차세대 반도체 디지털 혁신공유대학 사업단

POLARIS

Deep Learning Hardware 설계 경진대회
YOLO network, reference software

2022.02.10 (Thu)

Road map

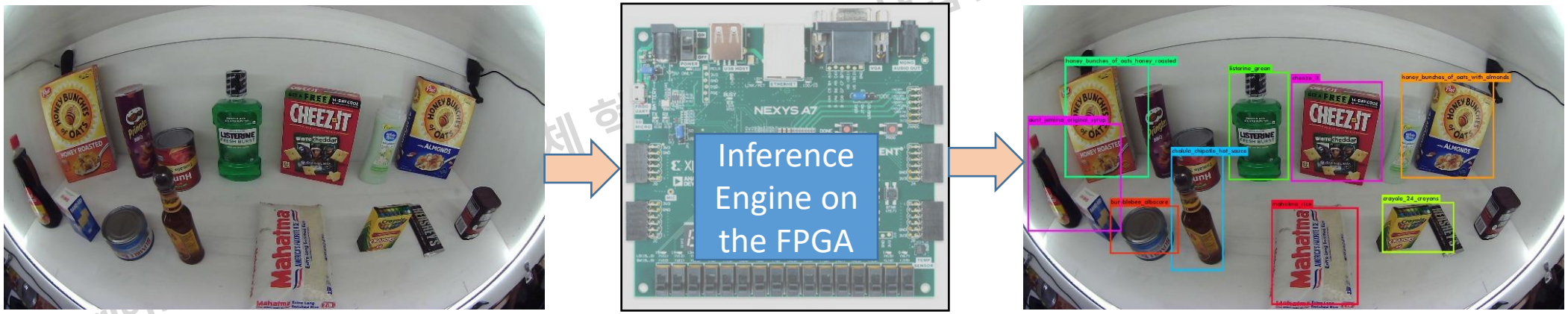
Review

YOLO network

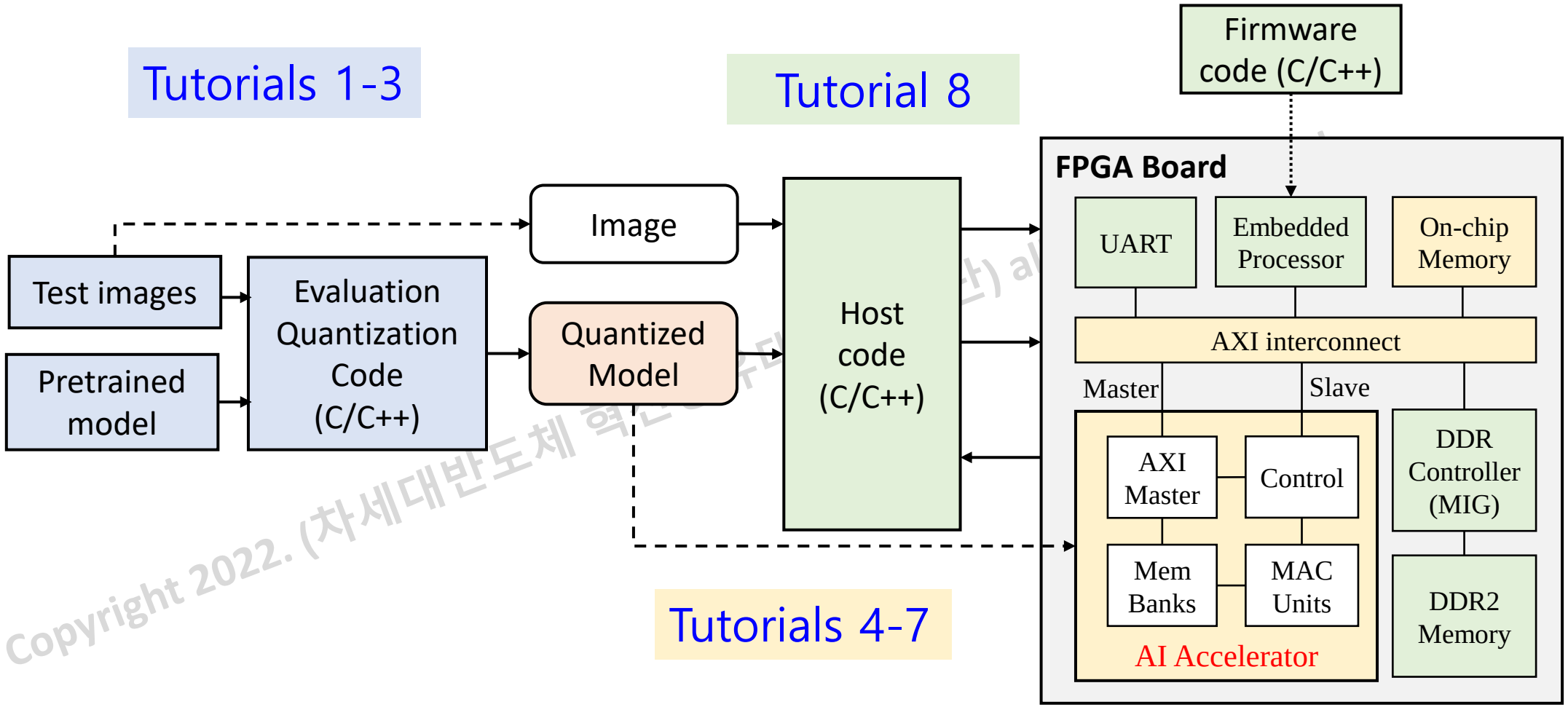
Reference S/W

설계의 목표

무인판매대에서 상품 인식을 위하는 딥러닝 추론 하드웨어를 설계한다.

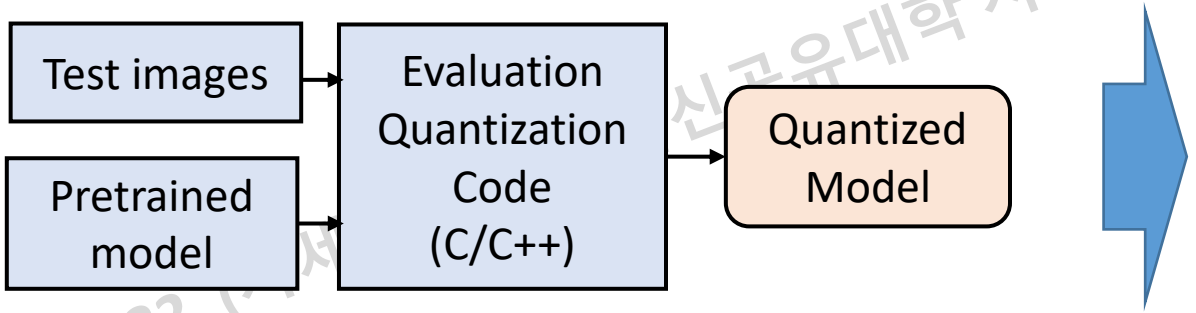


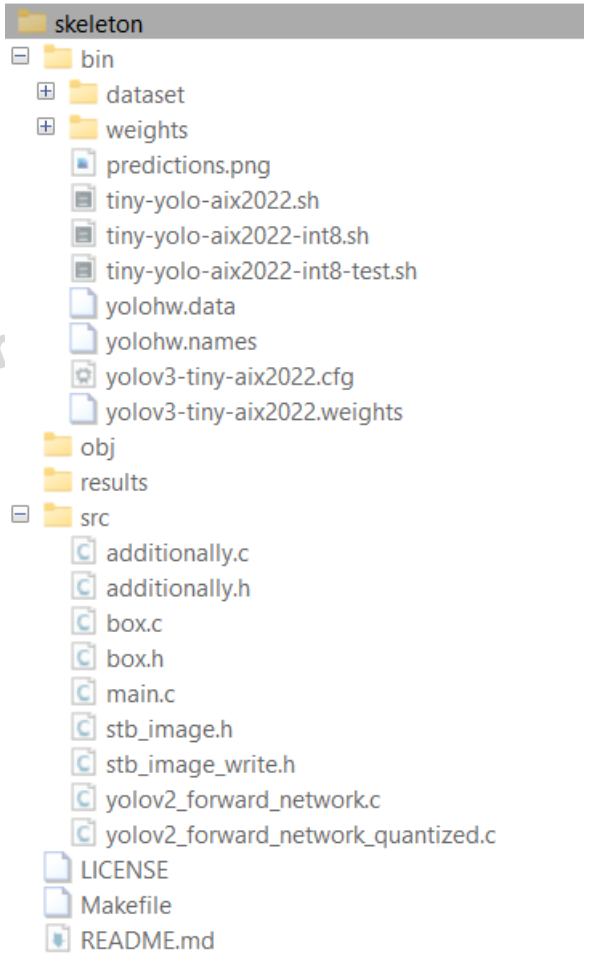
Top structure and tutorials



Tutorials 1-3: Reference S/W

- Skeleton
  - Test images, object classes, and ground truth
  - The pretrained model (tiny-yolov3)
  - Reference code for evaluation and quantization
  - The S/W code is based on the darknet code





Test images (skeleton/bin/dataset)

- 150 test images in three categories
  - Long: distances among objects are far
  - Close: items are close
  - Color: change light conditions
- Ground truth: 150 ".txt" files
  - Each line represents a labelled object
    - [class_index x_pos y_pos width height]
- yolohw.names: names of all 60 classes of products

```
12 0.19973958333333333 0.39166666666666666 0.0671875 0.2777777777777778
13 0.26484375 0.39351851851851855 0.1265625 0.2796296296296296
11 0.3419270833333333 0.38472222222222224 0.06302083333333333 0.287962962962963
6 0.4427083333333333 0.33055555555555555 0.178125 0.4425925925925926
0 0.5614583333333333 0.3842592592592593 0.090625 0.3388888888888889
9 0.6377604166666666 0.3773148148148148 0.09010416666666667 0.325
7 0.7296875 0.3699074074074074 0.14895833333333333 0.34351851851851856
1 0.19817708333333334 0.649537037037037 0.08072916666666667 0.18055555555555555
2 0.2671875 0.7041666666666667 0.08645833333333333 0.1824074074074074
3 0.30859375 0.6421296296296296 0.0859375 0.3212962962962963
4 0.3890625 0.7495370370370371 0.090625 0.1787037037037037
14 0.45598958333333334 0.7847222222222222 0.0734375 0.23425925925925928
10 0.621875 0.7532407407407408 0.25625 0.23425925925925928
5 0.765625 0.6810185185185186 0.11041666666666666 0.21574074074074076
8 0.825 0.6587962962962963 0.09270833333333334 0.23240740740740742
```

long

close

color

Pretrained model (skeleton/bin)

- A pretrained model is defined by two files
  - yolov3-tiny-aix2022.cfg: Network's configuration
    - Input size
    - Training/testing options
    - Layer's settings
  - yolov3-tiny-aix2022.weights (3354 KB)
    - 32-bit floating point parameters

```
[net]
# Testing
batch=1
subdivisions=1
# Training
#batch=64
#subdivisions=2
width=320
height=320
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 50200
policy=steps
steps=40000,45000
scales=.1,.1

[convolutional]
batch_normalize=1
filters=16
size=3
stride=1
pad=1
activation=leaky

[maxpool]
```

Source files

- additionally.c *// Definitions of darknet functions used*
- additionally.h *// Declaration of darknet functions + additional functions for forward pass of yolo model*
- box.c
- box.h

  // For bounding boxes
- stb_image_write.h
- stb_image.h

  // For loading/writing images
- yolov2_forward_network.c *// Functions for forward pass of yolo network*
- **yolov2_forward_network_quantized.c** *// Functions for quantization, saving of the quantized model, and the forward pass of quantized yolo model*
- main.c *// The main functions*

You should mainly edit this file for quantization!

Objectives

- YOLO network architecture
  - Image format
  - Tiny-YOLO-v3
    - Convolutional layer
    - Batch normalization
    - Max pooling

- Code
  - Release an updated version for both UNIX and Windows
  - Flow
  - Evaluation (mAP)

Road map

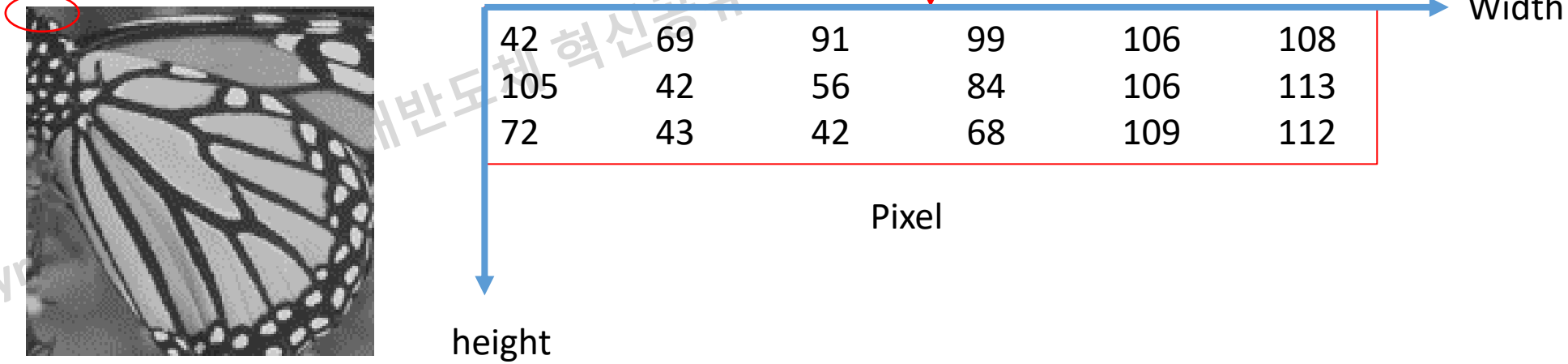
Review

YOLO network

Reference S/W

What is an image?

- An image is an artifact that depicts visual perception, such as a photograph or other two-dimensional picture, that resembles a subject—usually a physical object—and thus provides a depiction of it.
- In the context of signal processing, an image is a distributed amplitude of color(s)
- Example: Each pixel in a gray image is stored in an 8-bit format
  - Pixel value: {BLACK=0, 1, …, 255=WHITE}



Test image

- Each test image has **three color channels**: red, green and blue.
  - Each pixel
  - Image size: height=1080, width=1920, channel=3 → 6,220,800 (=1080×1920×3) bytes
  - Stored in a compressed format (jpeg or jpg) (402KB)

205	204	204	204	204	204	202	202
208	207	207	207	207	206	205	204
209	208	208	207	207	207	205	204
206	206	206	205	205	204	202	201

236	235	234	234	232	232	231	231
239	238	237	237	235	234	234	233
241	240	238	237	236	236	234	233
238	238	236	235	234	233	231	230

238	237	234	234	233	233	229	229
241	240	237	237	236	235	232	231
240	239	238	237	234	234	232	231
237	237	236	235	232	231	229	226

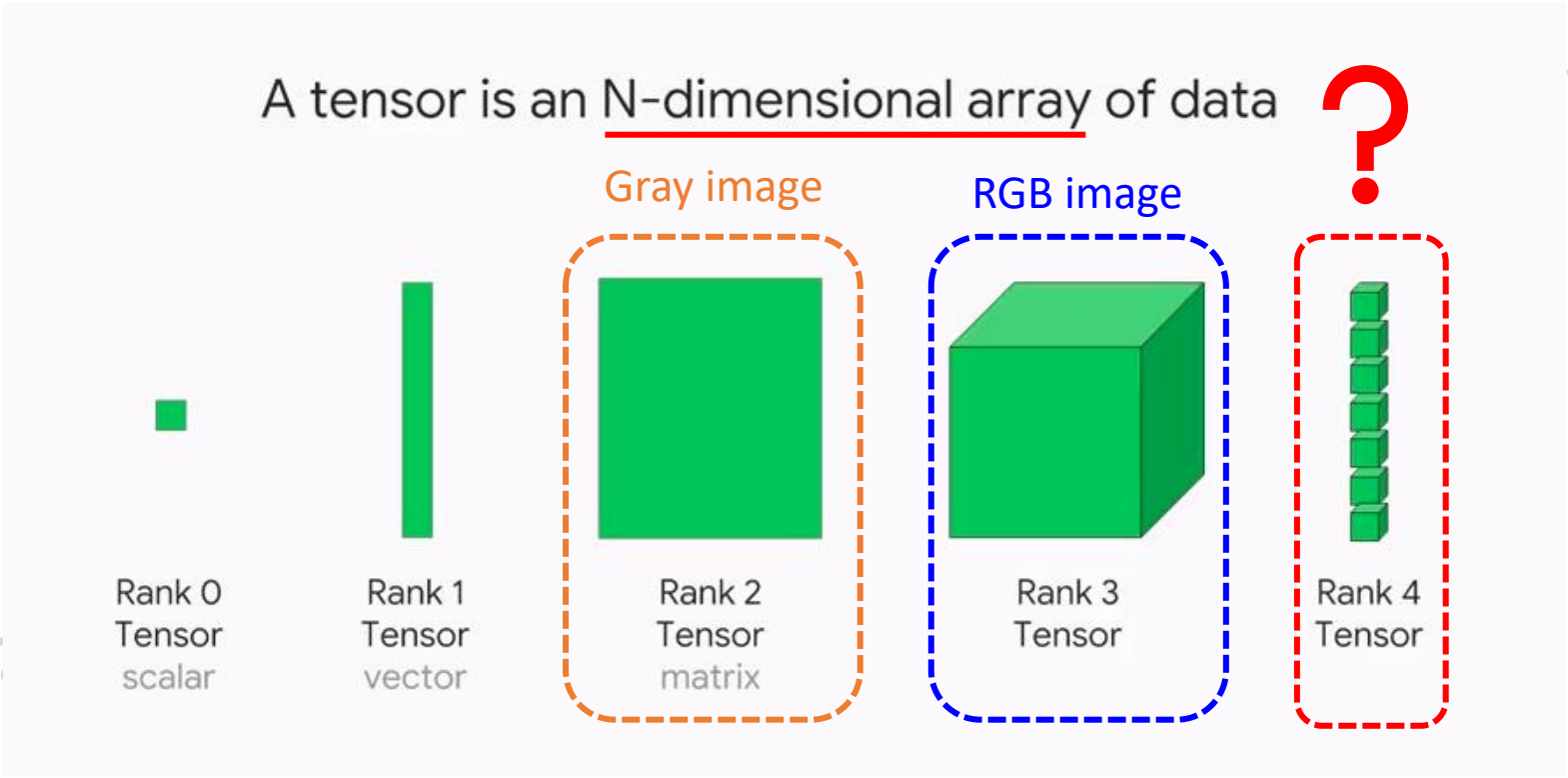
YOLO input

- How to make an input for a YOLO network?
  - An input image is rescaled in a square RGB image, i.e., width=height=320
  - Maintain the aspect ratios of objects after rescaling
    - Put a rescaled image at the center of a 320x320 image


```
[net]
# Testing
batch=1
subdivisions=1
# Training
#batch=64
#subdivisions=2
width=320
height=320
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 50200
policy=steps
steps=40000,45000
scales=.1,.1

[convolutional]
batch_normalize=1
filters=16
size=3
stride=1
pad=1
activation=leaky

[maxpool]
```

Tensor

- From Wikipedia[1] : "In mathematics, a tensor is an algebraic object that describes a multilinear relationship between sets of algebraic objects related to a vector space."



Source: https://medium.com/mlait/tensors-representation-of-data-in-neural-networks-bbe8a711b93b

[1] https://en.wikipedia.org/wiki/Tensor

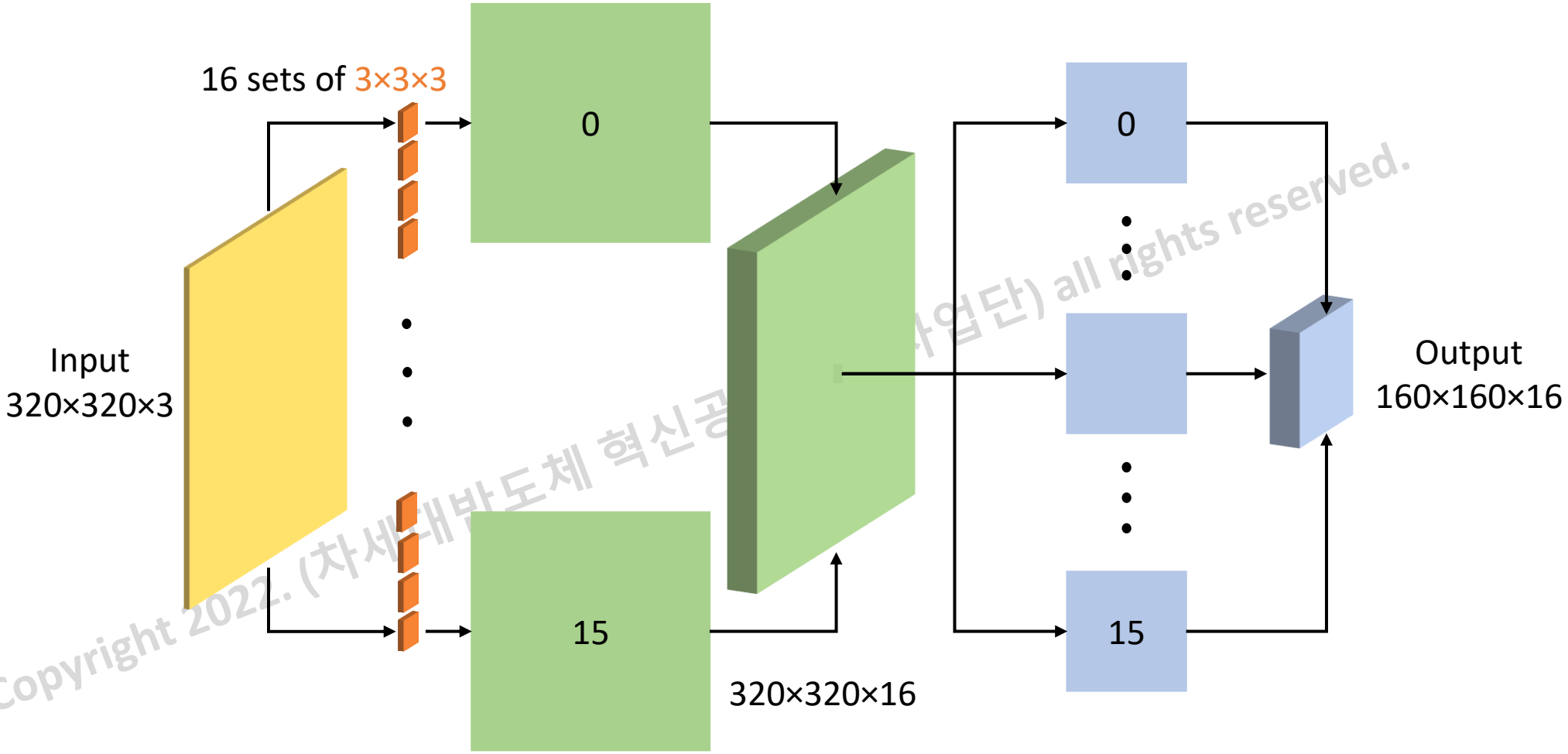
Tiny-YOLO-v3 (AIX)

- A pretrained model is defined by two files
  - yolov3-tiny-aix2022.cfg:
    - Network's configuration
  - yolov3-tiny-aix2022.weights (3354 KB)
    - 32-bit floating point parameters

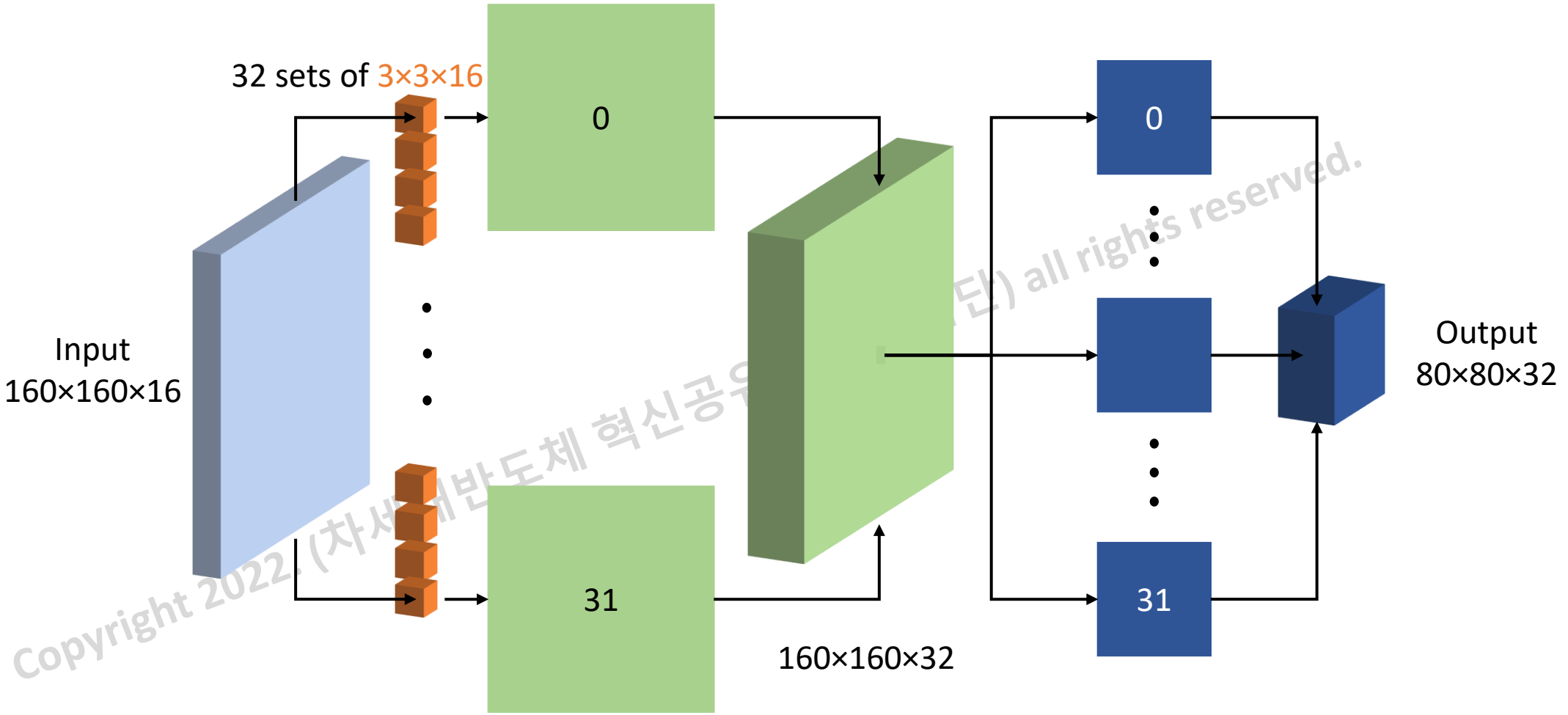
- Tiny-YOLOv3 inference
  - 22 layers
    - 11 convolutional layers
      - 3x3: eight layers, 1x1: three layers
    - 6 max pooling layer (one has stride = 1)
    - Route, upsample and yolo layers
  - Inputs: image (320x320x3) and filters
  - Outputs
    - Layer 13: 10x10x195, Layer 20: 20x20x195

layer	type	filter	input	output
0	conv	3x3x3x16	320x320x3	320x320x16
1	max		320x320x16	160x160x16
2	conv	3x3x16x32	160x160x16	160x160x32
3	max		160x160x32	80x80x32
4	conv	3x3x32x64	80x80x32	80x80x64
5	max		80x80x64	40x40x64
6	conv	3x3x64x128	40x40x64	40x40x128
7	max		40x40x128	20x20x128
8	conv	3x3x128x128	20x20x128	20x20x128
9	max		20x20x128	10x10x128
10	conv	3x3x128x128	10x10x128	10x10x128
11	max		10x10x128	10x10x128
12	conv	3x3x128x128	10x10x128	10x10x128
13	conv	1x1x128x195	10x10x128	10x10x195
14	yolo			
15	route	12		10x10x128
16	conv	1x1x128x128	10x10x128	10x10x128
17	upsample		10x10x128	20x20x128
18	route	17,8		20x20x128
19	conv	3x3x128x128	20x20x256	20x20x128
20	conv	1x1x128x195	20x20x128	20x20x195
21	yolo			

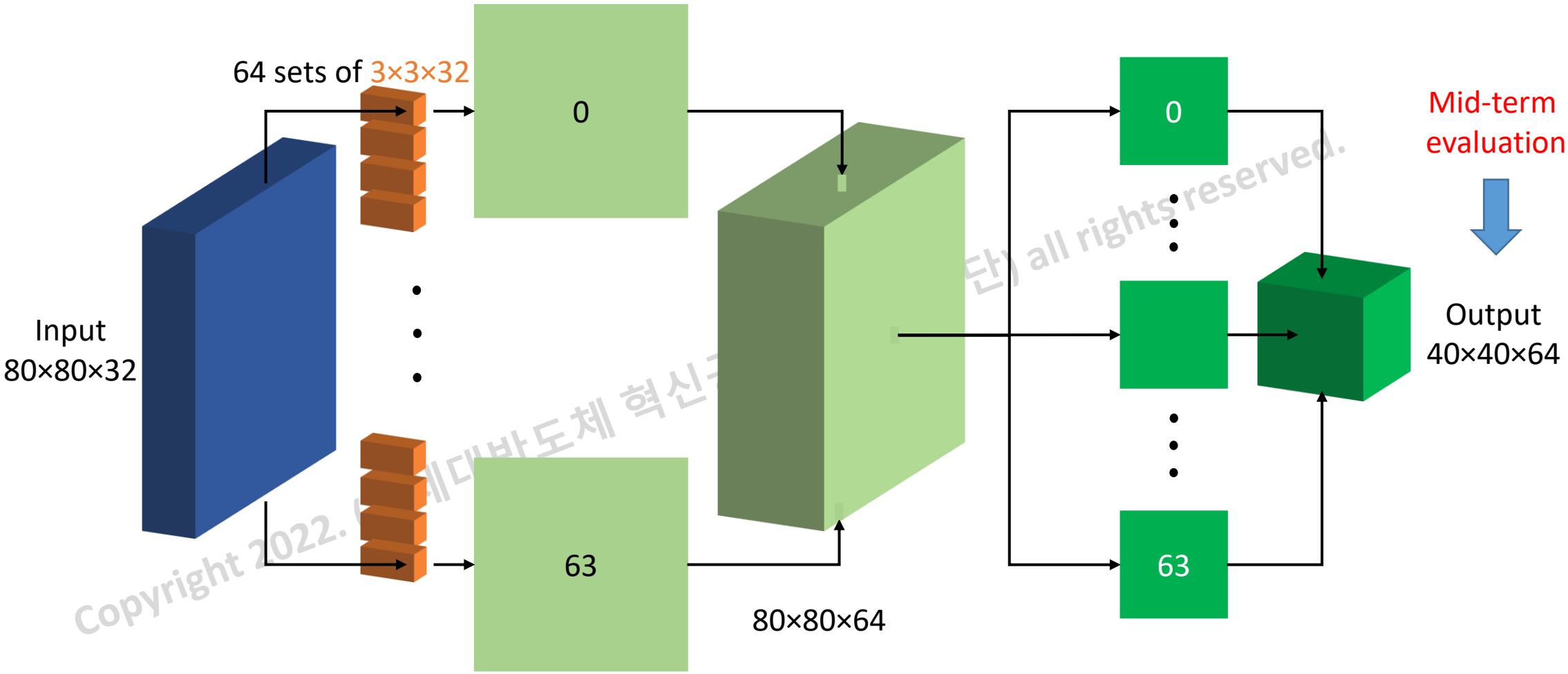
Convolution + max-pooling: Layers 0, 1



Convolution + max-pooling: Layers 2, 3

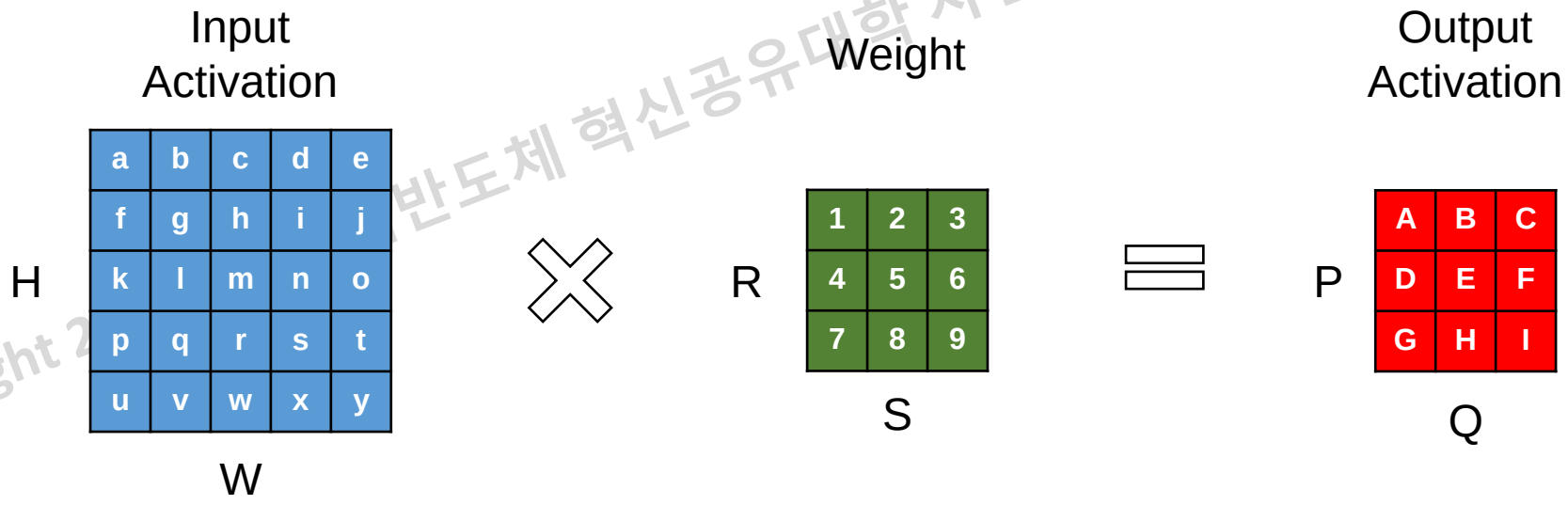


Convolution + max-pooling: Layers 4, 5



2D convolution

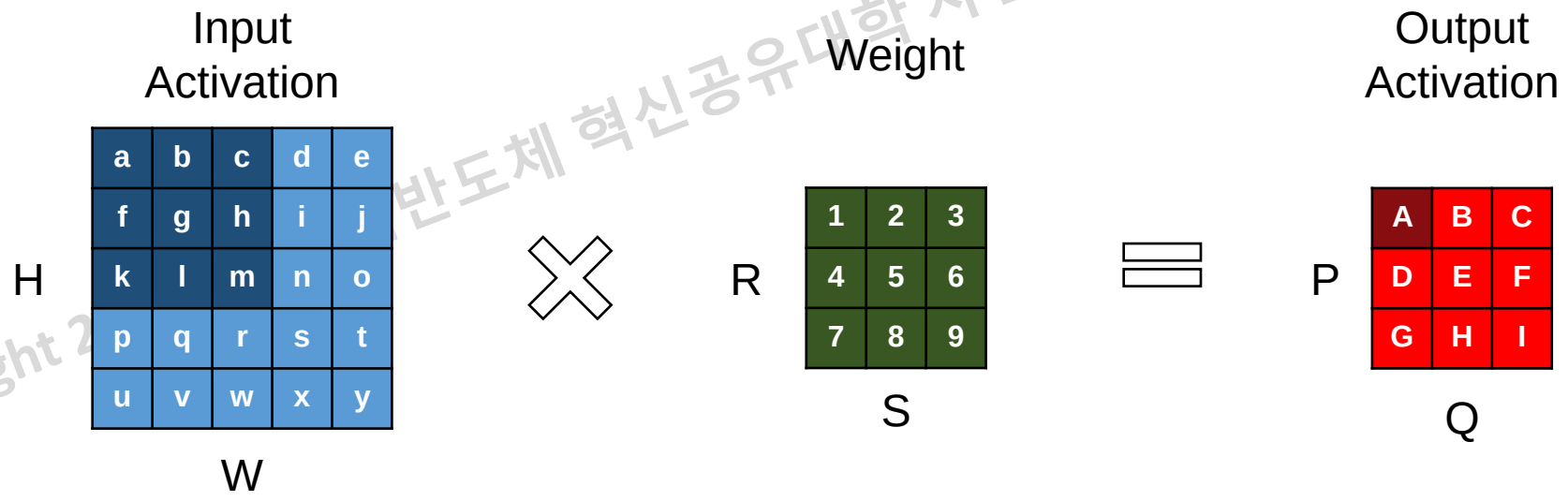
- 2D convolution:
  - Calculate output activation (OA) from input activation (IA) and weights (W)



2D convolution

- 2D convolution:
  - Calculate output activation (OA) from input activation (IA) and weights (W)

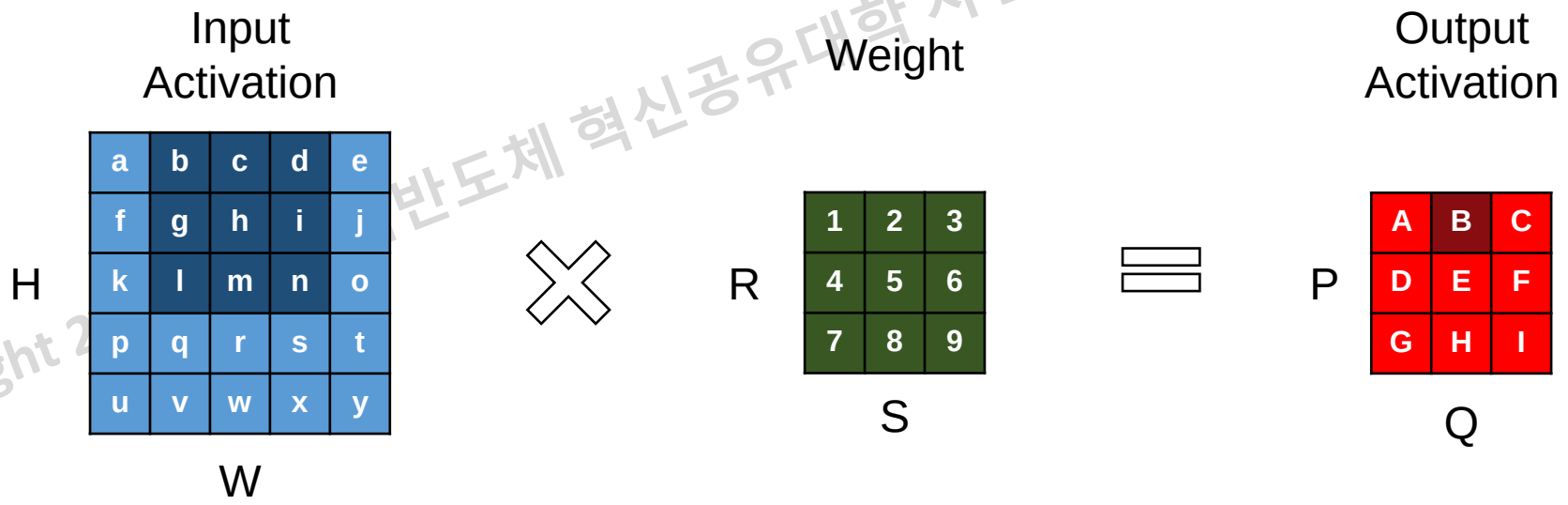
$$A = a * 1 + b * 2 + c * 3$$
$$+ f * 4 + g * 5 + h * 6$$
$$+ k * 7 + l * 8 + m * 9$$



2D convolution

- 2D convolution:
  - Calculate output activation (OA) from input activation (IA) and weights (W)

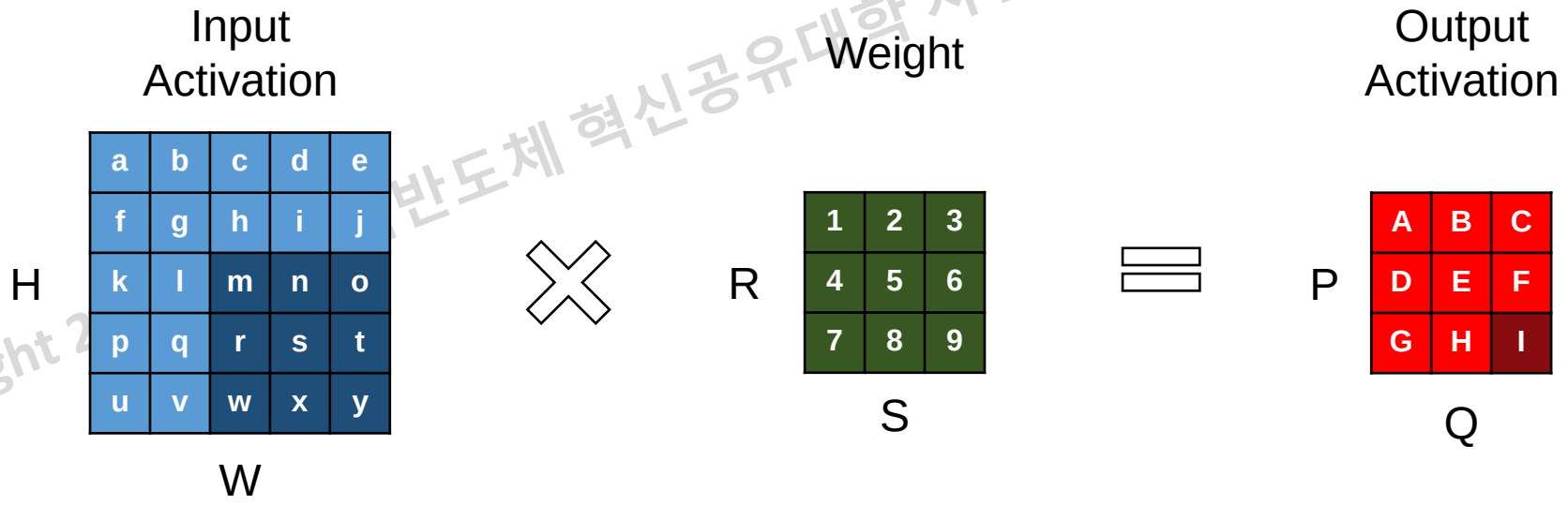
$$B = b * 1 + c * 2 + d * 3 + g * 4 + h * 5 + i * 6 + l * 7 + m * 8 + n * 9$$



2D convolution

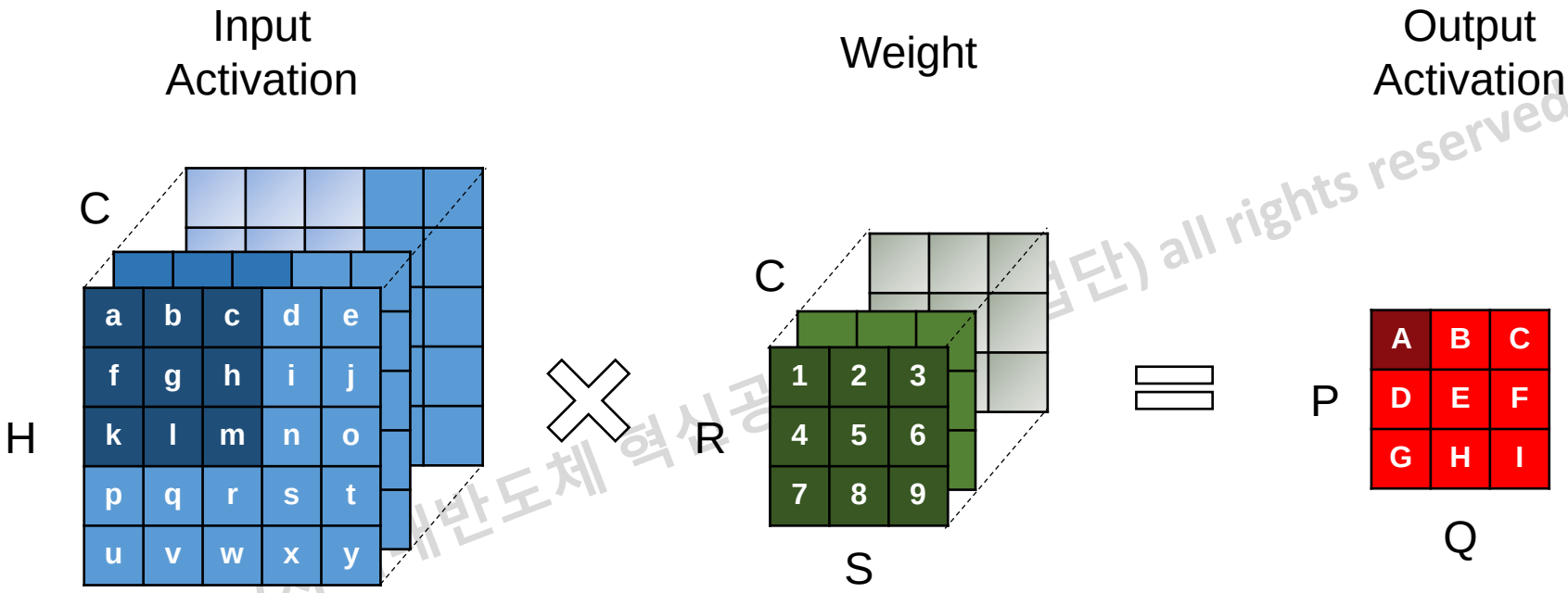
- 2D convolution:
  - Calculate output activation (OA) from input activation (IA) and weights (W)

$$I = m * 1 + n * 2 + o * 3$$
$$+r * 4 + s * 5 + t * 6$$
$$+w * 7 + x * 8 + y * 9$$

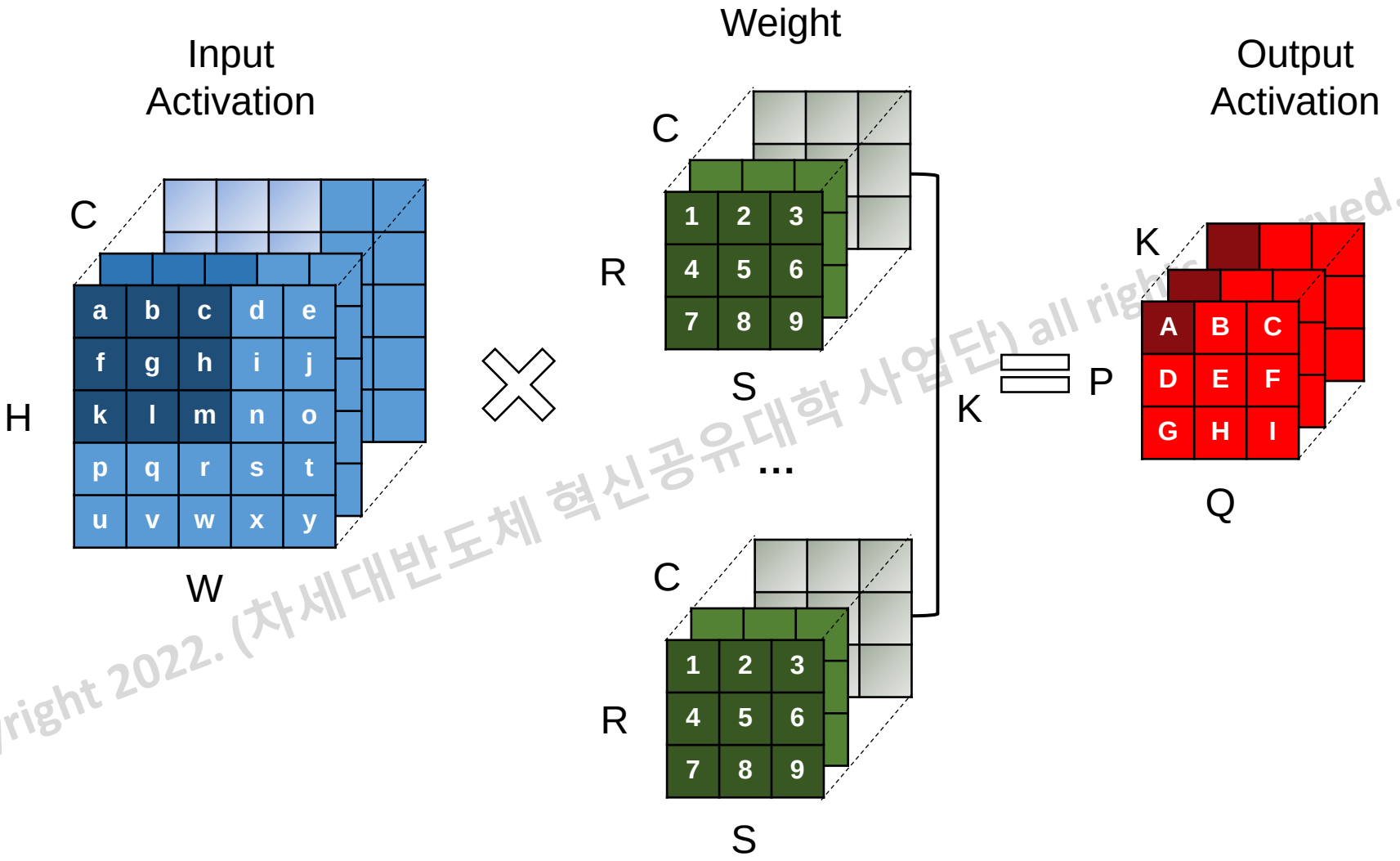


3-D Convolution: Multiple input channels

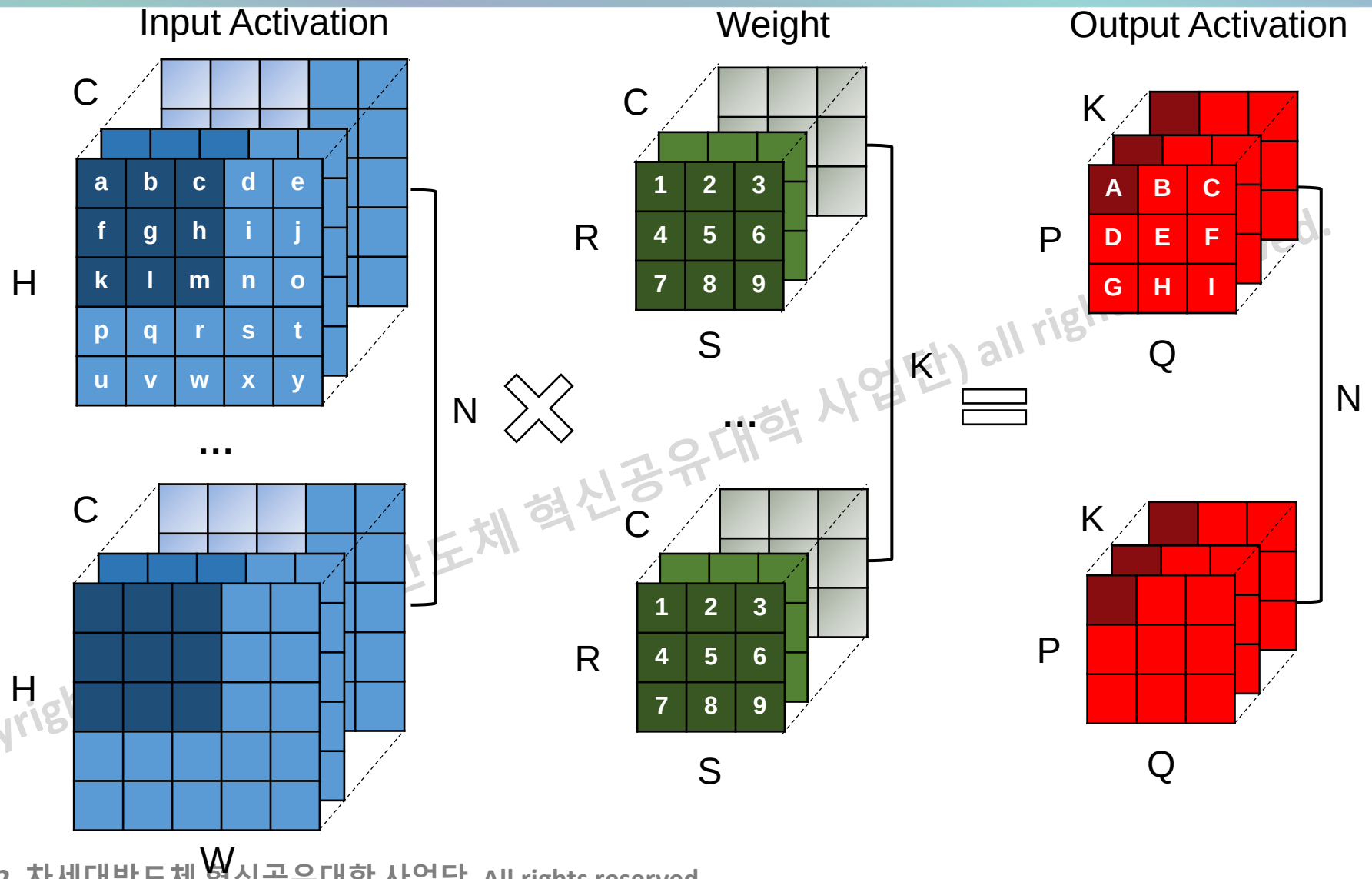
- **C:** # of Input Channels



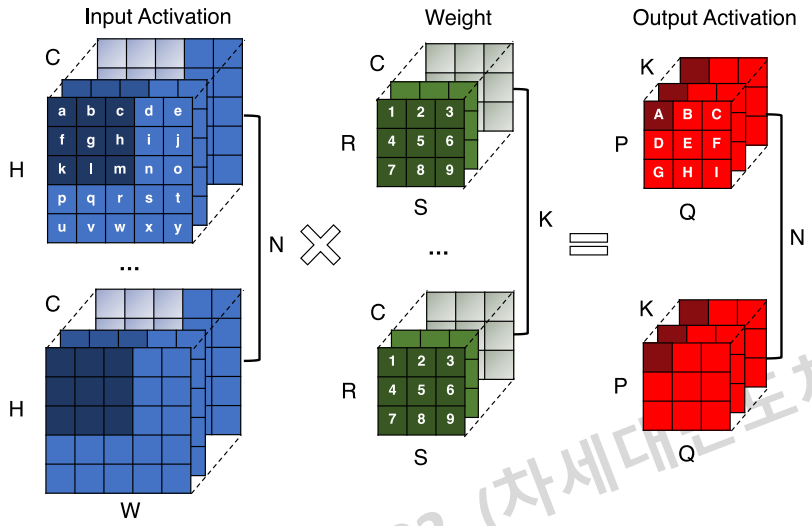
3-D Convolution: multiple output channels



Batch: Multiple inputs



CONV loop




```
for (n=0; n<N; n++) {
  for (k=0; k<K; k++) {
    for (p=0; p<P; p++) {
      for (q=0; q<Q; q++) {
        OA[n][k][p][q]= 0;
        for (r=0; r<R; r++) {
          for (s=0; s<S; s++) {
            for (c=0; c<C; c++) {
              h = p * stride - pad + r;
              w = q * stride - pad + s;
              OA[n][k][p][q] += IA[n][c][h][w] * W[k][c][r][s];
            }
          }
        }
        OA[n][k][p][q]= Activation(OA[n][k][p][q]);
      }
    }
  }
}
```

for each output activation

Convolution window

Normalization, Batch normalization

- Two methods are usually used for scaling or normalizing data
  - Scale data all numeric variables to the range [0, 1]:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

  - Scale data to have zero mean (μ) and unit variance (σ):

$$x_{new} = \frac{x - \mu}{\sigma}, \quad or \quad x_{new} = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon^2}} + \beta$$

  - In the neural network community, this is called *Whitening*

- *Batch normalization essentially performs Whitening to the intermediate layers of Neural Networks*
  - Reduce covariance shift
  - Reduce effects of exploding and vanishing gradients
- For each layer, mean (μ), variance (σ) and scale (γ)are given.

Ioffe, Sergey; Szegedy, Christian (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", in *Proceedings of the 32nd International Conference on International Conference on Machine Learning* (ICML'15), Vol. 37, July 2015 pp. 448–456

Batch norm folding: Speed up inference

Before

$$y_i = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}} x_i - \frac{\gamma}{\sqrt{\sigma^2+\epsilon}} \mu + \beta \quad (1)$$

$$x_i = W.z_{ci} + b \quad (2)$$

Convolution

Need divider operations

$$y_i = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}} (W.z_{ci} + b) - \frac{\gamma}{\sqrt{\sigma^2+\epsilon}} \mu + \beta \quad (3)$$

After

$$y_i = W'.z_{ci} + b' \quad (4)$$

New convolution

$$W' = W \frac{\gamma}{\sqrt{\sigma^2+\epsilon}} \quad (5)$$

$$b' = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}} (b - \mu) + \beta \quad (6)$$

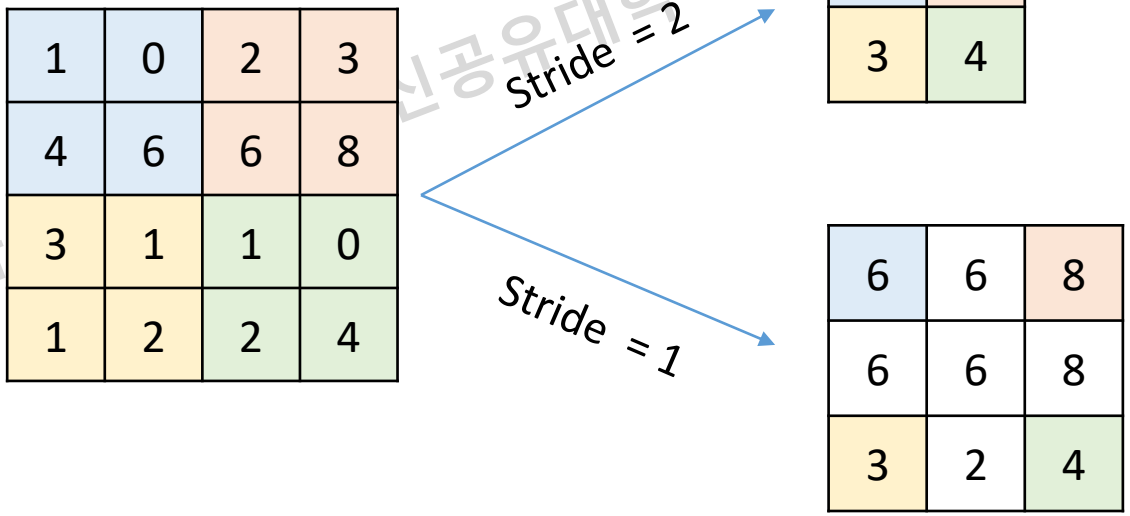
Pre-calculate (offline)

One multiplier and one adder

Source: https://scortex.io/batch-norm-folding-an-easy-way-to-improve-your-network-speed/

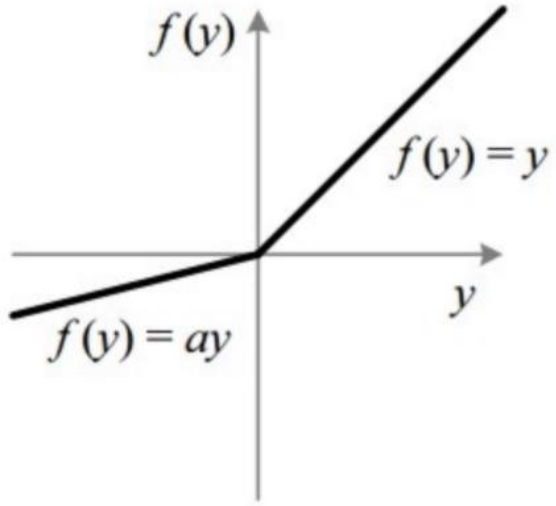
Max pooling

- Kernel size 2x2: Find the maximum value of all elements in a kernel
  - Reduce the spatial dimensions of tensors
  - Stride = 2: Layers 1, 3, 5, 7, 9.
    - Reduce width and height by 2x
  - Stride = 1: Layer 11



Activation leaky ReLU

- **Leaky Rectified Linear Unit**, or **Leaky ReLU**, is a type of activation function based on a ReLU, but it has a small slope for negative values instead of a flat slope.
- The slope coefficient is determined before training, i.e. it is NOT learnt during training.
- This type of activation function is popular in tasks where we may suffer from sparse gradients



Road map

Review

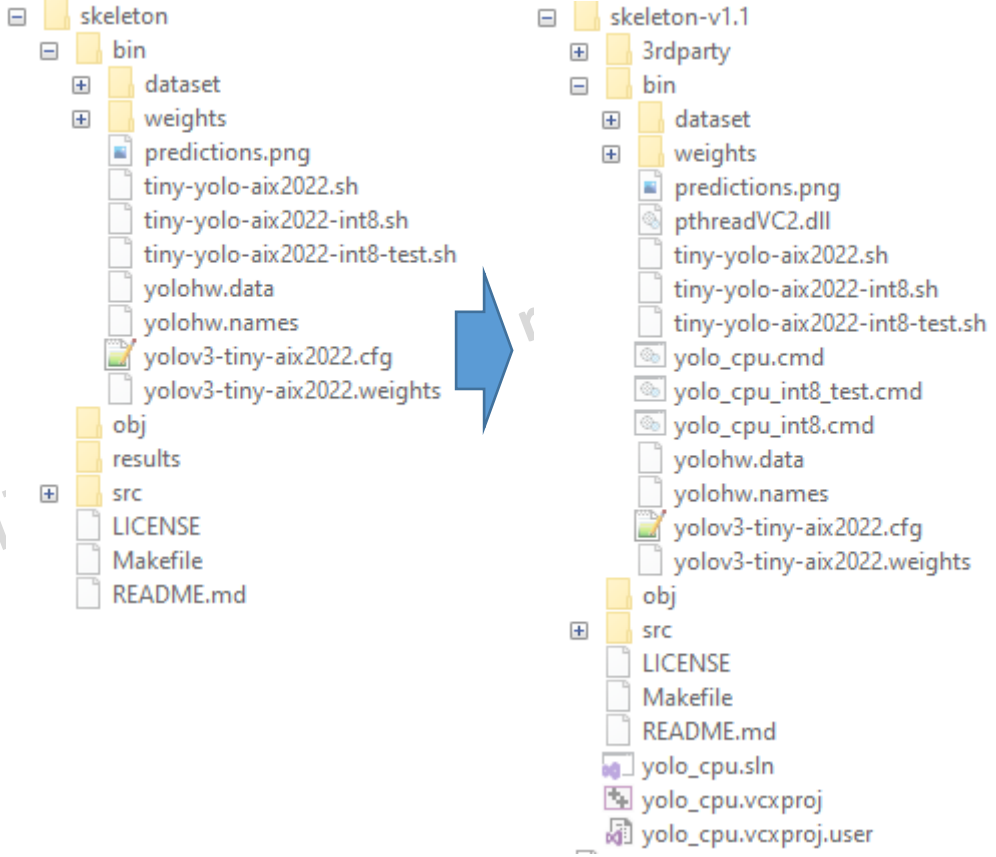
YOLO network

Reference S/W

skeleton v1.1

- We release skeleton v1.1 which supports Windows users.
  - No change in src/, bin/data/set and the model (yolov3-tiny-aix2022.cfg, and yolov3-tiny-aix2022.weights).

- Changes
  - Add 3rdparty/
  - Add Visual studio (VS) project files:
    - yolo_cpu.sln, yolo_cpu.vcxproj, and yolo_cpu.vcxproj.user
  - Add example scripts
    - bin/yolo_cpu.cmd
    - bin/yolo_cpu_int8.cmd
    - bin/yolo_cpu_int7_test.cmd



How to compile and run: UNIX

- After decompress the code, you can use the following commands

```
cd skeleton-v1.1                // Direct to the code folder
make                            // Compile the code
cd bin/dataset                  // Go to the bin/dataset/
python make_list_cur.py         // Update the directories for test images
cd ..                           // Go to bin/
sh tiny-yolo-aix2022.sh         // Do inference on all test images, and calculate mAP
```

How to compile and run: UNIX

- Compile the code
  - Ignore warnings
  - The objects files are store in obj/
  - The executable file is bin/darknet

```
gcc  -Wall -Wfatal-errors -Ofast -c ./src/yolov2_forward_network_quantized.c -o obj/yolov2_forward_network_quantized.o
./src/yolov2_forward_network_quantized.c: In function 'gemm_nn_int8_int32':
./src/yolov2_forward_network_quantized.c:11:32: warning: integer overflow in expression [-Woverflow]
 #define MAX_VAL_32 (256*256*256*256/2 - 1) // 31-bit (1-bit sign)
                                ^
./src/yolov2_forward_network_quantized.c:104:47: note: in expansion of macro 'MAX_VAL_32'
            C[i*ldc + j] += max_abs(c_tmp[j], MAX_VAL_32);
                                              ^~~~~~~~~~
./src/yolov2_forward_network_quantized.c: In function 'forward_convolutional_layer_q':
./src/yolov2_forward_network_quantized.c:140:0: warning: ignoring #pragma omp parallel [-Wunknown-pragmas]
     #pragma omp parallel for

./src/yolov2_forward_network_quantized.c: In function 'do_quantization':
./src/yolov2_forward_network_quantized.c:243:26: warning: unused variable 'weights_size' [-Wunused-variable]
            size_t const weights_size = l->size*l->size*l->c*l->n;
                         ^~~~~~~~~~~~
./src/yolov2_forward_network_quantized.c: In function 'save_quantized_model':
./src/yolov2_forward_network_quantized.c:287:26: warning: unused variable 'filter_size' [-Wunused-variable]
            size_t const filter_size = l->size*l->size*l->c;
                         ^~~~~~~~~~~
In file included from ./src/yolov2_forward_network_quantized.c:1:0:
At top level:
./src/additionally.h:130:17: warning: 'activate_array' defined but not used [-Wunused-function]
     static void activate_array(float *x, const int n, const ACTIVATION a)
                 ^~~~~~~~~~~~~~
gcc  -Wall -Wfatal-errors -Ofast obj/main.o obj/additionally.o obj/box.o obj/yolov2_forward_network.o obj/yolov2_forward_network_quantized.o -o bin/darknet -lm -pthread
(base) truongnx@marlin:~/aix2022/skeleton-v1.1$
```

How to compile and run: UNIX

- Loading a model, test images and do evaluation

  ./darknet detector map yolohw.names yolov3-tiny-aix2022.cfg yolov3-tiny-aix2022.weights -thresh 0.24

```
(base) truongnx@marlin:~/aix2022/skeleton-v1.1/bin$ ./tiny-yolo-aix2022.sh
valid: Using default 'dataset/target.txt'
names: Using default 'yolohw.names'
layer     filters    size              input                output
    0 conv     16  3 x 3 / 1   320 x 320 x   3   ->   320 x 320 x  16 0.088 BF
    1 max          2 x 2 / 2   320 x 320 x  16   ->   160 x 160 x  16
    2 conv     32  3 x 3 / 1   160 x 160 x  16   ->   160 x 160 x  32 0.236 BF
    3 max          2 x 2 / 2   160 x 160 x  32   ->    80 x  80 x  32
    4 conv     64  3 x 3 / 1    80 x  80 x  32   ->    80 x  80 x  64 0.236 BF
    5 max          2 x 2 / 2    80 x  80 x  64   ->    40 x  40 x  64
    6 conv    128  3 x 3 / 1    40 x  40 x  64   ->    40 x  40 x 128 0.236 BF
    7 max          2 x 2 / 2    40 x  40 x 128   ->    20 x  20 x 128
    8 conv    128  3 x 3 / 1    20 x  20 x 128   ->    20 x  20 x 128 0.118 BF
    9 max          2 x 2 / 2    20 x  20 x 128   ->    10 x  10 x 128
   10 conv    128  3 x 3 / 1    10 x  10 x 128   ->    10 x  10 x 128 0.029 BF
   11 max          2 x 2 / 1    10 x  10 x 128   ->    10 x  10 x 128
   12 conv    128  3 x 3 / 1    10 x  10 x 128   ->    10 x  10 x 128 0.029 BF
   13 conv    195  1 x 1 / 1    10 x  10 x 128   ->    10 x  10 x 195 0.005 BF
   14 yolo
   15 route  12
   16 conv    128  1 x 1 / 1    10 x  10 x 128   ->    10 x  10 x 128 0.003 BF
   17 upsample            2x    10 x  10 x 128   ->    20 x  20 x 128
   18 route  17 8
   19 conv    128  3 x 3 / 1    20 x  20 x 256   ->    20 x  20 x 128 0.236 BF
   20 conv    195  1 x 1 / 1    20 x  20 x 128   ->    20 x  20 x 195 0.020 BF
   21 yolo
Loading weights from yolov3-tiny-aix2022.weights...Done!
```

How to compile and run: UNIX

- Report the accuracy of all 60 items
- The 32-bit model achieves the mean average precision (mAP) is 83.20%.

```
class_id = 30, name = coffee_mate_french_vanilla,         ap = 97.68 %
class_id = 31, name = pepperidge_farm_milk_chocolate_macadamia_cookies,          ap = 71.63
class_id = 32, name = kitkat_king_size,           ap = 85.32 %
class_id = 33, name = snickers,           ap = 36.60 %
class_id = 34, name = toblerone_milk_chocolate,           ap = 97.04 %
class_id = 35, name = clif_z_bar_chocolate_chip,          ap = 98.70 %
class_id = 36, name = nature_valley_crunchy_oats_n_honey,         ap = 72.28 %
class_id = 37, name = ritz_crackers,      ap = 97.73 %
class_id = 38, name = palmolive_orange,           ap = 55.47 %
class_id = 39, name = crystal_hot_sauce,          ap = 100.00 %
class_id = 40, name = tapatio_hot_sauce,          ap = 0.00 %
class_id = 41, name = nabisco_nilla_wafers,       ap = 0.00 %
class_id = 42, name = pepperidge_farm_milano_cookies_double_chocolate,    ap = 0.00 %
class_id = 43, name = campbells_chicken_noodle_soup,      ap = 0.00 %
class_id = 44, name = frappuccino_coffee,         ap = 0.00 %
class_id = 45, name = chewy_dips_chocolate_chip,          ap = 34.73 %
class_id = 46, name = chewy_dips_peanut_butter,           ap = 0.00 %
class_id = 47, name = nature_vally_fruit_and_nut,         ap = 0.00 %
class_id = 48, name = cheerios,           ap = 0.00 %
class_id = 49, name = lindt_excellence_cocoa_dark_chocolate,      ap = 0.00 %
class_id = 50, name = hersheys_symphony,          ap = 0.00 %
class_id = 51, name = campbells_chunky_classic_chicken_noodle,    ap = 0.00 %
class_id = 52, name = martinellis_apple_juice,    ap = 0.00 %
class_id = 53, name = dove_pink,          ap = 0.00 %
class_id = 54, name = dove_white,         ap = 0.00 %
class_id = 55, name = david_sunflower_seeds,      ap = 0.00 %
class_id = 56, name = monster_energy,     ap = 0.00 %
class_id = 57, name = act_ii_butter_lovers_popcorn,       ap = 0.00 %
class_id = 58, name = coca_cola_glass_bottle,     ap = 0.00 %
class_id = 59, name = twix,       ap = 0.00 %
 for thresh = 0.24, precision = 0.83, recall = 0.71, F1-score = 0.76
 for thresh = 0.24, TP = 1362, FP = 282, FN = 562, average IoU = 62.79 %

 mean average precision (mAP) = 0.831957, or 83.20 %
Total Detection Time: 32.000000 Seconds
(base) truongnx@marlin:~/aix2022/skeleton-v1.1/bin$
```

How to compile and run: Windows

- After decompress the code, you can use the following commands

  Go to skeleton-v1.1

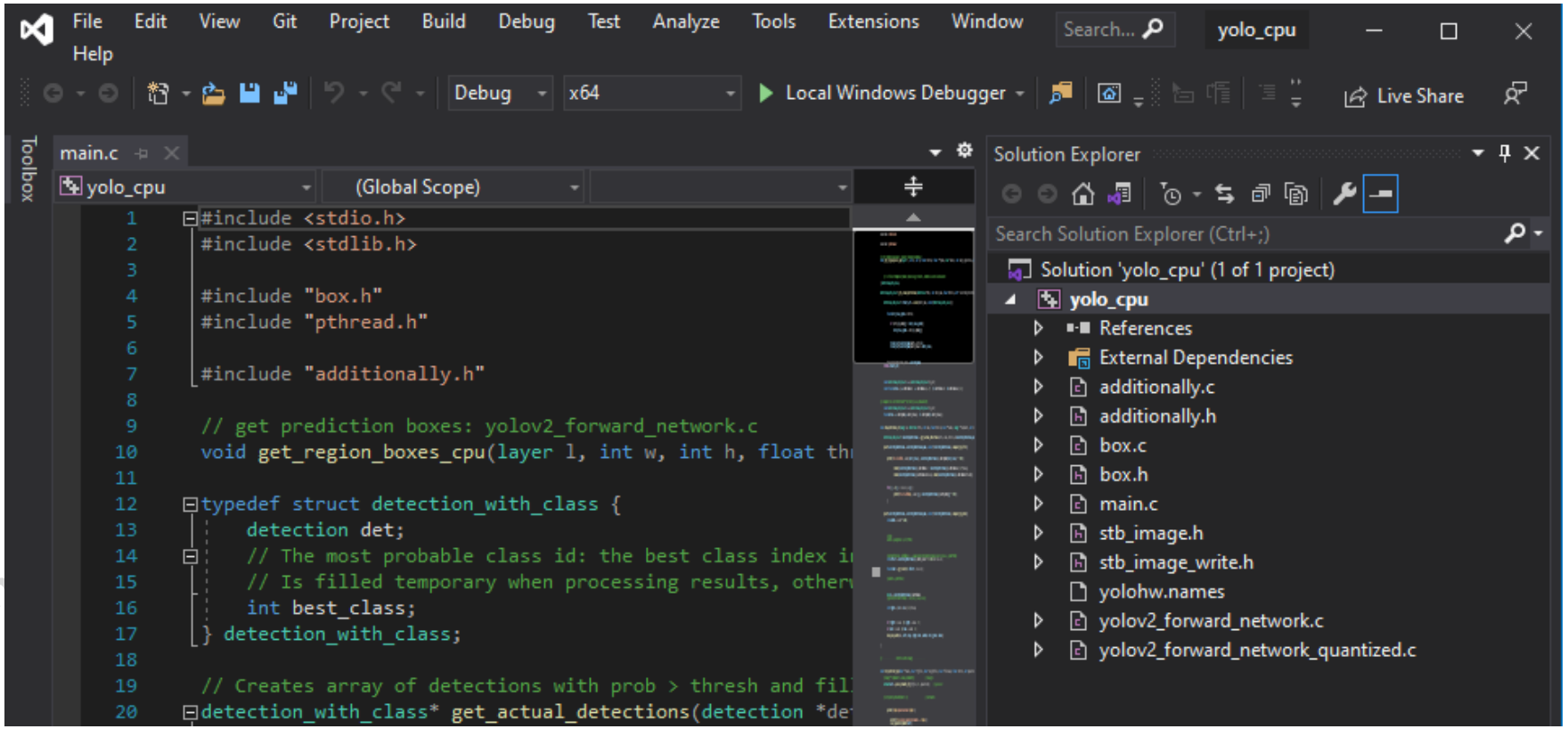
  Open the VS solution (yolo_cpu.sln) and compile the code

  ```
  cd bin/dataset                  // Go to the bin/dataset/
  python make_list_cur.py         // Update the directories for test images
  cd ..                           // Go to bin/
  yolo_cpu.cmd                    // Do inference on all test images, and calculate mAP
  ```

- Note:
  - If there is a version conflict, remove yolo_cpu.sln, and double click to yolo_cpu.vcxproj to make a new project with your VS version

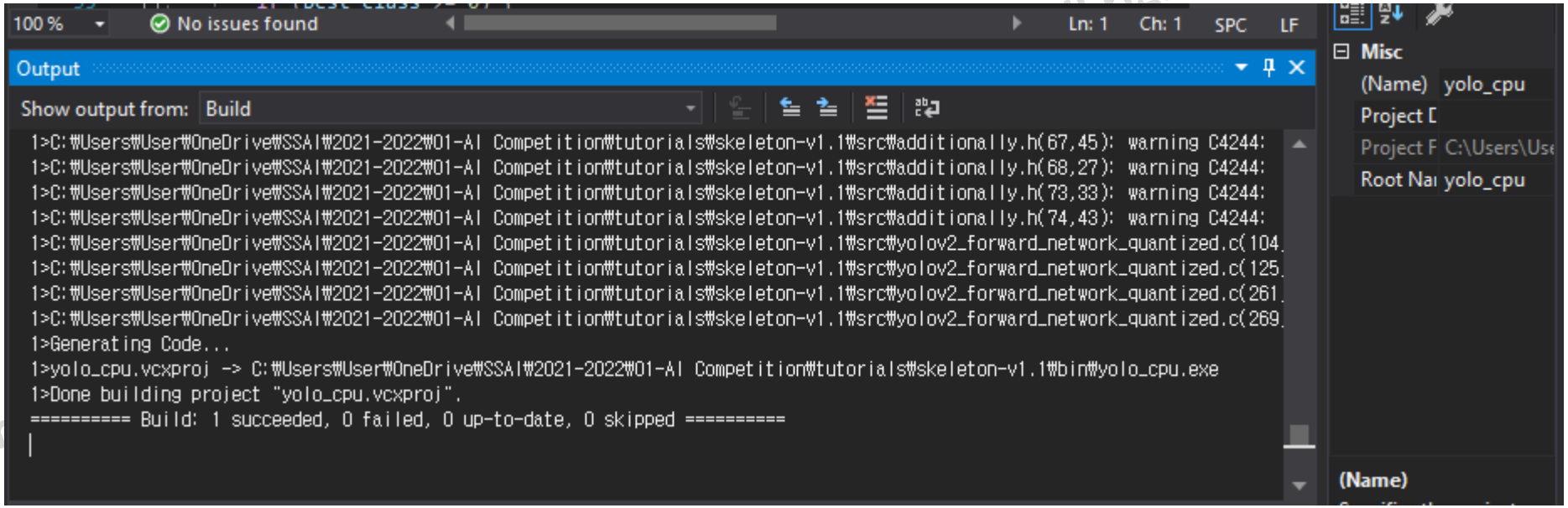
How to compile and run: Windows

- Open yolo_cpu.sln
- The default mode is Debug/x64



How to compile and run: Windows

- Choose the target mode: Debug or Release
- Build → Rebuild Solution
  - x64/ is generated or updated
  - The executable file is bin/yolo_cpu.exe



How to compile and run: Windows

- Loading a model, test images and do evaluation

  yolo_cpu.exe detector map yolohw.names yolov3-tiny-aix2022.cfg yolov3-tiny-aix2022.weights -thresh 0.24

```
C:₩Users₩User₩OneDrive₩SSAI₩2021-2022₩01-AI Competition₩tutorials₩skeleton-v1.1₩bin>yolo_cpu.exe
detector map yolohw.names yolov3-tiny-aix2022.cfg yolov3-tiny-aix2022.weights -thresh 0.24
valid: Using default 'dataset/target.txt'
names: Using default 'yolohw.names'
layer     filters    size              input                output
    0 conv     16  3 x 3 / 1   320 x 320 x   3   ->   320 x 320 x  16 0.088 BF
    1 max          2 x 2 / 2   320 x 320 x  16   ->   160 x 160 x  16
    2 conv     32  3 x 3 / 1   160 x 160 x  16   ->   160 x 160 x  32 0.236 BF
    3 max          2 x 2 / 2   160 x 160 x  32   ->    80 x  80 x  32
    4 conv     64  3 x 3 / 1    80 x  80 x  32   ->    80 x  80 x  64 0.236 BF
    5 max          2 x 2 / 2    80 x  80 x  64   ->    40 x  40 x  64
    6 conv    128  3 x 3 / 1    40 x  40 x  64   ->    40 x  40 x 128 0.236 BF
    7 max          2 x 2 / 2    40 x  40 x 128   ->    20 x  20 x 128
    8 conv    128  3 x 3 / 1    20 x  20 x 128   ->    20 x  20 x 128 0.118 BF
    9 max          2 x 2 / 2    20 x  20 x 128   ->    10 x  10 x 128
   10 conv    128  3 x 3 / 1    10 x  10 x 128   ->    10 x  10 x 128 0.029 BF
   11 max          2 x 2 / 1    10 x  10 x 128   ->    10 x  10 x 128
   12 conv    128  3 x 3 / 1    10 x  10 x 128   ->    10 x  10 x 128 0.029 BF
   13 conv    195  1 x 1 / 1    10 x  10 x 128   ->    10 x  10 x 195 0.005 BF
   14 yolo
   15 route  12
   16 conv    128  1 x 1 / 1    10 x  10 x 128   ->    10 x  10 x 128 0.003 BF
   17 upsample            2x    10 x  10 x 128   ->    20 x  20 x 128
   18 route  17 8
   19 conv    128  3 x 3 / 1    20 x  20 x 256   ->    20 x  20 x 128 0.236 BF
   20 conv    195  1 x 1 / 1    20 x  20 x 128   ->    20 x  20 x 195 0.020 BF
   21 yolo
Loading weights from yolov3-tiny-aix2022.weights...Done!
```

How to compile and run: Windows

- Report the accuracy of all 60 items
- The 32-bit model achieves the mean average precision (mAP) is 83.20%.

```
class_id = 30, name = coffee_mate_french_vanilla,         ap = 97.68 %
class_id = 31, name = pepperidge_farm_milk_chocolate_macadamia_cookies,          ap = 71.63 %
class_id = 32, name = kitkat_king_size,           ap = 85.32 %
class_id = 33, name = snickers,           ap = 36.60 %
class_id = 34, name = toblerone_milk_chocolate,           ap = 97.04 %
class_id = 35, name = clif_z_bar_chocolate_chip,          ap = 98.70 %
class_id = 36, name = nature_valley_crunchy_oats_n_honey,         ap = 72.28 %
class_id = 37, name = ritz_crackers,      ap = 97.73 %
class_id = 38, name = palmolive_orange,           ap = 55.47 %
class_id = 39, name = crystal_hot_sauce,          ap = 100.00 %
class_id = 40, name = tapatio_hot_sauce,          ap = 0.00 %
class_id = 41, name = nabisco_nilla_wafers,       ap = 0.00 %
class_id = 42, name = pepperidge_farm_milano_cookies_double_chocolate,    ap = 0.00 %
class_id = 43, name = campbells_chicken_noodle_soup,      ap = 0.00 %
class_id = 44, name = frappuccino_coffee,         ap = 0.00 %
class_id = 45, name = chewy_dips_chocolate_chip,          ap = 34.73 %
class_id = 46, name = chewy_dips_peanut_butter,           ap = 0.00 %
class_id = 47, name = nature_vally_fruit_and_nut,         ap = 0.00 %
class_id = 48, name = cheerios,           ap = 0.00 %
class_id = 49, name = lindt_excellence_cocoa_dark_chocolate,      ap = 0.00 %
class_id = 50, name = hersheys_symphony,          ap = 0.00 %
class_id = 51, name = campbells_chunky_classic_chicken_noodle,    ap = 0.00 %
class_id = 52, name = martinellis_apple_juice,    ap = 0.00 %
class_id = 53, name = dove_pink,          ap = 0.00 %
class_id = 54, name = dove_white,         ap = 0.00 %
class_id = 55, name = david_sunflower_seeds,      ap = 0.00 %
class_id = 56, name = monster_energy,     ap = 0.00 %
class_id = 57, name = act_ii_butter_lovers_popcorn,       ap = 0.00 %
class_id = 58, name = coca_cola_glass_bottle,     ap = 0.00 %
class_id = 59, name = twix,       ap = 0.00 %
 for thresh = 0.24, precision = 0.79, recall = 0.68, F1-score = 0.73
 for thresh = 0.24, TP = 1300, FP = 344, FN = 624, average IoU = 60.41 %

 mean average precision (mAP) = 0.831957, or 83.20 %
Total Detection Time: 4.000000 Seconds

C:₩Users₩User₩OneDrive₩SSAI₩2021-2022₩01-AI Competition₩tutorials₩skeleton-v1.1₩bin>pause
Press any key to continue . . .
```

Road map

Review

YOLO network

Reference S/W

Source files

- additionally.c *// Definitions of darknet functions used*
- additionally.h *// Declaration of darknet functions + additional functions for forward pass of yolo model*
- box.c
- box.h

  } *// For bounding boxes*
- stb_image_write.h
- stb_image.h

  } *// For loading/writing images*
- yolov2_forward_network.c *// Functions for forward pass of yolo network*
- **yolov2_forward_network_quantized.c** *// Functions for quantization, saving of the quantized model, and the forward pass of quantized yolo model*
- main.c *// The main functions*

You should mainly edit this file for quantization!

main.c

- void test_detector_cpu(

```
    char **names,                   // List of all items (bin/yolohw.names)
    char *cfgfile,                  // Configuration file (bin/yolov3-tiny-aix2022.cfg)
    char *weightfile,               // Configuration file (bin/yolov3-tiny-aix2022.weights)
    char *filename,                 // Input image file
    float thresh,                   // Hierarchical threshold
    int quantized,                  // On/off quantization
    int save_params,                // On/off save output
    int dont_show                   // Don't show
)
```

test_detector_cpu

- Parse the configuration file
  - A network architecture is stored in the variable "net"
  - Example: Layer index 0
    - Convolutional, 16 filters, filter size 3x3, padding = 1
    - Use Leaky function

```
155  // Detect on Image: this function uses other functions not from this file
156  void test_detector_cpu(char **names, char *cfgfile, char *weightfile, char *filename, float thresh, int quantized, int dont_show)
157  {
158      //image **alphabet = load_alphabet();              // image.c
159      image **alphabet = NULL;
160      network net = parse_network_cfg(cfgfile, 1, quantized);    // parser.c
161      if (weightfile) {
162          load_weights_upto_cpu(&net, weightfile, net.n);    // parser.c
163      }
164      //set_batch_network(&net, 1);                      // network.c
165      srand(2222222);
166      yolov2_fuse_conv_batchnorm(net);
167      calculate_binary_weights(net);
168      if (quantized) {
169          printf("\n\n Quantization! \n\n");
170          quantization_and_get_multipliers(net);
171      }
```

yolov3-tiny-aix2022

Layer 0

```
25  [convolutional]
26  batch_normalize=1
27  filters=16
28  size=3
29  stride=1
30  pad=1
31  activation=leaky
32
33  [maxpool]
34  size=2
35  stride=2
36
37  [convolutional]
38  batch_normalize=1
39  filters=32
40  size=3
41  stride=1
42  pad=1
43  activation=leaky
44
45  [maxpool]
46  size=2
47  stride=2
```

yolov3-tiny-aix2022.cfg

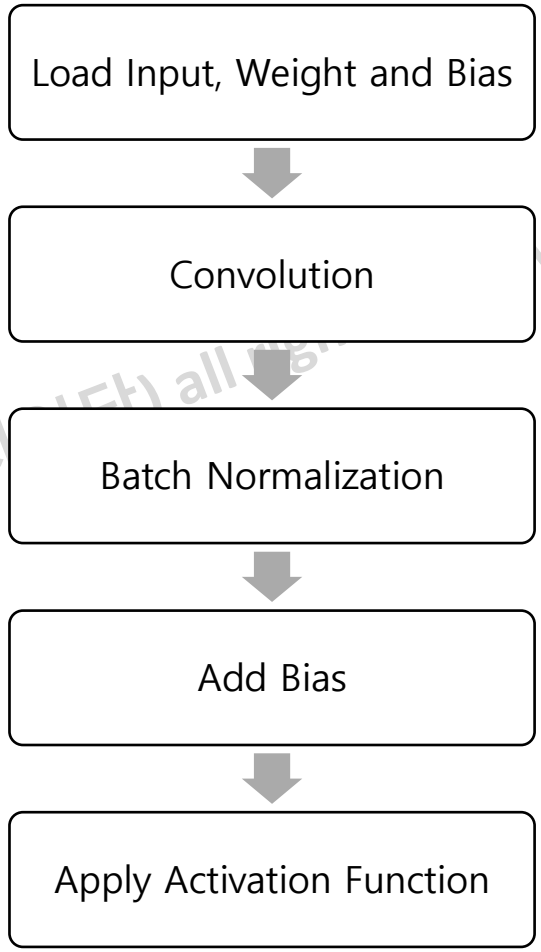
test_detector_cpu

- Load an input image (load_image)
  - X: image data
- Call a function
  - network_predict_cpu(net,X): Inference
  - network_predict_quantized(net,X): Inference with a quantization mode

```
176     float nms = .4;
177     while (1) {
178         if (filename) {
179             strncpy(input, filename, 256);
180         }
181         else {
182             printf("Enter Image Path: ");
183             fflush(stdout);
184             input = fgets(input, 256, stdin);
185             if (!input) return;
186             strtok(input, "\n");
187         }
188         image im = load_image(input, 0, 0, 3);           // image.c
189         image sized = resize_image(im, net.w, net.h);    // image.c
190         layer l = net.layers[net.n - 1];
191
192         box *boxes = calloc(l.w*l.h*l.n, sizeof(box));
193         float **probs = calloc(l.w*l.h*l.n, sizeof(float *));
194         for (j = 0; j < l.w*l.h*l.n; ++j) probs[j] = calloc(l.classes, sizeof(float *));
195
196         float *X = sized.data;
197         time = clock();
198         //network_predict(net, X);
199 #ifdef GPU
200         if (quantized) {
201             network_predict_gpu_cudnn_quantized(net, X);    // quantized
202                                                             //nms = 0.2;
203         }
204         else {
205             network_predict_gpu_cudnn(net, X);
206         }
207 #else
208 #ifdef OPENCL
209         network_predict_opencl(net, X);
210 #else
211         if (quantized) {
212             network_predict_quantized(net, X);    // quantized
213             nms = 0.2;
214         }
215         else {
216             network_predict_cpu(net, X);
217         }
218 #endif
219 #endif
```

yolov2_forward_network_cpu

```
390  void yolov2_forward_network_cpu(network net, network_state state)
391  {
392      state.workspace = net.workspace;
393      int i;
394      for (i = 0; i < net.n; ++i) {
395          state.index = i;
396          layer l = net.layers[i];
397
398          if (l.type == CONVOLUTIONAL) {
399              forward_convolutional_layer_cpu(l, state);
400          }
401          else if (l.type == MAXPOOL) {
402              forward_maxpool_layer_cpu(l, state);
403          }
404          else if (l.type == ROUTE) {
405              forward_route_layer_cpu(l, state);
406          }
407          else if (l.type == REORG) {
408              forward_reorg_layer_cpu(l, state);
409          }
410          else if (l.type == UPSAMPLE) {
411              forward_upsample_layer_cpu(l, state);
412          }
413          else if (l.type == SHORTCUT) {
414              forward_shortcut_layer_cpu(l, state);
415          }
416          else if (l.type == YOLO) {
417              forward_yolo_layer_cpu(l, state);
418          }
419          else if (l.type == REGION) {
420              forward_region_layer_cpu(l, state);
421          }
422          else {
423              printf("\n layer: %d \n", l.type);
424          }
425
426
427          state.input = l.output;
428      }
429  }
```



void forward_convolutional_layer_cpu(layer l, network_state state) *[yolov2_forward_network.c]*

yolov2_forward_network.c

- void forward_convolutional_layer_cpu(layer l, network_state state)

```
// 4 layers in 1: convolution, batch-normalization, BIAS and activation
void forward_convolutional_layer_cpu(layer l, network_state state)
{

    int out_h = (l.h + 2 * l.pad - l.size) / l.stride + 1;    // output_height=input_height for stride=1 and pad=1
    int out_w = (l.w + 2 * l.pad - l.size) / l.stride + 1;    // output_width=input_width for stride=1 and pad=1
    int i, f, j;

    // fill zero (ALPHA)
    for (i = 0; i < l.outputs*l.batch; ++i) l.output[i] = 0;

    // l.n - number of filters on this layer
    // l.c - channels of input-array
    // l.h - height of input-array
    // l.w - width of input-array
    // l.size - width and height of filters (the same size for all filters)

    // 1. Convolution
    int m = l.n;
    int k = l.size*l.size*l.c;
    int n = out_h*out_w;
    float *a = l.weights;
    float *b = state.workspace;
    float *c = l.output;

    // convolution as GEMM (as part of BLAS)
    for (i = 0; i < l.batch; ++i) {
        im2col_cpu_custom(state.input, l.c, l.h, l.w, l.size, l.stride, l.pad, b);    // AVX2
        int t;
        #pragma omp parallel for
        for (t = 0; t < m; ++t) {
            gemm_nn(1, n, k, 1, a + t*k, k, b, n, c + t*n, n);
        }
        c += n*m;
        state.input += l.c*l.h*l.w;

    }

    int const out_size = out_h*out_w;
```

yolov2_forward_network.c

- void forward_convolutional_layer_cpu(layer l, network_state state)

```
// 2. Batch normalization
if (l.batch_normalize) {
    int b;
    for (b = 0; b < l.batch; b++) {
        for (f = 0; f < l.out_c; ++f) {
            for (i = 0; i < out_size; ++i) {
                int index = f*out_size + i;
                l.output[index+b*l.outputs] = (l.output[index+b*l.outputs] - l.rolling_mean[f]) / (sqrtf(l.rolling_variance[f]) + .000001f);
            }
        }

        // scale_bias
        for (i = 0; i < l.out_c; ++i) {
            for (j = 0; j < out_size; ++j) {
                l.output[i*out_size + j+b*l.outputs] *= l.scales[i];
            }
        }
    }
}

// 3. Add BIAS
{
    int b;
    for (b = 0; b < l.batch; b++) {
        for (i = 0; i < l.n; ++i) {
            for (j = 0; j < out_size; ++j) {
                l.output[i*out_size + j + b*l.outputs] += l.biases[i];
            }
        }
    }
}

// 4. Activation function (LEAKY or LINEAR)
activate_array_cpu_custom(l.output, l.outputs*l.batch, l.activation);
```

Incoming lecture …

- Quantization
- Save a quantized model