Computer Architecture Final Project

309505001 翁紹恩

A. Title

a. Image transformation – Mirror and grayscale

b. 本次實驗實作影像的鏡射同時轉換成灰階

c. 現今機器學習與影像處理當紅，在處理影像以方便當作各種機器學習的輸入資料時，常會需要各種角度的轉換圖片或是適時轉換成灰階，以增加學習資料多樣及降低模型學習難度，例如下圖，學習自監督式學習 (self-supervised learning) 所使用的方法即為將輸入的圖片轉成四個角度仍然可以學習出原始的圖片，利用這種方式以確保就算圖片以不同角度呈現，模型仍然可以學習出正確的結果。

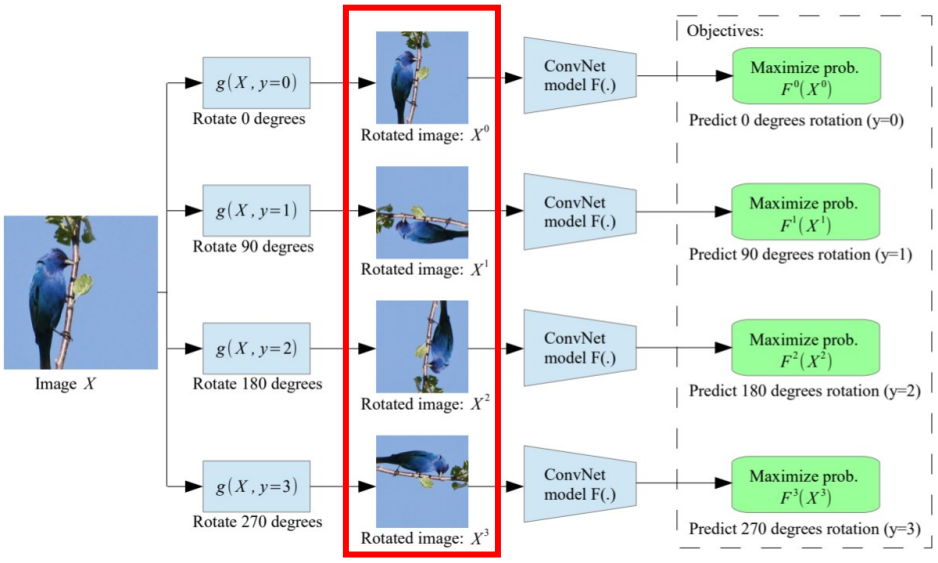


圖 1. 取自 ICLR 2018 UNSUPERVISED REPRESENTATION LEARNING BY PREDICTING IMAGE ROTATIONS

另外，以 ECCV 2018 workshop Pre-training on Grayscale ImageNet Improves Medical Image Classification 為例，此篇文章證明了在物體辨識上，很大部分顏色並沒辦法大幅的幫助訓練，反而因為顏色維度變多訓練變複雜，因此只有一維的灰階圖片反而能幫助訓練加快速度。

d. 綜合以上敘述，因此本文將嘗試:

i. **灰階轉換**

ii. 鏡射，但此次只會簡單實做**水平及上下鏡射**

iii. 以下關於演算法敘述只會著重以上兩點，其他讀圖片等等功能就不做討論

B. Environment

a. 使用助教提供 virtualbox

i. Memory: 1024MB

ii. Processer: 3 CPU

b. 需要安裝 c++ opencv library (詳細安裝過程寫在 Readme.txt)

C. File guidance

檔案名稱	說明
Readme.txt	說明安裝及執行流程
images.jpeg	一張 162*108 的圖片，為輸入資料
main.cu	host 端程式，會處理讀圖片、存圖片、把原始圖片放到 gpu 上的動作
cuda1.cu	device 端程式，在 gpu 上執行，使用作業一中方法一的資料切割方式
cuda2.cu	device 端程式，在 gpu 上執行，使用作業一中方法二的資料切割方式
cuda.h	cuda1、cuda2 的 header file
cuda1.sh	compile 執行 cuda1 相關指令，直接跑該 script 檔可以產生結果
cuda2.sh	compile 執行 cuda2 相關指令，直接跑該 script 檔可以產生結果
clean.sh	刪除所有結果
exe1	cuda1.cu 執行檔
exe2	cuda2.cu 執行檔
output1.jpg	執行 exe1 的產生結果
output2.jpg	執行 exe2 的產生結果

D. Algorithm

a. Color to Grayscale Conversion:

要將 rgb 圖片轉成灰階，有兩種方法，average method 和 weighted method:

i. Average method: Grayscale = R / 3 + G / 3 + B / 3.

ii. Weighted method: Grayscale = 0.299R + 0.587G + 0.114B

這次實作的為 weighted method，兩種方法都是正確的，但由於人眼對綠光較敏感對紅光較不敏感，所以加上 weight 較能真實反映人類腦中解讀的圖片。

b. Mirror:

此處的 vertical 和 horizontal 為垂直線映射或是水平線映射

i 為該點原始 index

numCols 為圖片 column 數

numRows 為圖片 row 數

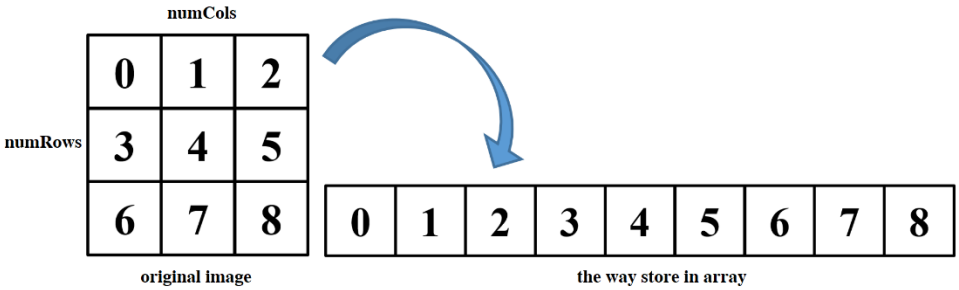


圖 2. 原始圖片和轉換到 gpu 上的儲存方式

i. Vertical:

可以由上面簡易示意圖簡單的推導出:

$$\text{轉換後的index} = \text{int}(\frac{\text{i}}{\text{numCols}}) * \text{numCols} + (\text{numCols} - \text{i\%numCols}) - 1$$

ii. Horizontal:

$$\text{轉換後的index} = \left(\text{numRows} - \text{int}\left(\frac{\text{i}}{\text{numCols}}\right) - 1\right) * \text{numCols} + (\text{i\%numCols})$$

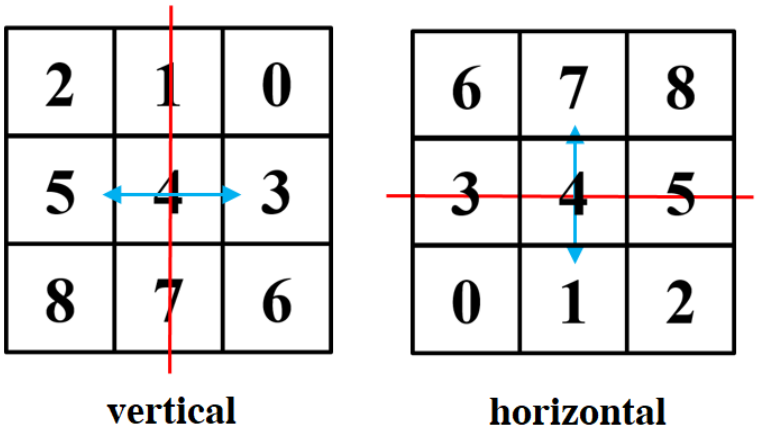


圖 3. 轉換後的圖片 index

E. How to work

a. 設置 block size 和 grid size 大小，需要注意設置兩者乘積為圖片 pixels 因數，否則會有 pixel 少計算到，且注意若設置太大可能引起 segmentation fault

b. 方法一及方法二切割方式皆如同作業一所做，剩下根據切割特性設置 for loop 做資料計算及讀取

c. Cuda1

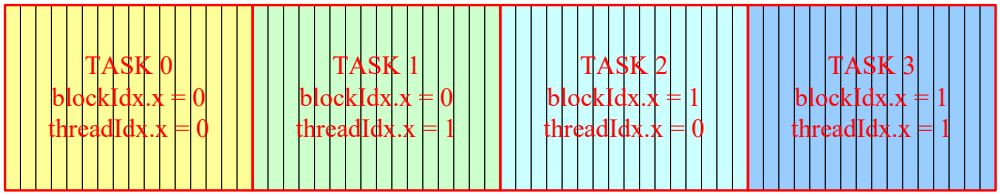


圖 4. 資料切割方式 取自劉志尉老師投影片

```
void mirror(uchar4* inputChannel, uchar4* outputChannel, int numRows, int numCols, bool vertical)
{
    int TotalThread = blockDim.x * gridDim.x;
    int stripe = numRows*numCols / TotalThread;
    int col = (blockIdx.x * blockDim.x + threadIdx.x) * stripe;
    int LoopLim = col + stripe;
```

方法一切割方式

```
    for(int i=col ; i<LoopLim && i<numRows*numCols; i++)
    {
        unsigned char Y = 0.299f * inputChannel[i].x + 0.587 * inputChannel[i].y + 0.114 * inputChannel[i].z;
        if(vertical)
            outputChannel[i/numCols*numCols+(numCols-i%numCols)-1] = make_uchar4(Y, Y, Y, 255);
        else
            outputChannel[(numRows- (i/numCols) -1)*numCols +(i%numCols)] = make_uchar4(Y, Y, Y, 255);
    }
}
```

轉灰階

鏡射

d. Cuda2

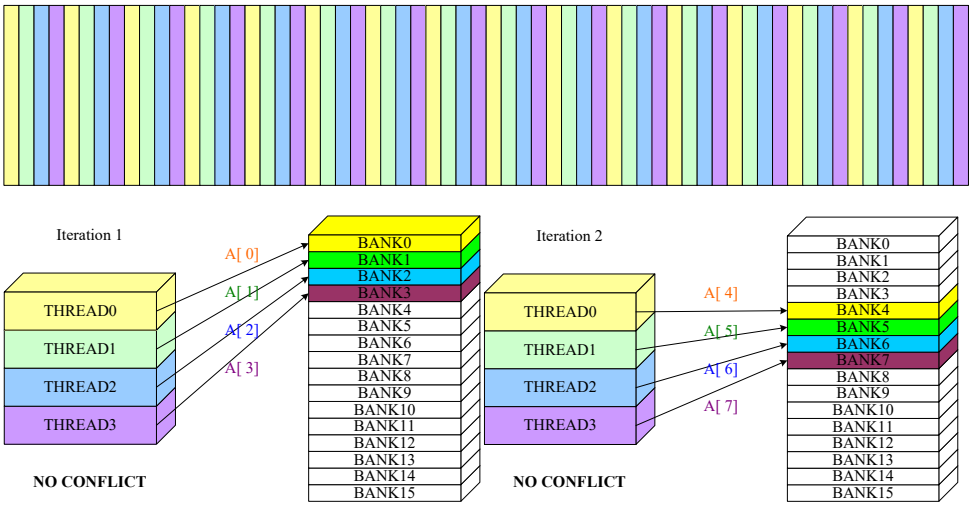


圖 5. 資料切割方式 取自劉志尉老師投影片

```
void mirror(uchar4* inputChannel, uchar4* outputChannel, int numRows, int numCols, bool vertical)
{
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int stripe = blockDim.x * gridDim.x;
    for(int i=col; i<numRows*numCols; i=i+stripe)
    {
        unsigned char Y = 0.299 * inputChannel[i].x + 0.587 * inputChannel[i].y + 0.114 * inputChannel[i].z;
        if(vertical)
            outputChannel[i/numCols*numCols+(numCols-i%numCols)-1] = make_uchar4(Y, Y, Y, 255);
        else
            outputChannel[(numRows- (i/numCols) -1)*numCols +(i%numCols)] = make_uchar4(Y, Y, Y, 255);
    }
}
```

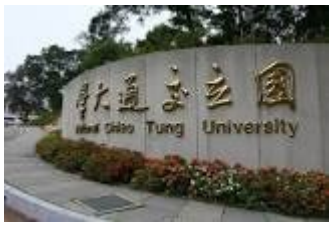
方法二切割方式

轉灰階

鏡射

F. Simulation result

a. Original image



b. Vertical

```
ca_project@CUDA:~/final$ sh cuda1.sh
Finish compiling cuda1.cu
Finish compiling main.cu
Finish generating object code
GPU time = 223.32 ms
Finish executing
ca_project@CUDA:~/final$ sh cuda2.sh
Finish compiling cuda2.cu
Finish compiling main.cu
Finish generating object code
GPU time = 211.49 ms
Finish executing
```

c. Horizontal

```
ca_project@CUDA:~/final$ sh cuda1.sh
Finish compiling cuda1.cu
Finish compiling main.cu
Finish generating object code
GPU time = 221.52 ms
Finish executing
ca_project@CUDA:~/final$ sh cuda2.sh
Finish compiling cuda2.cu
Finish compiling main.cu
Finish generating object code
GPU time = 198.35 ms
Finish executing
```

d. 由上圖結果可以看到使用方法二皆比較快速，因為 memory 存放方式的關係使用方法一在 memory 較容易有 conflict 的情形，方法二則不會。