Discussion #6:

CUDA Tutorial & Convolutional Neural Network

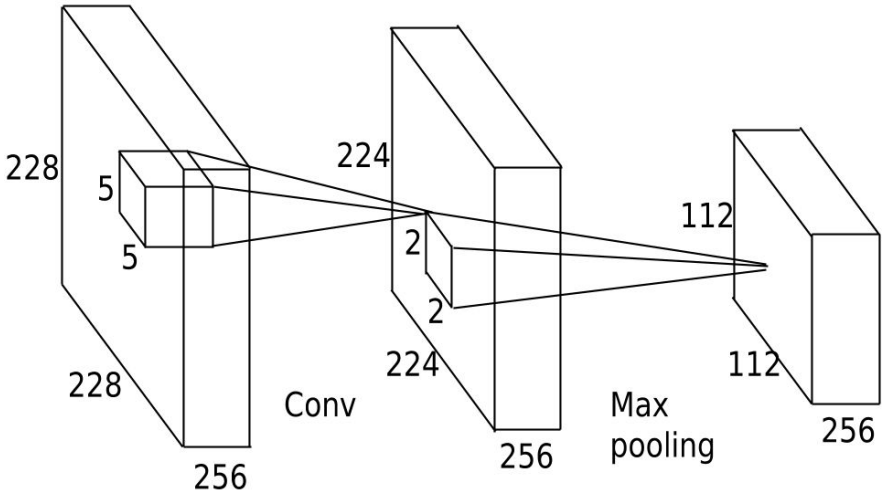
Karl Marrett, Jason Lau, and Jason Cong

Outline

- **Lab 3: Convolutional Neural Network Workload**
- **Live code**
- **Q & A**

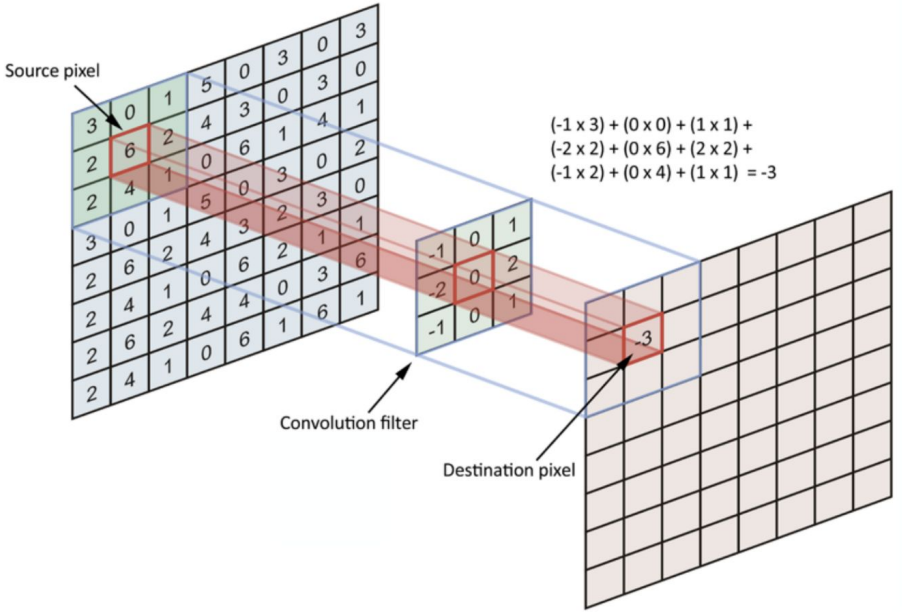
Lab 3: Workload to Accelerate

- Four (4) Loops:
  a. Conv Layer
     - Set Bias
     - Convolution
     - ReLU
  b. Max Pooling Layer



Lab 3: Workload to Accelerate

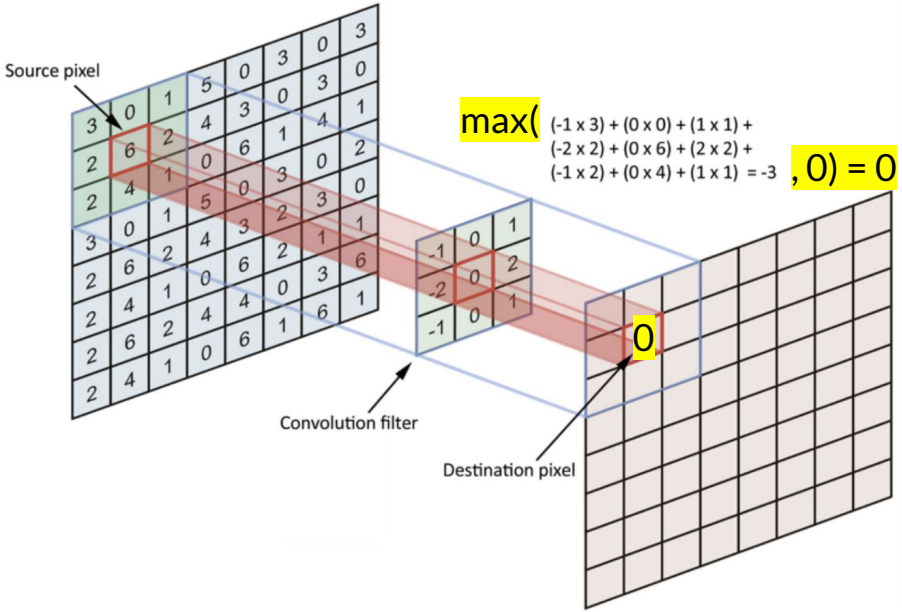
- Four (4) Loops:
  a. Conv Layer
     - Set Bias
     - Convolution
     - ReLU
  b. Max Pooling Layer



https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

Lab 3: Workload to Accelerate

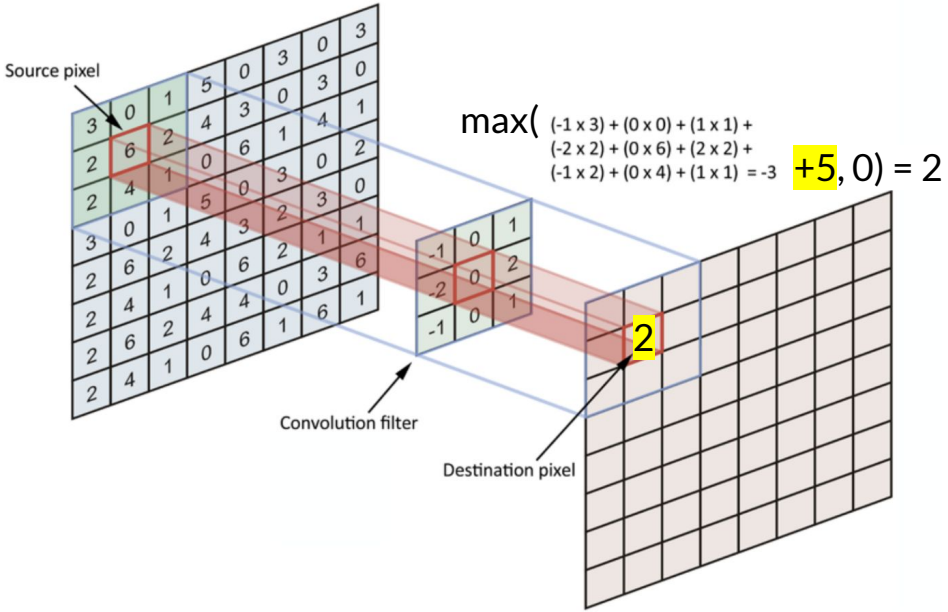
- Four (4) Loops:
  a. Conv Layer
     - Set Bias
     - Convolution
     - ReLU
  b. Max Pooling Layer



https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

Lab 3: Workload to Accelerate

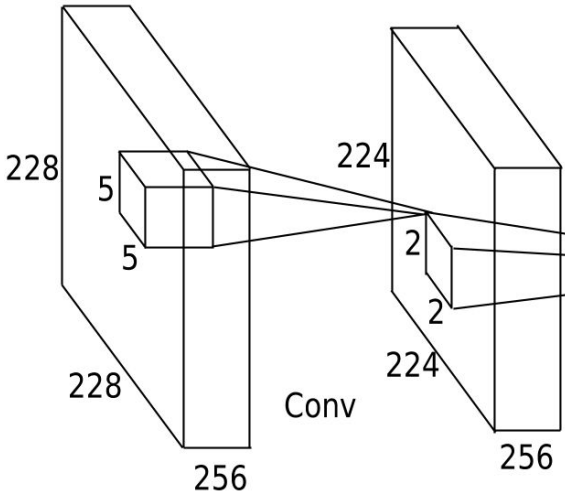
- Four (4) Loops:
  a. Conv Layer
     - Set Bias
     - Convolution
     - ReLU
  b. Max Pooling Layer

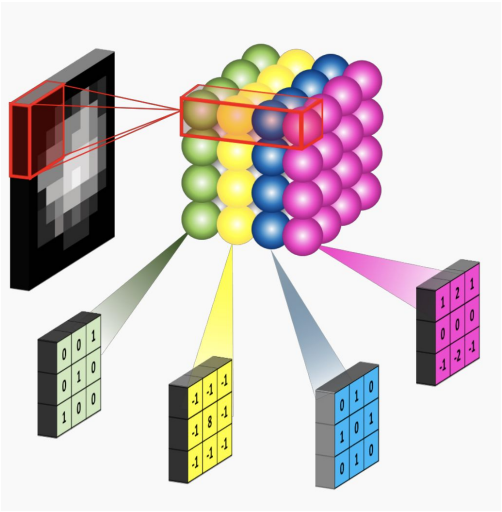


https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

Lab 3: Workload to Accelerate

- Four (4) Loops:
  - a. Conv Layer
    - Set Bias
    - Convolution
    - ReLU
  - b. Max Pooling Layer

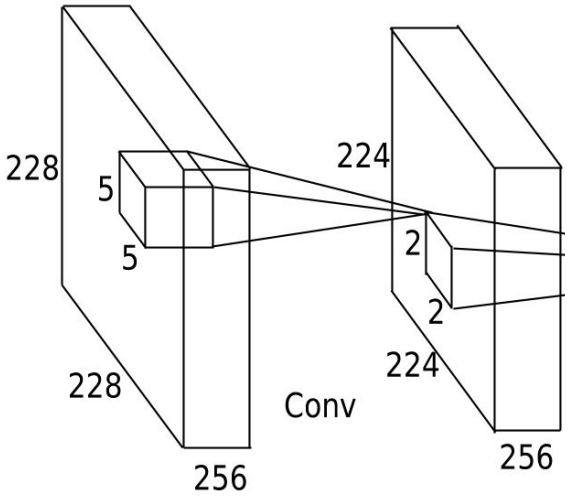


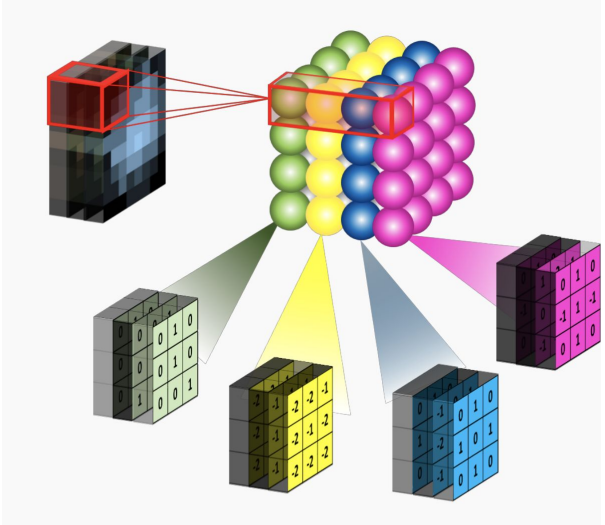


https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

Lab 3: Workload to Accelerate

- Four (4) Loops:
  - a. Conv Layer
    - Set Bias
    - Convolution
    - ReLU
  - b. Max Pooling Layer

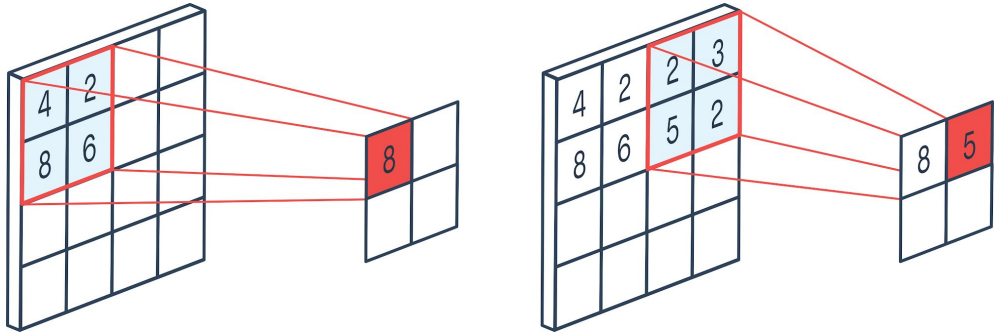




https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

Lab 3: Workload to Accelerate

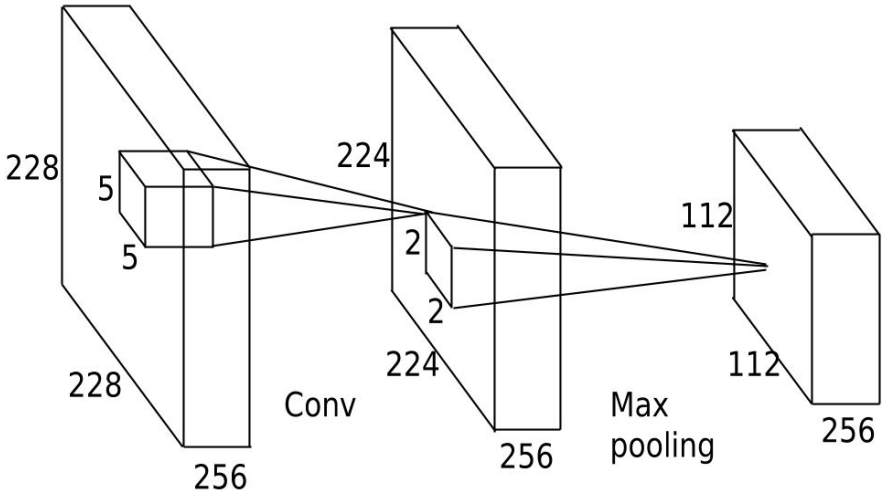
- Four (4) Loops:
  a. Conv Layer
     - Set Bias
     - Convolution
     - ReLU
  b. Max Pooling Layer



https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/2d-max-pooling-block

Lab 3: Workload to Accelerate

- **Four (4) Loops:**
  a. **Conv Layer**
     - **Set Bias**
     - **Convolution**
     - **ReLU**
  b. **Max Pooling Layer**



Lab 3: Workload to Accelerate

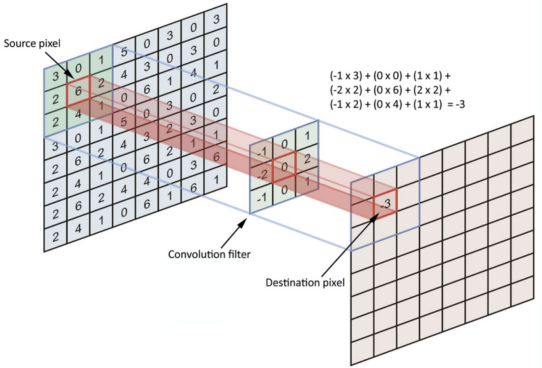
- **Four (4) Loops:**
  a. **Conv Layer**
     - **Set Bias**
     - **Convolution**
     - **ReLU**
  b. **Max Pooling Layer**

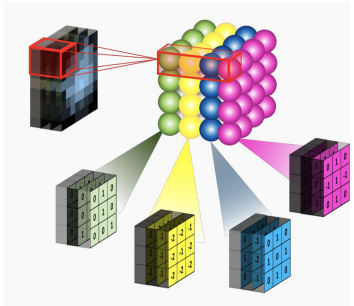
```
for (int i = 0; i < kNum; ++i)
  for (int h = 0; h < kImSize; ++h)
    for (int w = 0; w < kImSize; ++w)
      C[i][h][w] = bias[i];
```

Lab 3: Workload to Accelerate

- **Four (4) Loops:**
  - a. **Conv Layer**
    - **Set Bias**
    - **Convolution**
    - **ReLU**
  - b. **Max Pooling Layer**

```
for (int i = 0; i < kNum; ++i)
  for (int j = 0; j < kNum; ++j)
    for (int h = 0; h < kImSize; ++h)
      for (int w = 0; w < kImSize; ++w)
        for (int p = 0; p < kKernel; ++p)
          for (int q = 0; q < kKernel; ++q)
            C[i][h][w] += weight[i][j][p][q] *
                          input[j][h + p][w + q];
```





Lab 3: Workload to Accelerate

- **Four (4) Loops:**
  a. **Conv Layer**
     - **Set Bias**
     - **Convolution**
     - **ReLU**
  b. **Max Pooling Layer**

```
for (int i = 0; i < kNum; ++i)
  for (int h = 0; h < kImSize; ++h)
    for (int w = 0; w < kImSize; ++w)
      C[i][h][w] = max(0.f, C[i][h][w]);
```

Lab 3: Workload to Accelerate

- **Four (4) Loops:**
  a. **Conv Layer**
     - **Set Bias**
     - **Convolution**
     - **ReLU**
  b. **Max Pooling Layer**

```
for (int i = 0; i < kNum; ++i)
  for (int h = 0; h < kOutImSize; ++h)
    for (int w = 0; w < kOutImSize; ++w)
      output[i][h][w] = max(
          C[i][h * 2]    [w * 2    ],
          C[i][h * 2 + 1][w * 2    ]),
          C[i][h * 2    ][w * 2 + 1],
          C[i][h * 2 + 1][w * 2 + 1]);
```

Lab 3: Where Should I Start?

- Do NOT touch the host code (Hooray 🎉)
- Change `params.sh` if you want to change the workgroup sizes.
- Write your code in `kernel.cu`

Lab 3: Where Should I Start?

- **Four (4) Loops in `lib/cnn.cpp`**
- **Ctrl-C + Ctrl-V (or ⌘-C + ⌘-V) to `cnn.cl`**
  a. **Modify multidimensional access to one-dimensional.**
     - Discussed. `#define Input(x,y,z) input[(x)*Y*Z+(y)*Z+(z)];`
  b. **A serial version (that does not work).**
  c. **No sufficient resource to store the intermediate array C.**

- **Step 1: Experiment with sequential version**

Lab 3: Where Should I Start?

- **Step 1: Reduce memory usage (to get a working version).**

  Hint #1: Loop fusion and avoid resource waste

```
float C[I][H][W];
for i, for h, for w -- set bias for C[i][h][w]
for i, for j, for h, for w, for p, for q
                    -- compute      C[i][h][w]
for i, for h, for w -- relu         C[i][h][w]
for i, for h, for w -- maxpool      C[i][h][w]
                       and write to output[i][h/2][w/2]
```

Lab 3: Where Should I Start?

- **Step 1: Reduce memory usage (to get a working version).**

  Hint #2: Loop tiling

```
float C[I][H][W];
for i, for h, for w -- set bias for C[i][h][w]
for i, for j, for h, for w, for p, for q
                    -- compute      C[i][h][w]
for i, for h, for w -- relu         C[i][h][w]
for i, for h, for w -- maxpool      C[i][h][w]
                       and write to output[i][h/2][w/2]
```

Lab 3: Where Should I Start?

- Step 2: Make it faster 😄.

  Hint #1: Reuse data

  - Loop permutation and other transformations
  - Variable loop tiling sizes

  Hint #2: Parallelize easily parallelizable loops

  - Find your dependencies

  Hint #3: Spend your time on the bottlenecks

  - Profiler, or `gettimeofday`

Lab 3: Where Should I Start?

- **Step 2: Make it faster 😄.**

  Advanced Hint #1: Alignment

```
__private float c[C] __attribute__((aligned(16 * sizeof(float))));
```

  Advanced Hint #2: Manual Vectorization

  - **Automatically inferred in most cases**
  - **Try helping the compiler if not:**

```
for (i = 0; i < length; i += 16) {
    float16 a = vload16(0, &a[i]), b = vload16(0, &b[i]);
    vstore16(a + b, 0, &c[i]); }
```