d'entrée de taille 32×32 .
- **h_C1_kernel** : noyaux de convolution, 6 noyaux de taille 5×5 .
- **h_C1_data** : résultats de la première convolution, dimensions $6 \times 28 \times 28$.
- **h_S1_data** : résultats après sous-échantillonnage, dimensions $6 \times 14 \times 14$.

Construction des couches :

C1 : Feature maps (**h_C1_data**) est donné par le produit de convolution de INPUT (**h_raw_data**) et **h_C1_kernel**. La convolution 2D est réalisée en parallèle sur GPU à l'aide du kernel CUDA **convolution2D**.

C1 : Feature maps (**h_S1_data**) est donné par un sous-échantillonnage appliqué sur la couche (**h_C1_data**). Le sous-échantillonnage est implémenté via le kernel **subsampling2D**, basé sur le moyennage des blocs 2×2 (il est équivalent à un Average pooling(2×2) en Python).

Entre chaque layer, on applique une fonction d'activation **tanh**. Cette fonction d'activation tangente hyperbolique est implémentée comme un kernel CUDA **applyActivationTanh**.



```

__device__ float activation_tanh(float M) {
    float e_plus = expf(M); // e^M
    float e_minus = expf(-M); // e^{-M}
    return (e_plus - e_minus) / (e_plus + e_minus); // tanh(M)

```

Partie 3. Un peu de Python

Contenu du fichier ipynb:

- **Train set :** 60 000 exemples.
- **Test set :** 10 000 exemples.

Chaque exemple est une image de taille 28×28 .

- **Normalisation :** les pixels sont normalisés entre 0 et 1 $pixel\ norm = pixel/255$.
- **Adam :** pour un ajustement rapide des poids.
- **Dense 1 :** 120 neurones (ReLU).
- **Dense 2 :** 84 neurones (ReLU).
- **Dense 3 :** 10 neurones (Softmax, pour la classification).

Model summary :

Layer (type)	Output Shape	Param #
conv2d_11 (Conv2D)	(None, 28, 28, 6)	156
average_pooling2d_10 (AveragePooling2D)	(None, 14, 14, 6)	0
conv2d_12 (Conv2D)	(None, 10, 10, 16)	2,416
average_pooling2d_11 (AveragePooling2D)	(None, 5, 5, 16)	0
flatten_5 (Flatten)	(None, 400)	0
dense_15 (Dense)	(None, 120)	48,120
dense_16 (Dense)	(None, 84)	10,164
dense_17 (Dense)	(None, 10)	850

Synthèse des différents layers et fonctions manquantes :

COUCHE	STATUT DES FONCTIONS
Conv2d(28, 28, 6)	Créé dans les parties précédentes.
Average pooling2d (14, 14, 6)	Créé dans les parties précédentes.

Conv2d_1(10, 10, 16)	Créé dans les parties précédentes.
Average pooling2d_1 (5, 5, 16)	Créé dans les parties précédentes.
Flatten FL (400)	Pas besoin, on travaille sur un tableau 1D.
Dense (120)	Manquante
Dense_1 (84)	Manquante
Dense_2 (10)	Manquante

Synthèse des fonctions facilement parallélisables :

Fonctions	Description	Raison
Convolution 2D	Applique un noyau pour produire une feature map.	Calculs pour chaque pixel indépendants.
Sous-échantillonnage	Réduction des dimensions par moyenne	Chaque pixel de la sortie dépend d'un bloc local.
Produit matriciel	Multiplie l'entrée par les poids des couches denses.	Chaque élément de la matrice résultante est indépendant.
Calcul de la perte	Compare les prédictions et les étiquettes pour l'erreur.	Chaque exemple du batch est traité indépendamment.
Normalisation	Ajuste les pixels ou valeurs entre 0 et 1.	Chaque pixel ou élément est traité indépendamment.

Résultats :

```
Résultats finaux (Probabilités Softmax) :  
Classe 0 : 0.230634  
Classe 1 : 0.010422  
Classe 2 : 0.167865  
Classe 3 : 0.071030  
Classe 4 : 0.018155  
Classe 5 : 0.342009  
Classe 6 : 0.022548  
Classe 7 : 0.051520  
Classe 8 : 0.073182  
Classe 9 : 0.012635  
Classe prédictive : 5 avec probabilité 0.342009
```

Commentaire :

Les résultats obtenus avec notre implémentation CUDA montrent des prédictions globalement correctes pour les classes, mais avec des probabilités associées moins confiantes que celles obtenues avec un modèle Python équivalent. Par exemple, pour une image appartenant à la classe prédictive 5, la probabilité associée est de 0.34, contre une probabilité typique de 0.98 avec le modèle Python.

Comment expliquer cette différence ?

Une petite différence dans les poids ou biais entre les deux implémentations (par exemple, due à des arrondis ou erreurs dans le chargement des fichiers) peut se propager à travers les couches, amplifiant les écarts. On a quand même prêté attention à bien avoir les poids et biais correctement importés.