ECE408/CS483/CSE408 Fall 2021

Applied Parallel Programming

Lecture 12: Computation in Deep Neural Networks

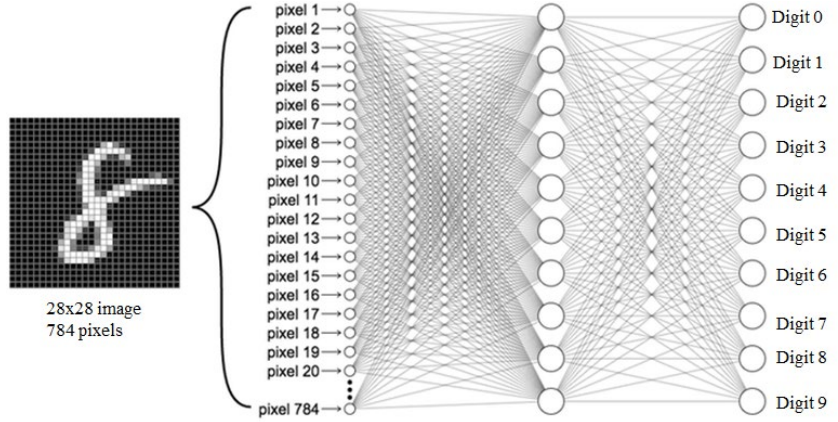
Course Reminders

- We are still grading Lab 4
- Midterm 1 is on Thursday, October 7th
  - On-line, everybody will be taking it at the same time
    - Thursday, Oct. 7th 8:00pm-9:20pm US Central time
    - Friday, Oct. 9th 9:00am-10:20am Beijing time
  - Includes materials from Lecture 1 through Lecture 10
  - See **https://wiki.illinois.edu/wiki/display/ECE408/Exams** for details
- Project Milestone 1: Rai Installation and baseline CPU implementation is due Friday October 15th
  - Project details to be posted this week on course wiki

Objective

- To learn to implement the different types of layers in a Convolutional Neural Network (CNN)

Multi-Layer Perceptron (MLP) for an Image



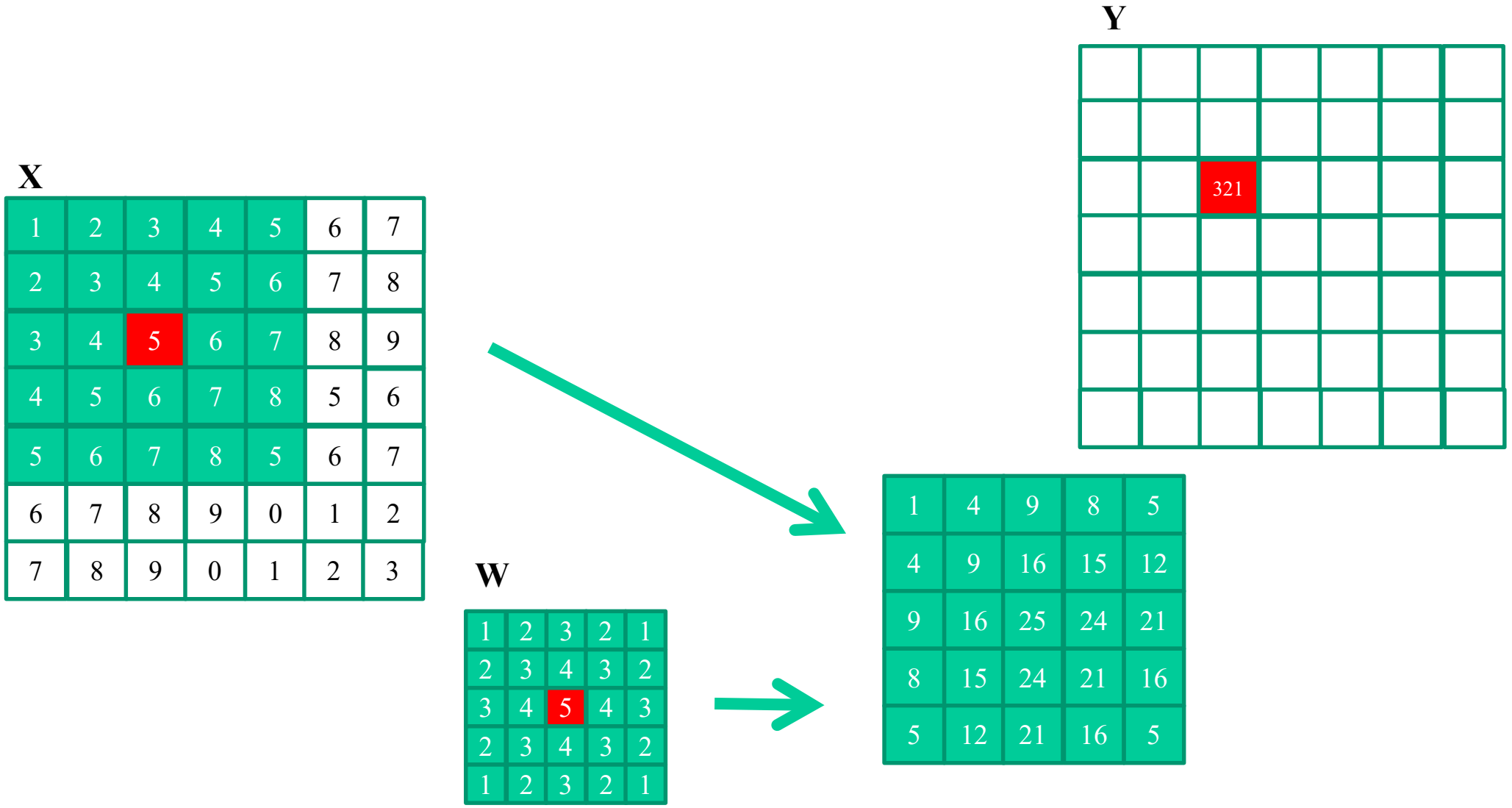
Consider a 250 x 250 image…

- input: 2D image treated as 1D vector
- Fully connected layer is huge:
  - 62,500 (250^2) weights per node!
  - Comparable number of nodes gives ~4B weights total!
- Need >1 hidden layer? Bigger images?
- Too much computation, and too much memory.

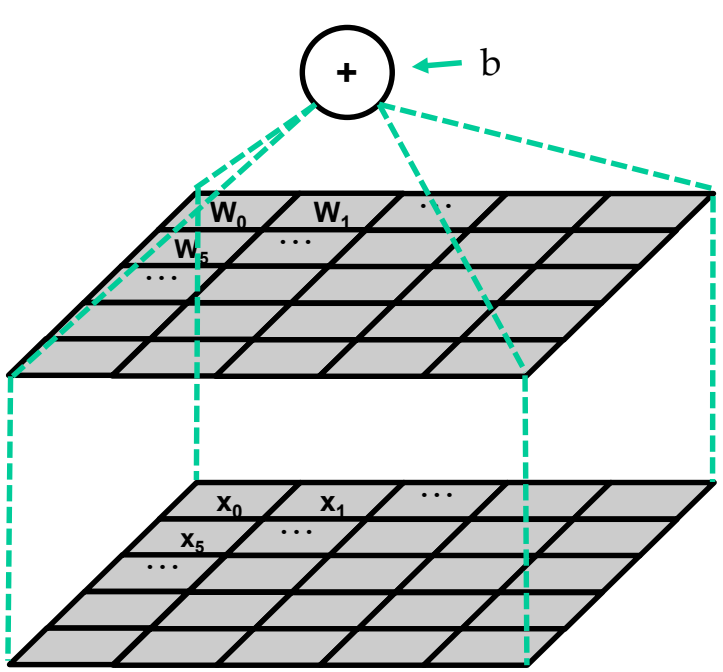
Traditional feature detection in image processing uses

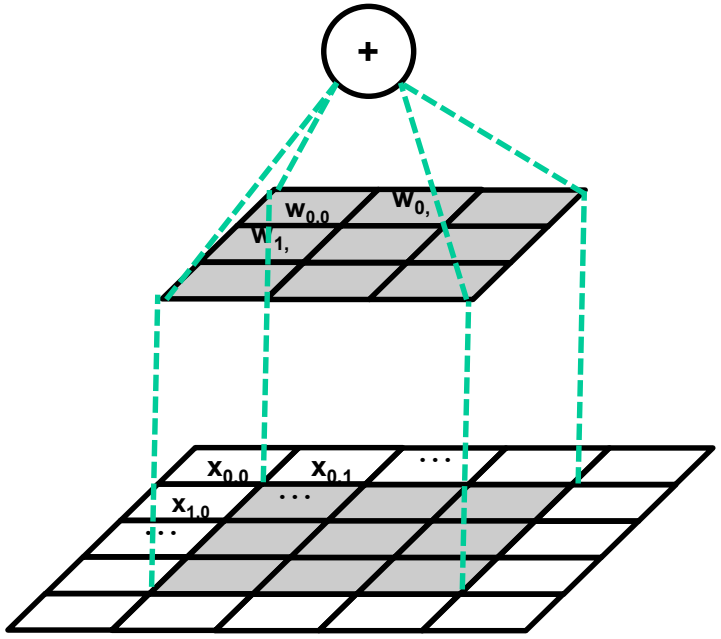
- Filters → Convolution kernels
- Can we use them in neural networks?

2-D Convolution



Convolution vs Fully-Connected (Weight Sharing)





Convolution Naturally Supports Varying Input Sizes

- As discussed so far,
  - perceptron layers have fixed structure, so
  - number of inputs / outputs is fixed.

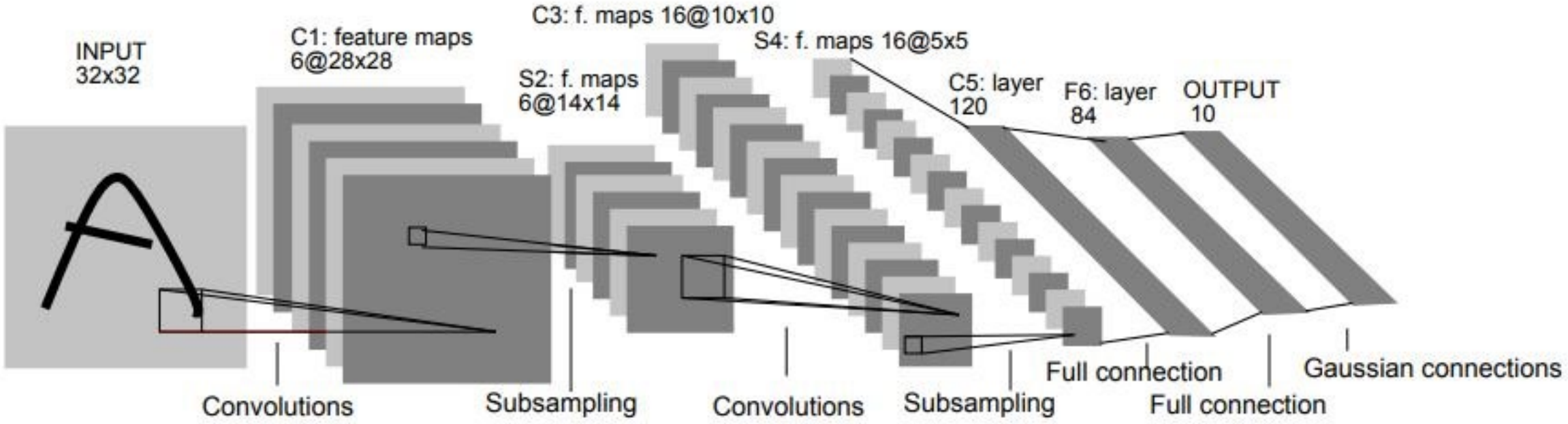
- Convolution enables variably-sized inputs (observations of the same kind of thing)
  - Audio recording of different lengths
  - Image with more/fewer pixels

Example Convolution Inputs

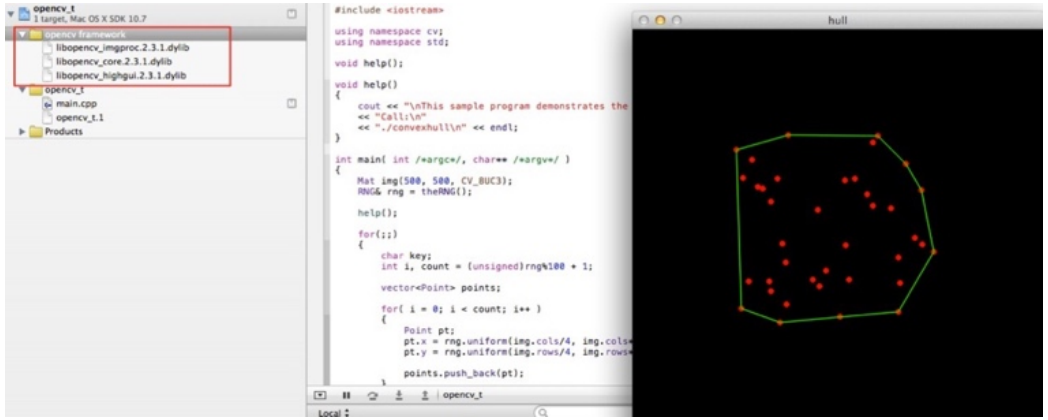
	Single-channel	Multi-channel
1D	audio waveform	Skeleton animation data: 1-D joint angles for each joint
2D	Fourier-transformed audio data Convolve over frequency axis: invariant to frequency shifts Convolve over time axis: invariant to shifts in time	Color image data: 2D data for R,G,B channels
3D	Volumetric data (example: medical imaging)	Color video: 2D data across 1D time for R,G,B channels

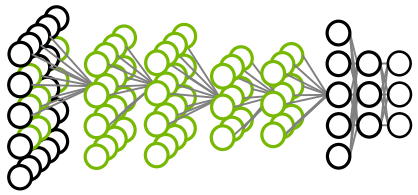
Deeplearningbook.org, ch 9, p 355

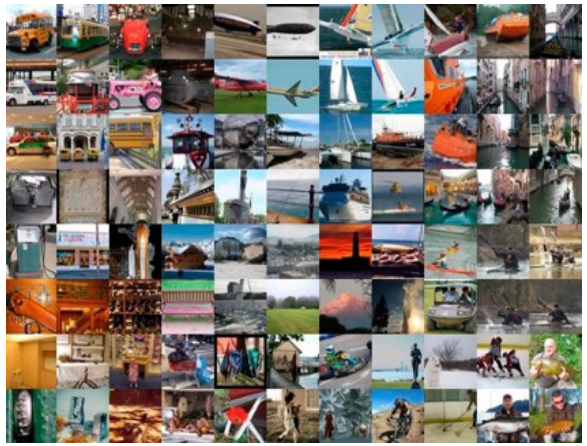
LeNet-5:CNN for hand-written digit recognition



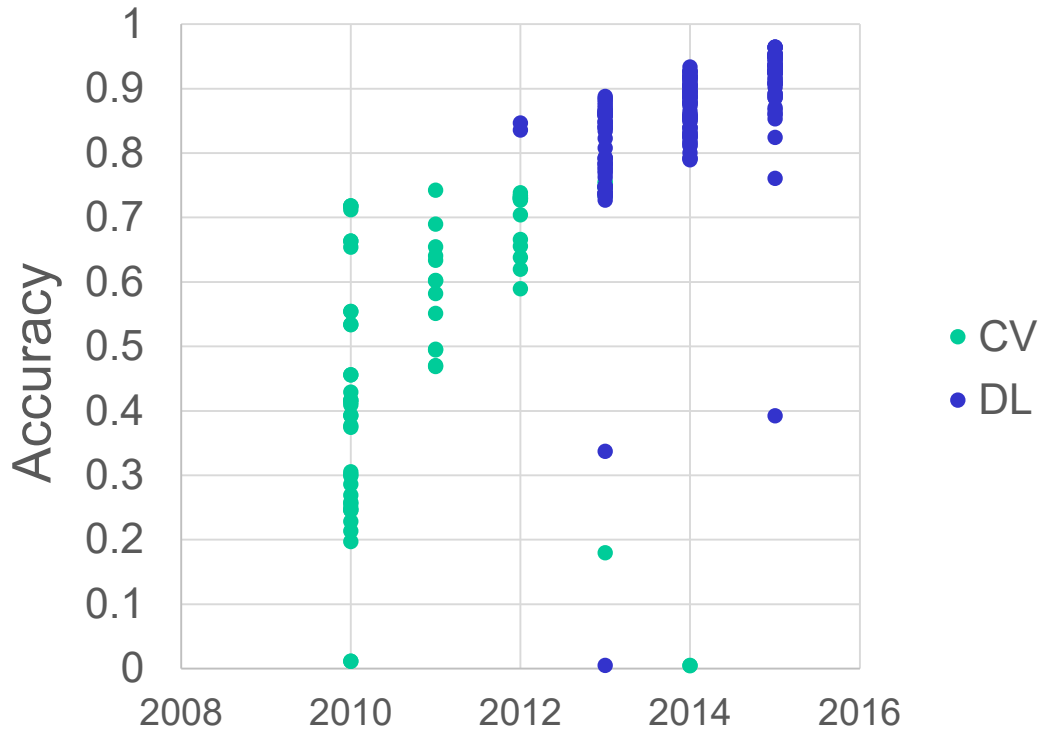
Deep Learning Impact in Computer Vision





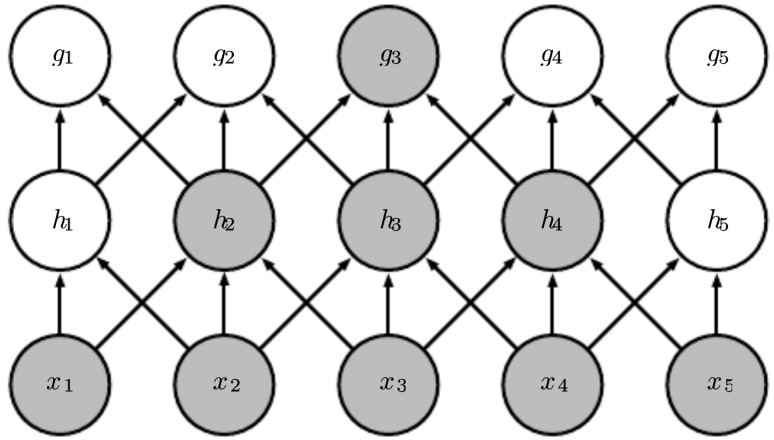


The Toronto team used GPUs and trained on 1.2M images in their 2012 winning entry at the Large Scale Visual Recognition Challenge



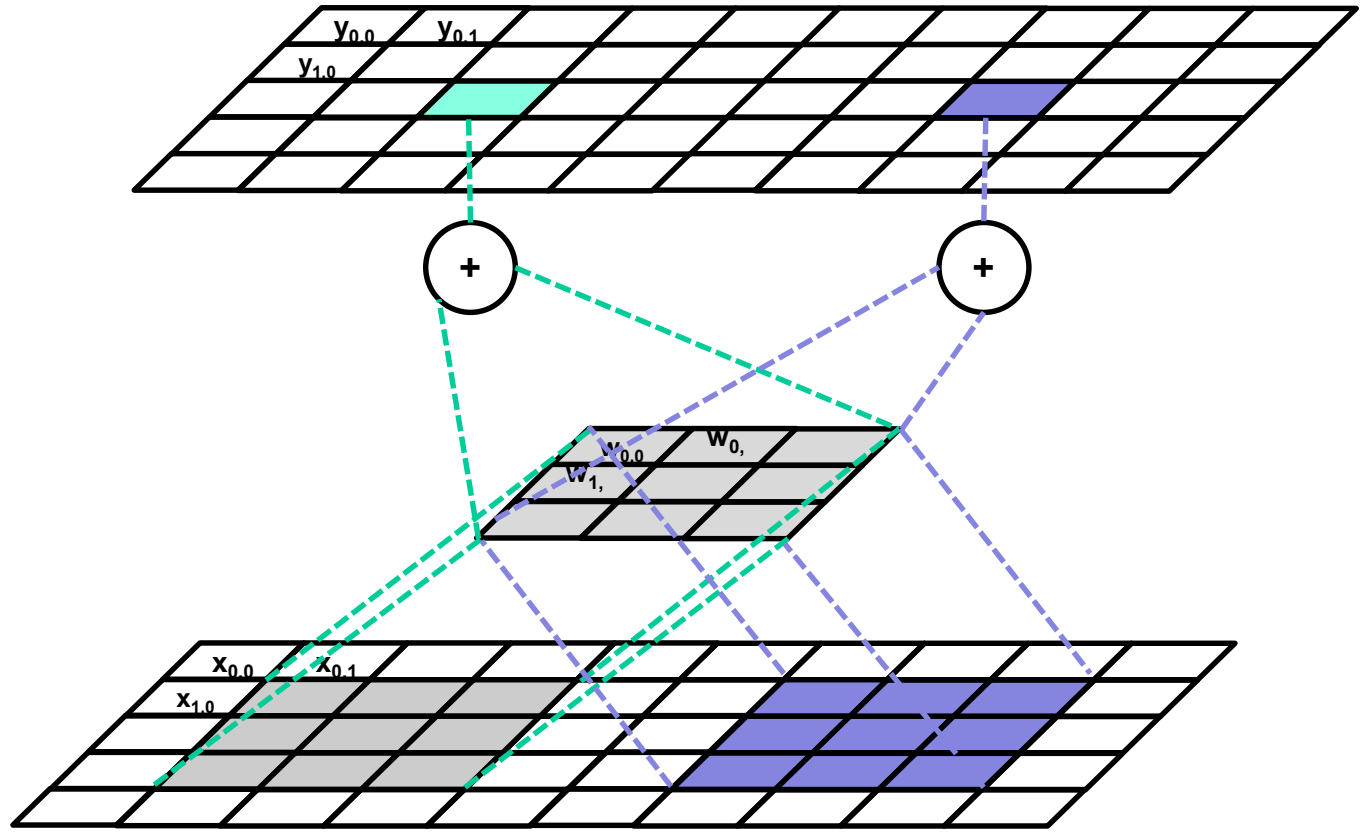
Why Convolution

- Sparse interactions
  - Meaningful features in small spatial regions
  - Need fewer parameters (less storage, better statistical characteristics, faster training)
  - Need multiple layers for wide receptive field



Why Convolution

- Parameter sharing
  - Kernel mask is applied repeatedly computing layer output
- Equivariant Representations
  - If input is translated, output is similarly translated
  - Output is a map of where features appear in input



Convolution

- 2-D Matrix
- $Y = W \otimes X$
- Kernel smaller than input: smaller receptive field
- Fewer Weights

MLP

- Vector
- $Y = w\ x + b$
- Maximum receptive field
- More weights

Anatomy of a Convolution Layer

Input features

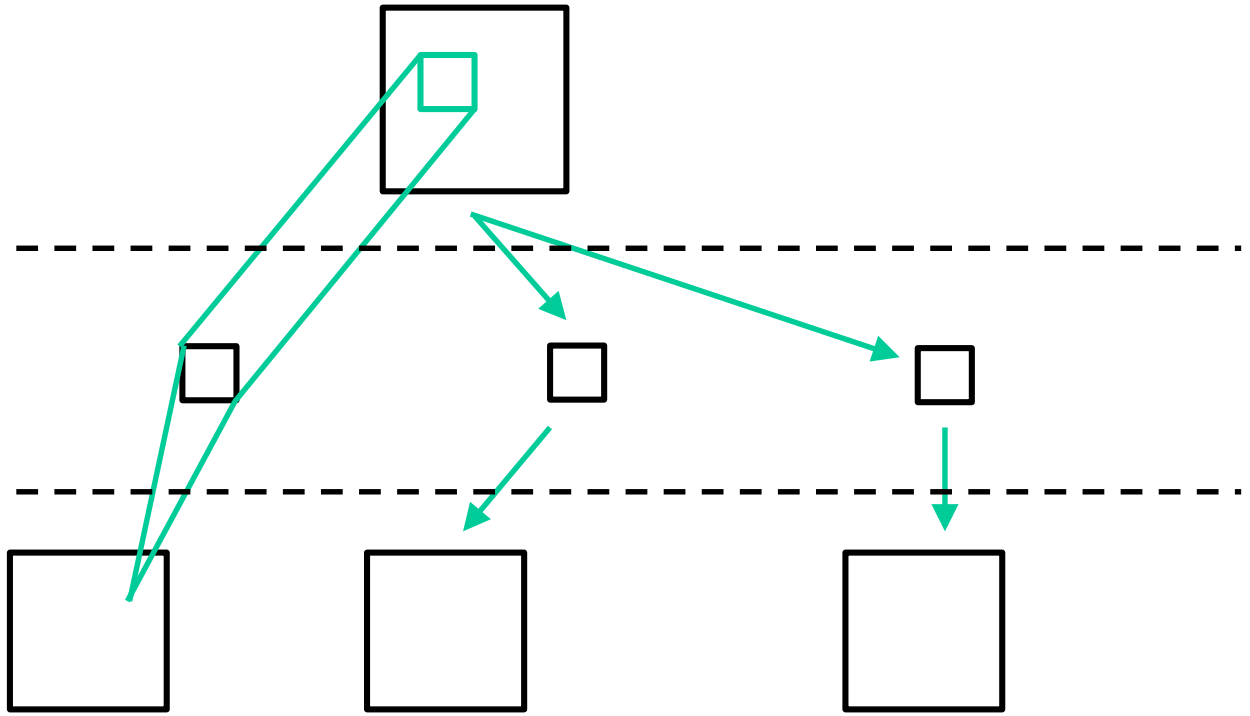
- A inputs each $N_1 \times N_2$

Convolution Layer

- B convolution kernels each $K_1 \times K_2$

Output Features (total of B)

- A × B outputs each
  $(N_1 - K_1+1) \times (N_2 - K_2+1)$



Notion of a Channel in Input Layer

Some Set of Input Features are Related

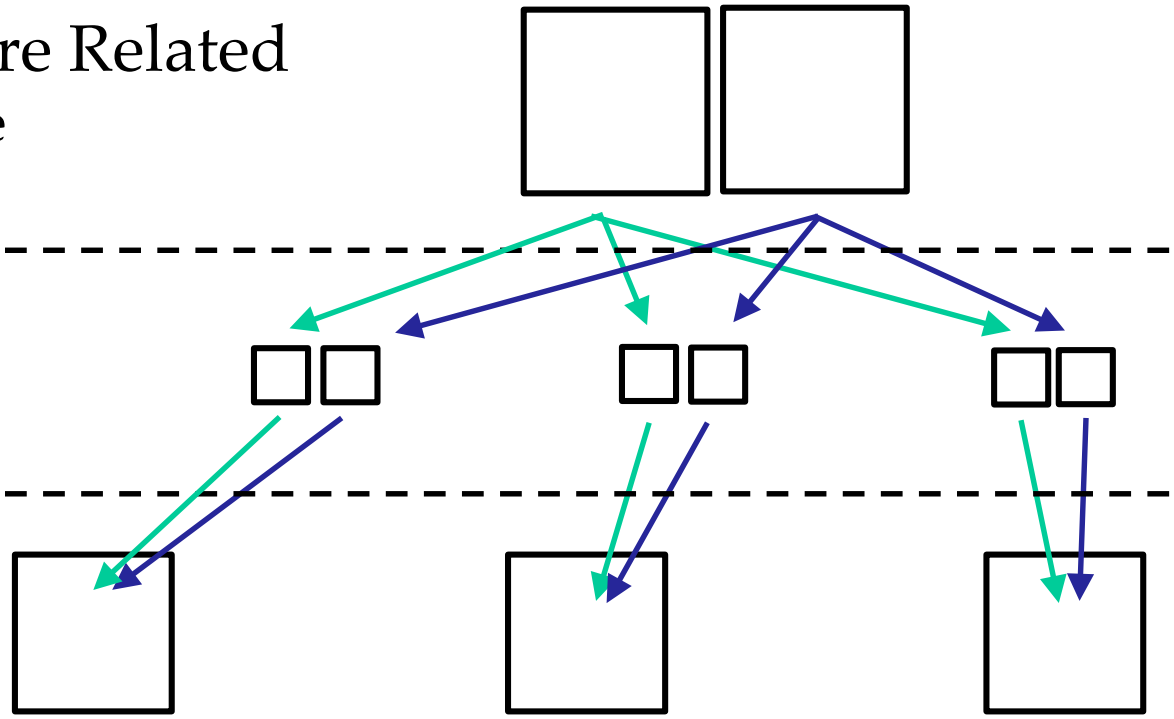
- For example: Red, Green, Blue

Convolution Layer

- Different kernels per channel

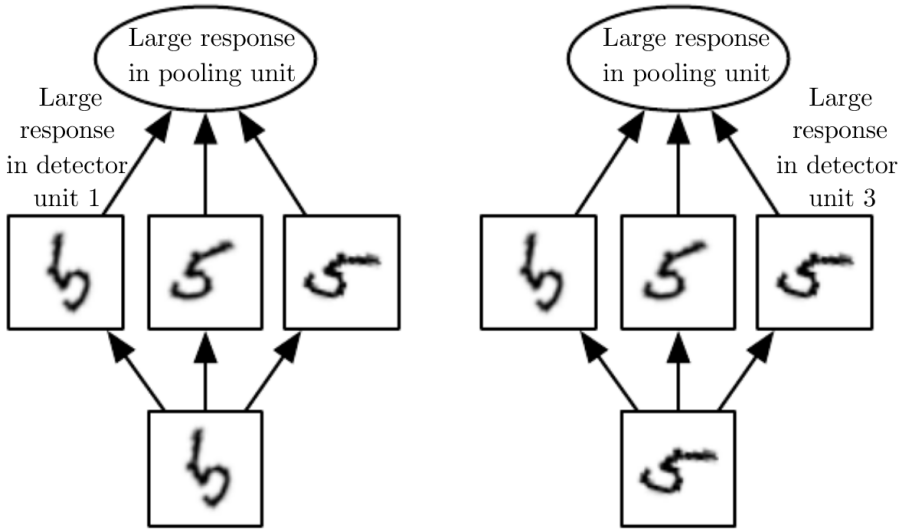
Output Features

- Channels combine per output feature

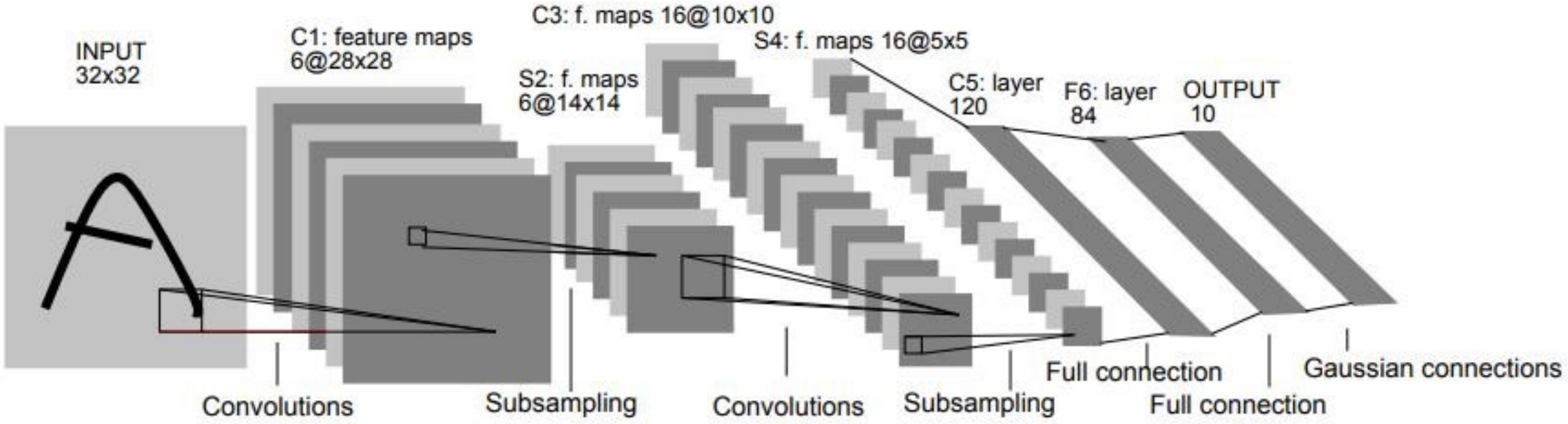


2-D Pooling (Subsampling)

- A subsampling layer
  - Sometimes with bias and non-linearity built in
- Common types
  - max, average, L^2 norm, weighted average
- Helps make representation invariant to size scaling and small translations in the input



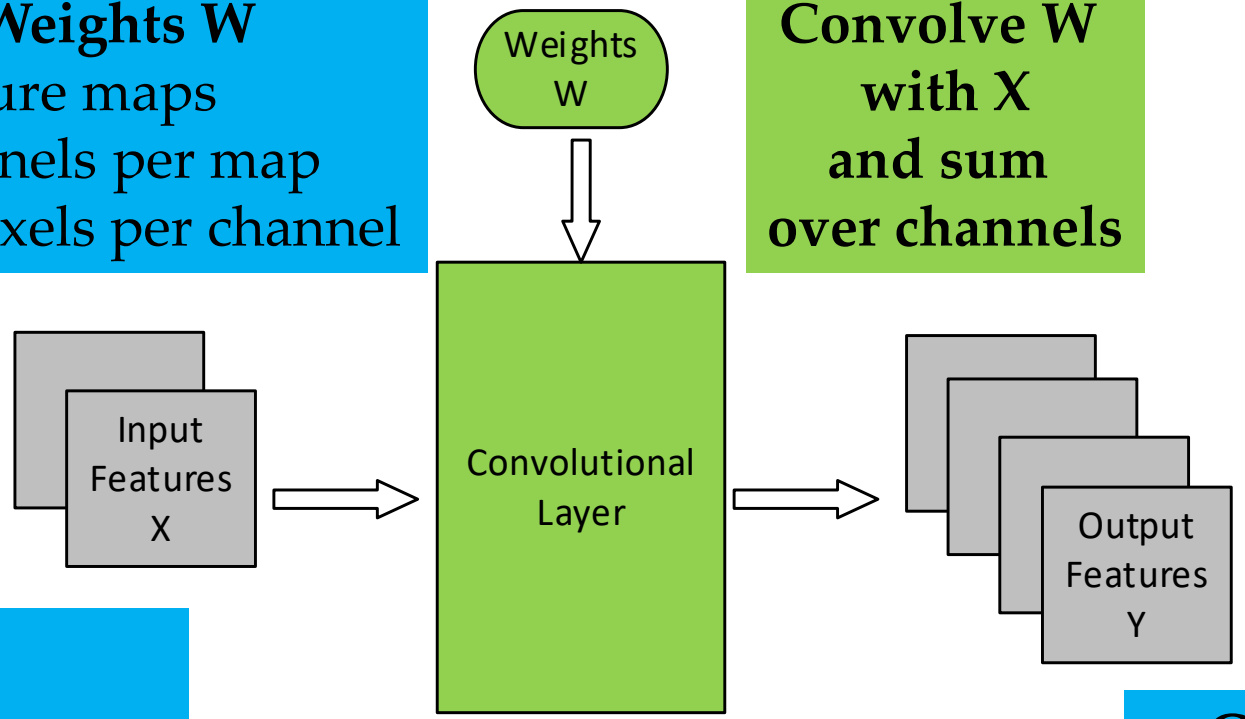
LeNet-5:CNN for hand-written digit recognition



Forward Propagation

Output Size
$H_{out} = H - K + 1$
$W_{out} = W - K + 1$

Weights W
M feature maps
C channels per map
K×K pixels per channel

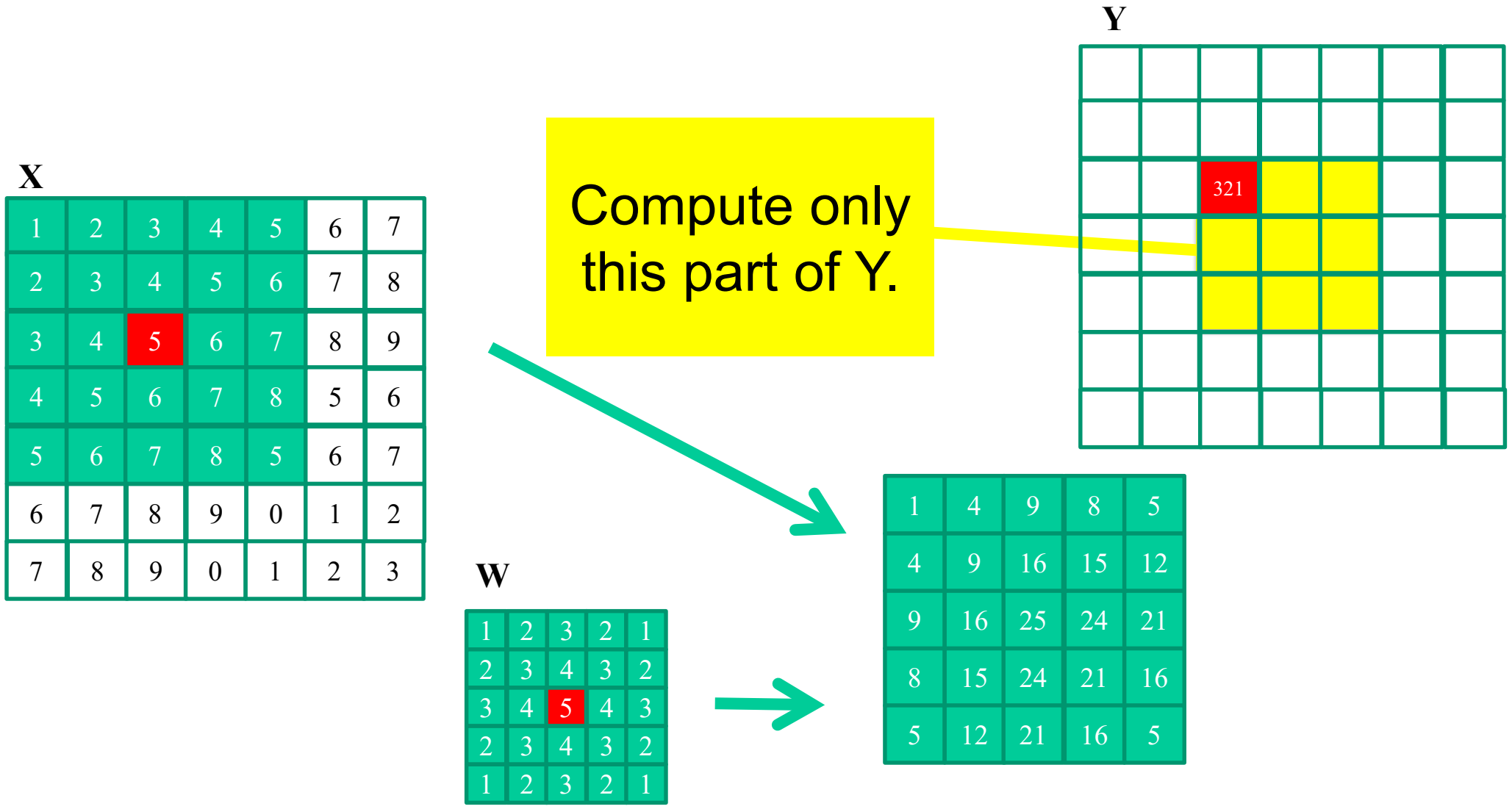


Convolve W with X and sum over channels

Input X
B images
C channels per image
H×W pixels per channel

Convolution Output Y
B images
M features per image
$H_{out} \times W_{out}$ values per feature

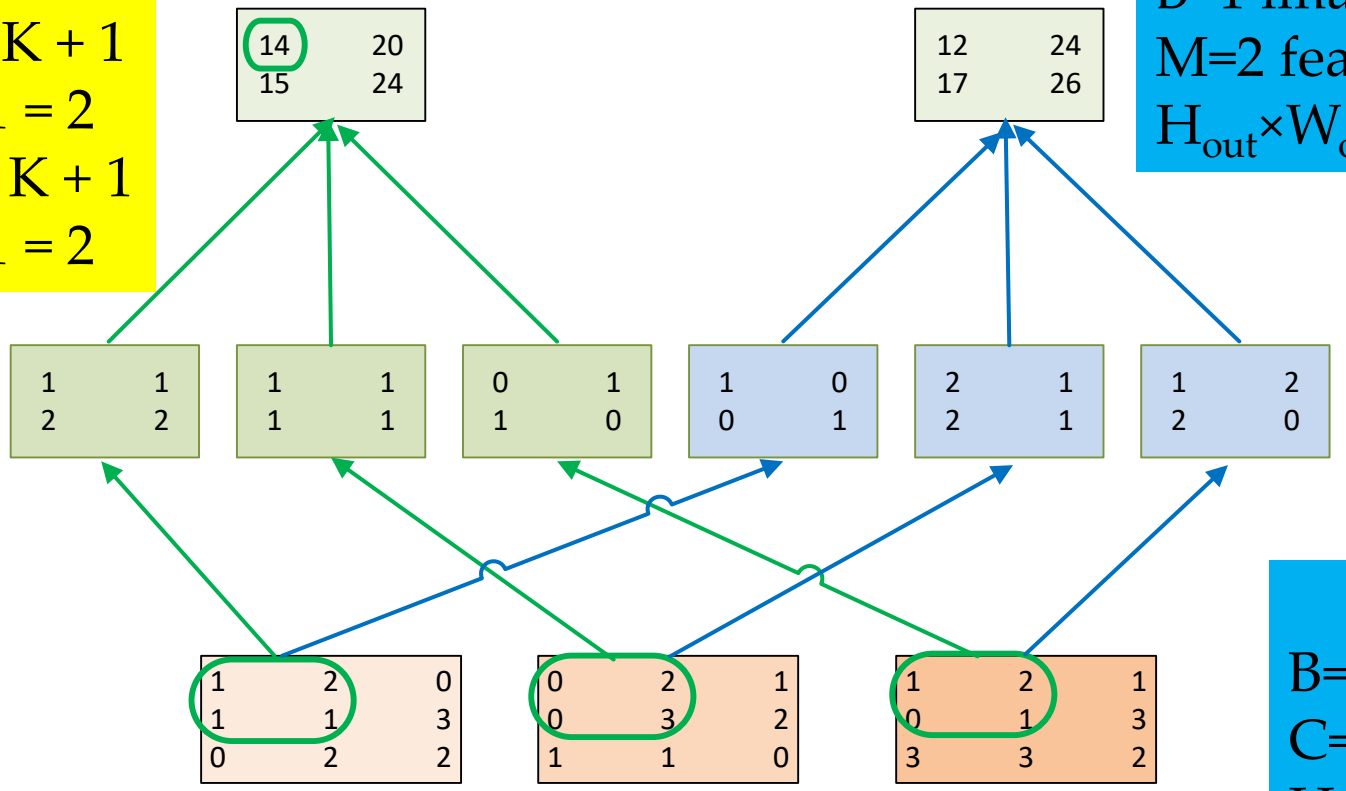
Outputs Must Use Full Mask/Kernel



Example of the Forward Path of a Convolution Layer

Output Size

$H_{out} = H - K + 1$
$= 3 - 2 + 1 = 2$
$W_{out} = W - K + 1$
$= 3 - 2 + 1 = 2$



Convolution Output Y

B=1 image
M=2 features per image
$H_{out} \times W_{out}$=2×2 values per feature

Weights W

M=2 feature maps
C=3 channels per map
K×K=2×2 pixels per channel

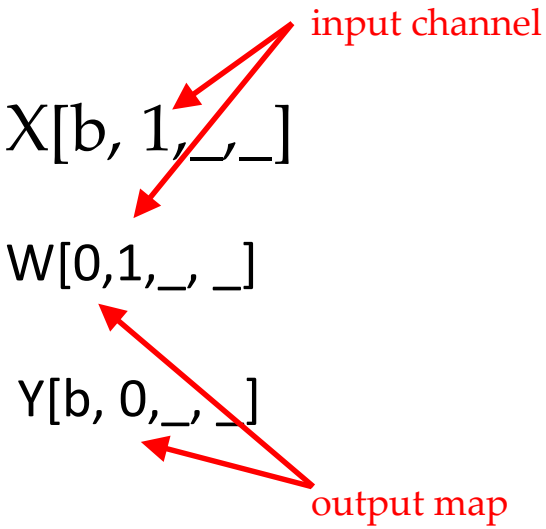
Input X

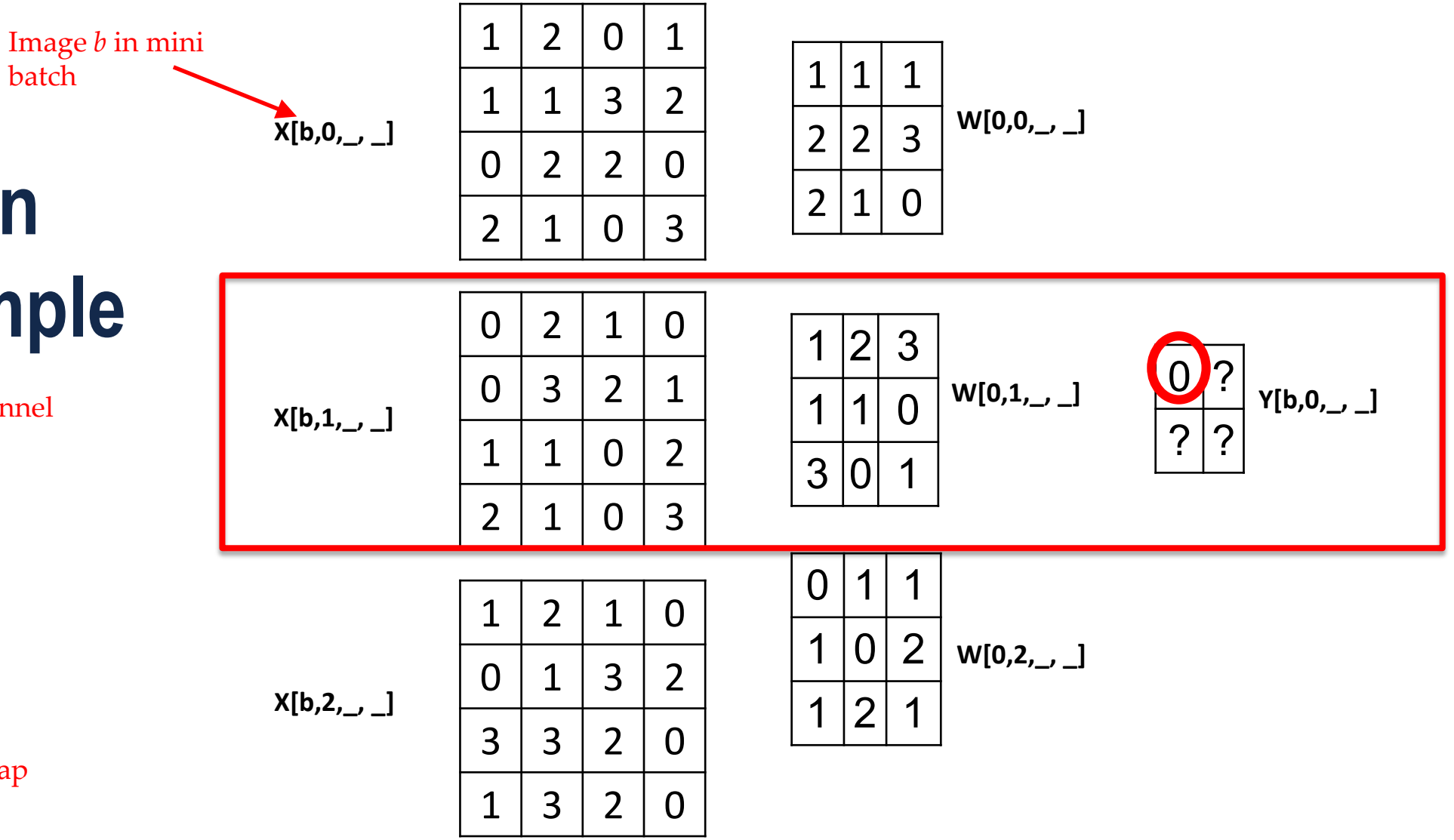
B=1 image
C=3 channels
H×W=3×3 pixels per channel

Sequential Code: Forward Convolutional Layer

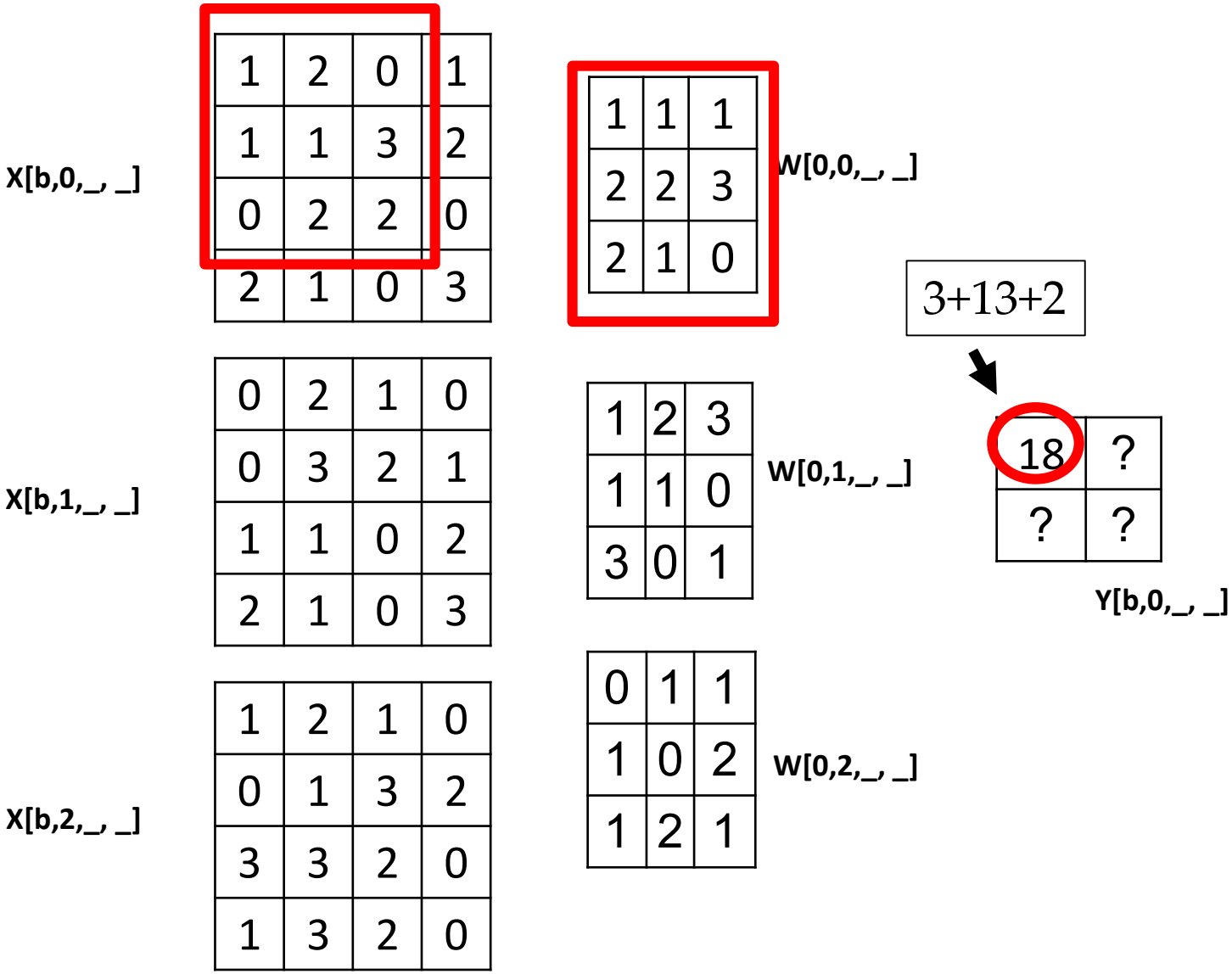
```
void convLayer_forward(int B, int M, int C, int H, int W, int K, float* X, float* W, float* Y) {
  int H_out = H - K + 1;                        // calculate H_out, W_out
  int W_out = W - K + 1;

  for (int b = 0; b < B; ++b)                   // for each image
    for(int m = 0;  m < M;  m++)                // for each output feature map
      for(int h = 0; h < H_out; h++)            // for each output value (two loops)
        for(int w = 0; w < W_out; w++) {
          Y[b, m, h, w] = 0.0f;                 // initialize sum to 0
          for(int c = 0; c < C; c++)            // sum over all input channels
            for(int p = 0; p < K; p++)            // KxK filter
              for(int q = 0; q < K; q++)
                Y[b, m, h, w] += X[b, c, h + p, w + q] * W[m, c, p, q];
        }
}
```

A Small Convolution Layer Example

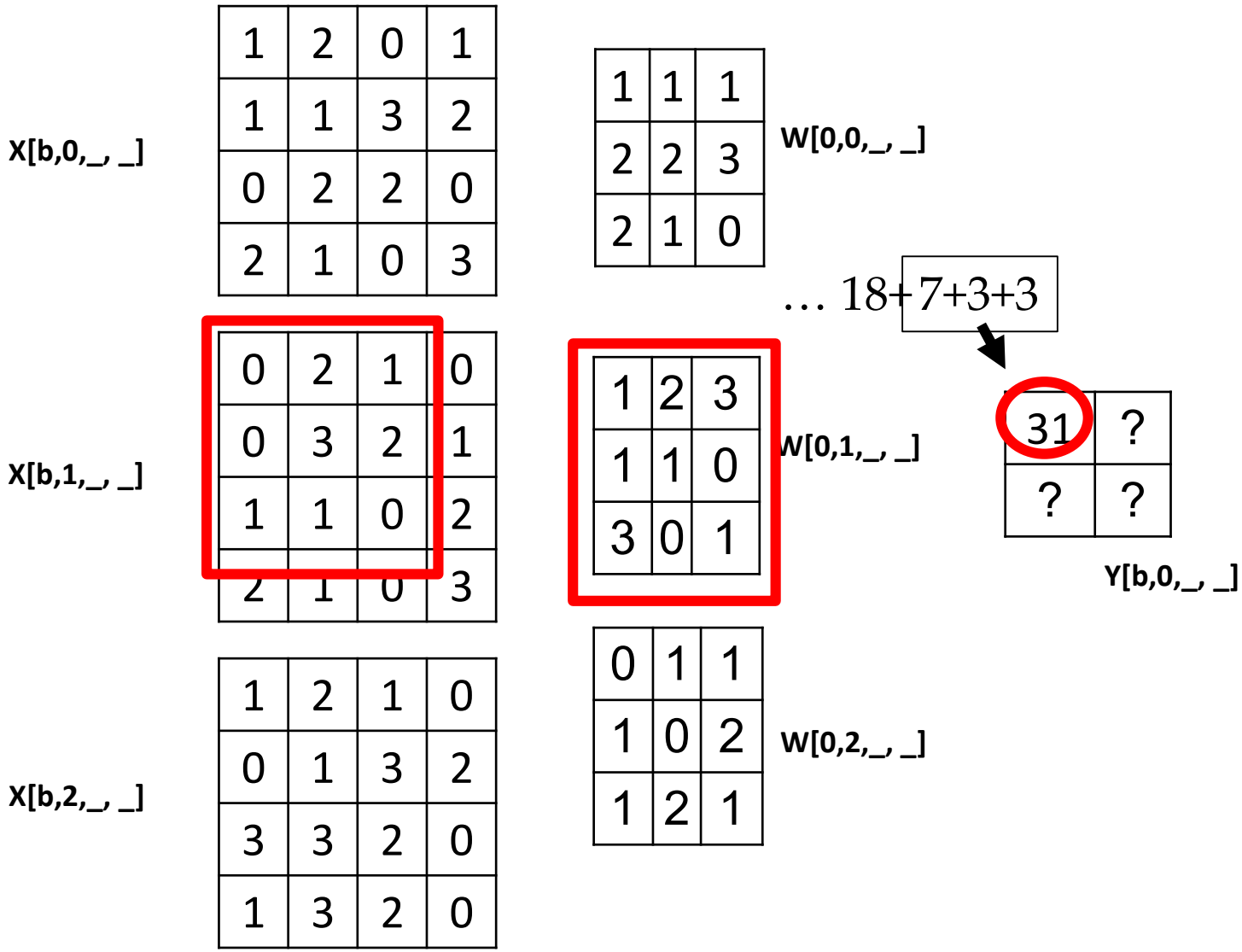




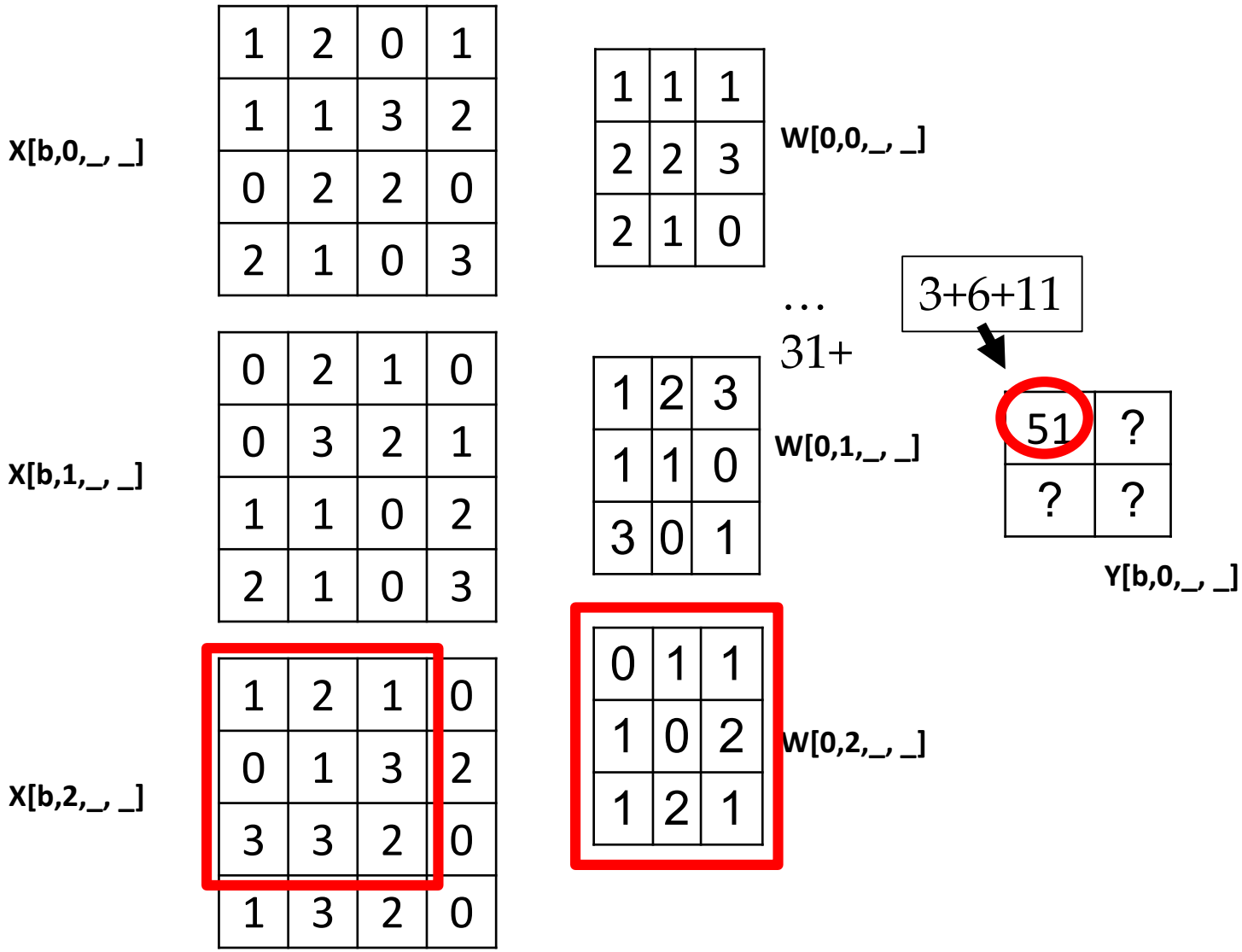
A Small Convolution Layer Example c = 0



A Small Convolution Layer Example c = 1



A Small Convolution Layer Example c = 2



Parallelism in a Convolution Layer

Output feature maps can be calculated in parallel

- Usually a small number, not sufficient to fully utilize a GPU

All **output** feature map **pixels** can be calculated in parallel

- All rows can be done in parallel
- All pixels in each row can be done in parallel
- Large number but diminishes as we go into deeper layers

All **input feature maps** can be processed in parallel,
but need atomic operation or tree reduction (we'll learn later)

Different layers may demand different strategies.

Subsampling (Pooling) by Scale N

Convolution Output Y
B images
M features per image
$H_{out} \times W_{out}$ values per feature

Average over N×N blocks, then calculate sigmoid

Output Size
$H_{S(N)} = \text{floor}\ (H_{out} / N)$
$W_{S(N)} = \text{floor}\ (W_{out} / N)$

Subsampling/Pooling Output S
B images
M features per image
$H_{S(N)} \times W_{S(N)}$ values per feature

Sequential Code: Forward Pooling Layer

```
void poolingLayer_forward(int B, int M, int H_out, int W_out, int N, float* Y, float* S)
{
  for (int b = 0; b < B; ++b)                  // for each image
    for (int m = 0;  m < M; ++m)               // for each output feature map
      for (int x = 0; x < H_out/N; ++x)        // for each output value (two loops)
        for (int y = 0; y < W_out/N; ++y) {
          float acc = 0.0f                     // initialize sum to 0
          for (int p = 0; p < N; ++p)          // loop over NxN block of Y (two loops)
             for (int q = 0; q < N; ++q)
                acc += Y[b, m, N*x + p, N*y + q];
          acc /= N * N;                        // calculate average over block
          S[b, m, x, y] = sigmoid(acc + bias[m])  // bias, non-linearity
        }
}
```

Kernel Implementation of Subsampling Layer

- Straightforward mapping from grid to subsampled output feature map pixels
- in GPU kernel,
  - need to manipulate index mapping
  - for accessing the output feature map pixels
  - of the previous convolution layer.
- Often merged into the previous convolution layer to save memory bandwidth

ANY MORE QUESTIONS?
READ CHAPTER 16