# SamsungDS22 Project

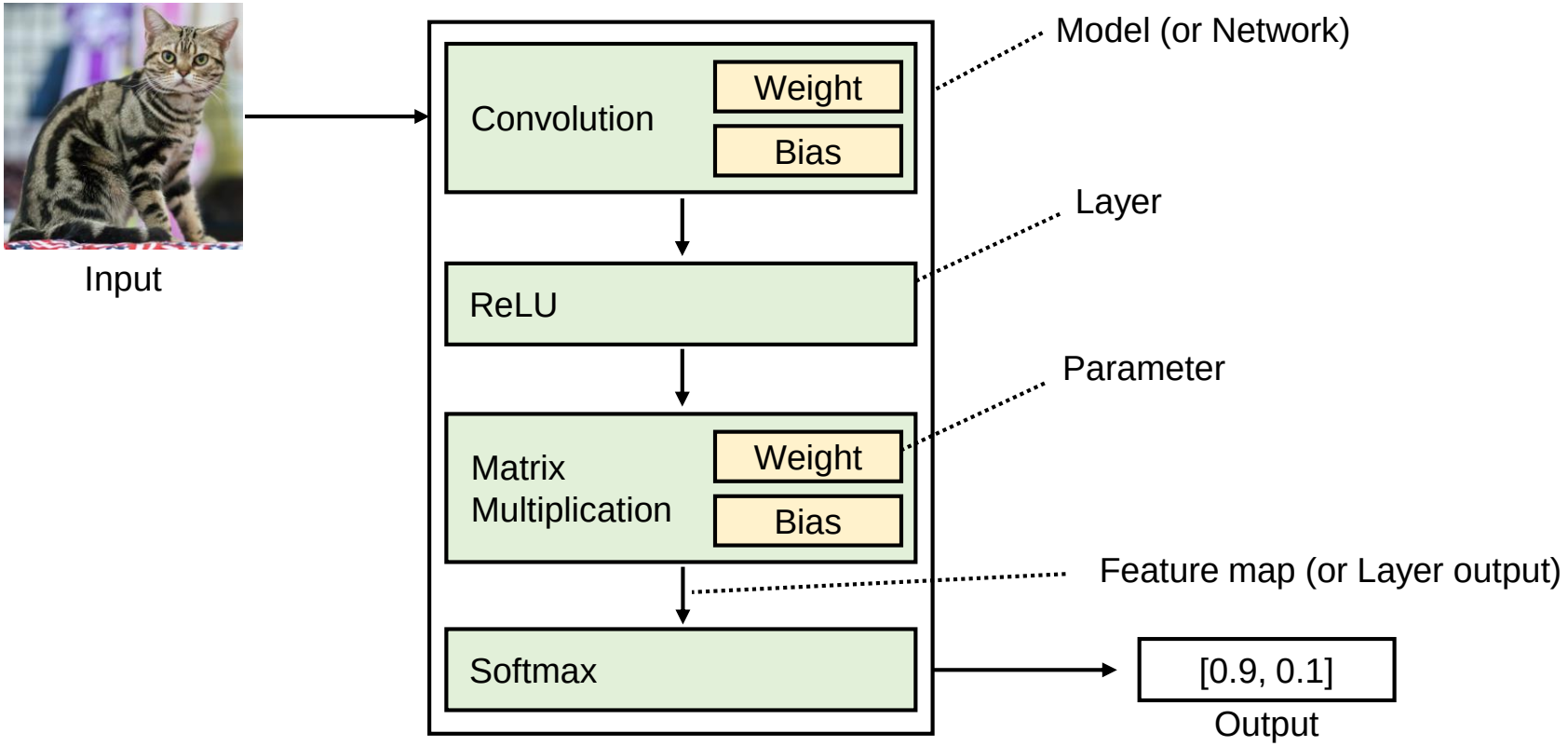
Colorizer

THUNDER Research Group
Seoul National University
서울대학교 천둥 연구실

# Background

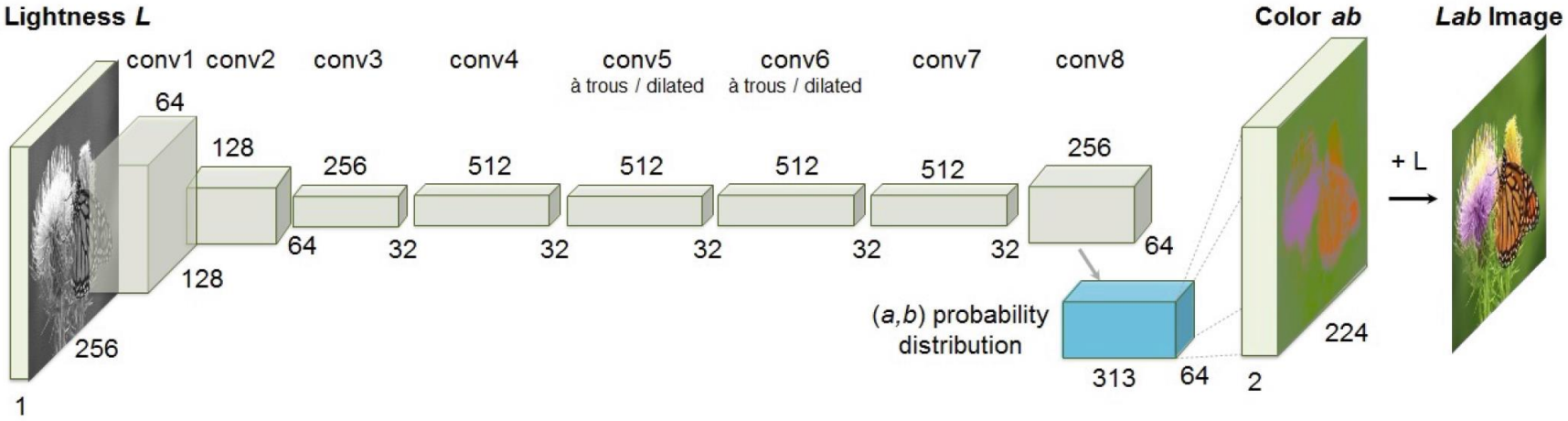
# Deep Learning

- 학습(Train): 파라미터를 찾는 단계
- 추론(Inference): 학습한 파라미터로 주어진 입력에 대해 출력을 계산



# Project Goal

- 주어진 흑백 이미지를 컬러 이미지로 변환
- 다음 논문의 결과를 사용
  - Zhang, Richard, Phillip Isola, and Alexei A. Efros. "Colorful image colorization." European conference on computer vision. Springer, Cham, 2016.
  - https://github.com/richzhang/colorization



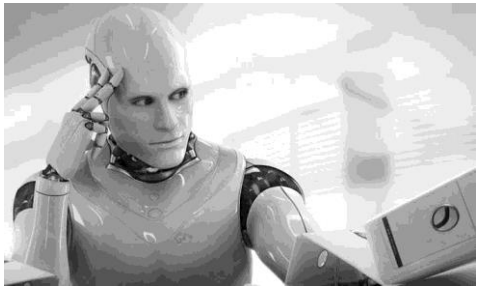
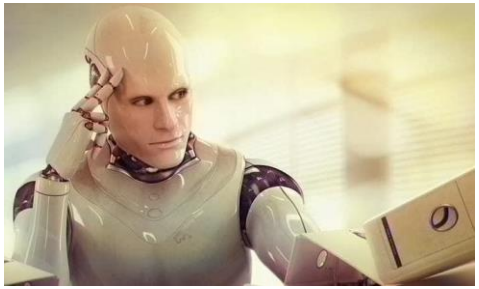
# Examples



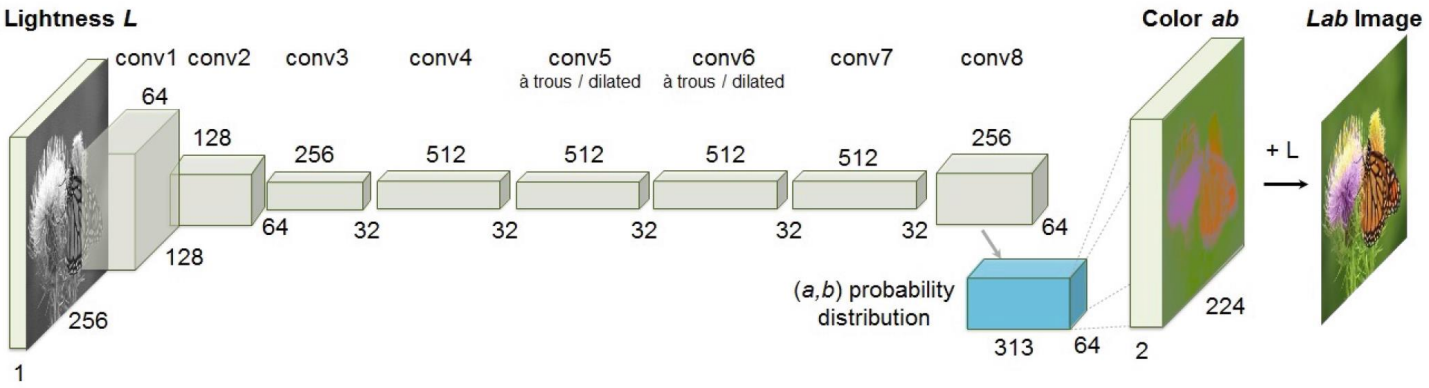






# Model Architecture




- 8개의 블럭으로 구성

Block 1~2

| Convolution |
|---|
| Convolution |
| BatchNorm2d |

Block 3~8

| Convolution |
|---|
| Convolution |
| Convolution |
| BatchNorm2d |

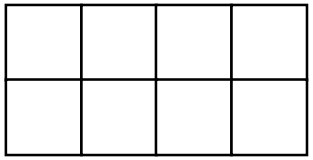
# Tensor

- 딥 러닝에서 데이터를 다루는 단위

1D Tensor
(e.g., Vector)

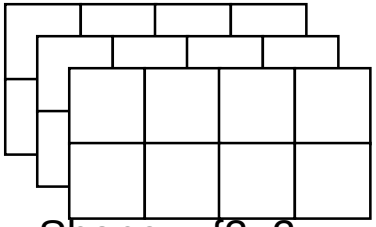
Shape = {4}

2D Tensor
(e.g., Matrix)



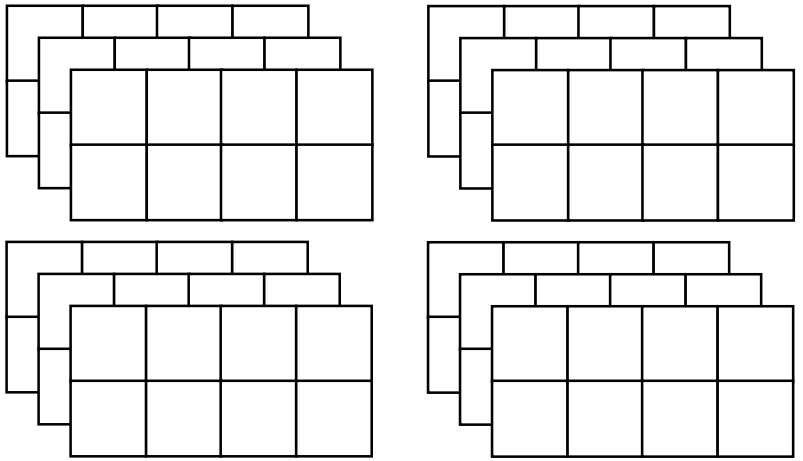
Shape = {2, 4}

3D Tensor
(e.g., RGB image)



Shape = {3, 2, 4}

4D Tensor
(e.g., multiple images, convolution weight)



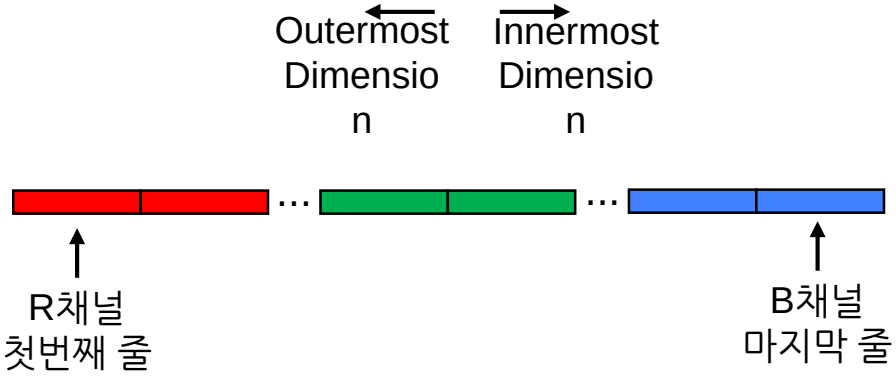
Shape = {4, 3, 2, 4}

# Data Layout

- 다차원 텐서도 메모리에는 1차원으로 저장되어 있음

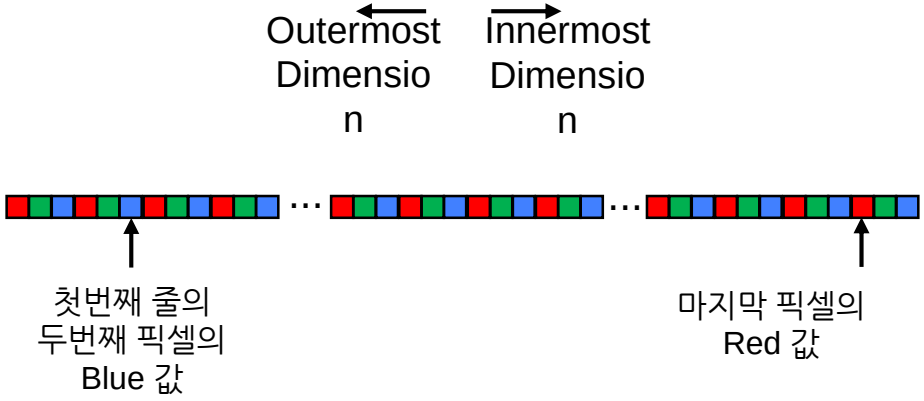
가로 256 픽셀, 세로 256 픽셀, RGB 채널을 가진 이미지 데이터

CHW = (Channel, Height, Width) =
(3, 256, 256)




array[c][h][w] ==
array[c * H * W + h * W + w]

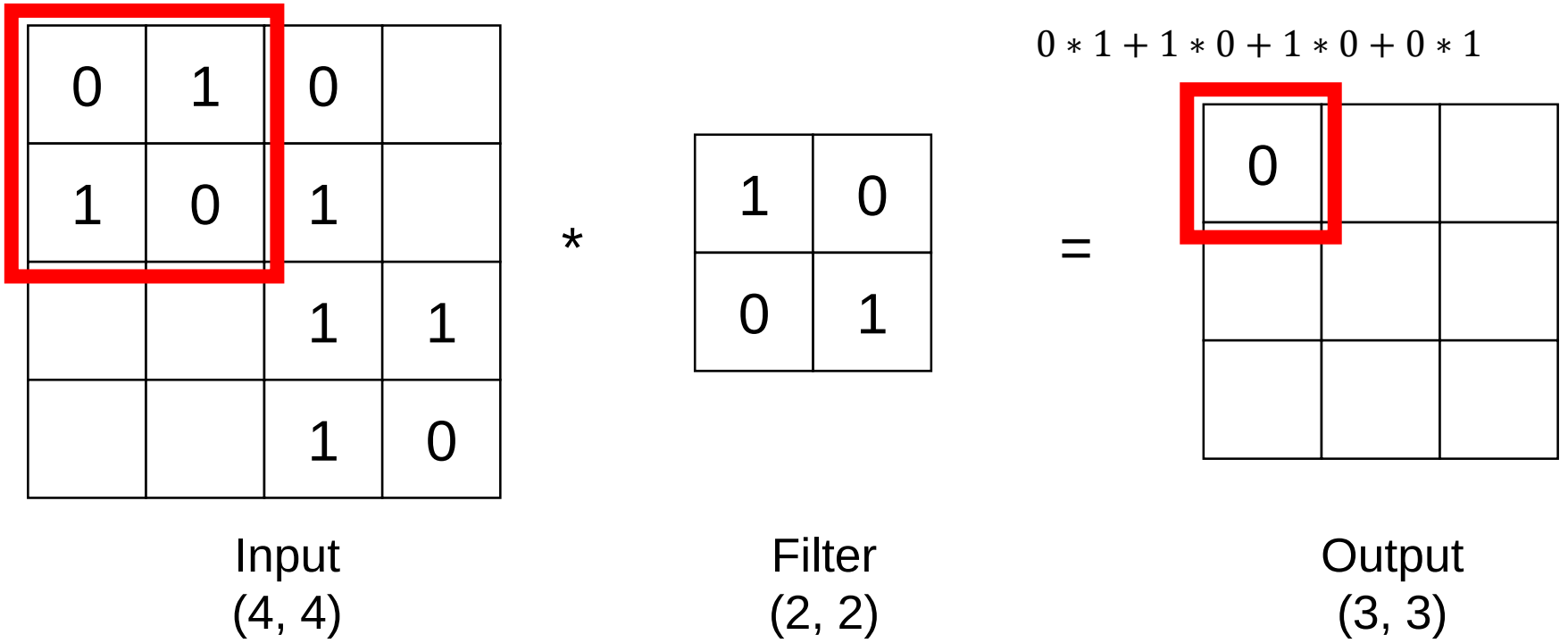
HWC = (Height, Width, Channel) =
(256, 256, 3)




array[h][w][c] ==
array[h * W * C + w * C + c]

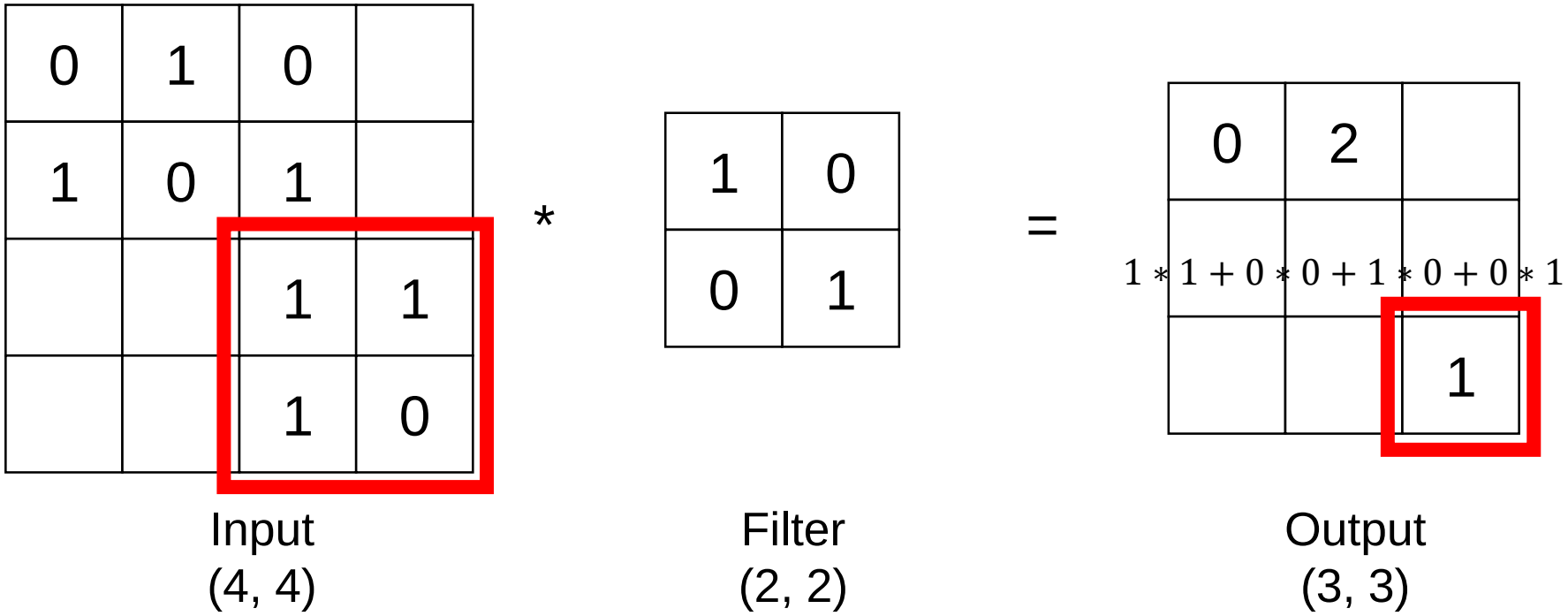
# Convolution

- 필터를 입력 위에서 움직이면서 내적 값을 계산
  - 특정 영역이 필터와 유사하다면, 대응되는 출력 값이 높게 나옴



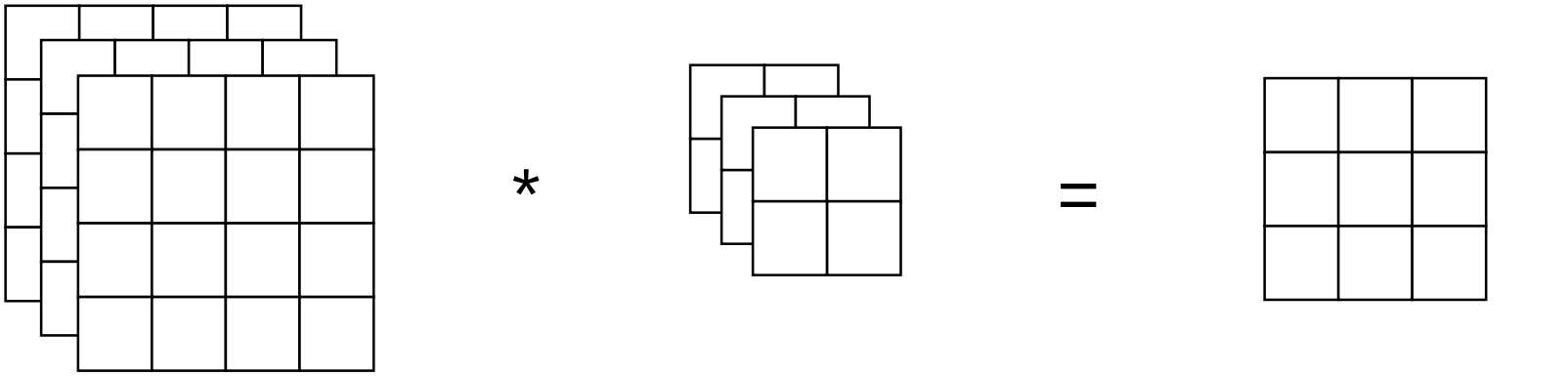
# Convolution

- 필터를 입력 위에서 움직이면서 내적 값을 계산
  - 특정 영역이 필터와 유사하다면, 대응되는 출력 값이 높게 나옴



# Convolution

- 입력의 채널이 여러 개라면?



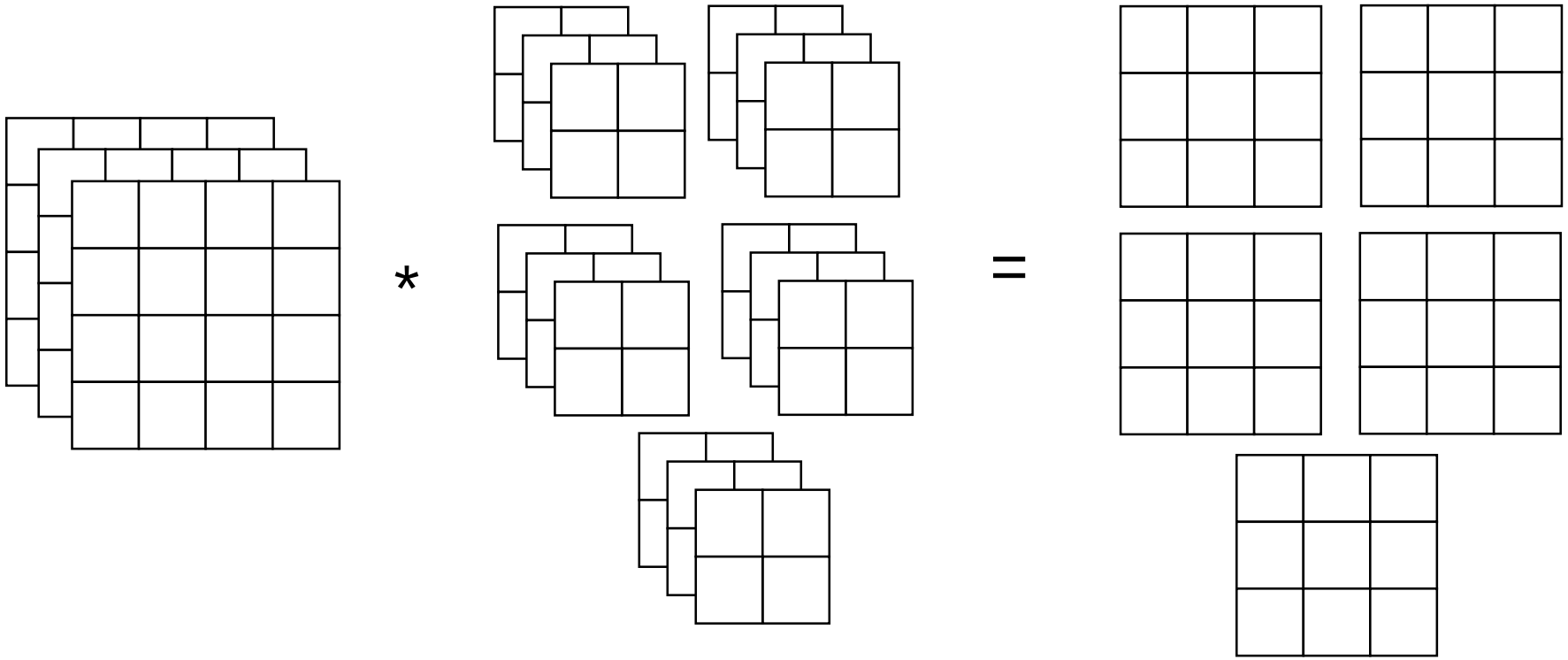
Input
(C, H, W) = (3, 4, 4)

Filter
(C, R, S) = (3, 2, 2)

Output
(OH, OW) = (3, 3)

# Convolution

- 출력의 채널이 여러 개라면?



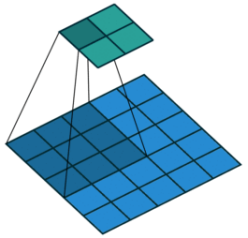
Input
(C, H, W) = (3, 4, 4)

Filter
(K, C, R, S) = (5, 3, 2, 2)

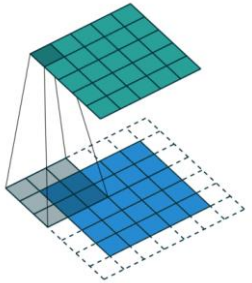
Output
(K, OH, OW) = (5, 3, 3)

# Convolution Variations

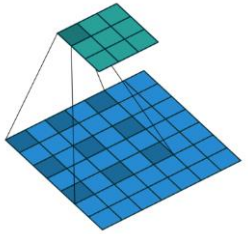
- Stride
  - 필터가 움직이는 간격



- Pad
  - 입력 가장자리에 0을 추가하여 출력 크기 조절



- Dilation
  - 필터 원소 하나하나 사이의 간격



- Transposed convolution
  - 코드 참고

- 이해에 도움이 되는 자료
  - https://github.com/vdumoulin/conv_arithmetic

# Batch Normalization

- 입력이 뉴럴 네트워크를 통과하면서 평균이나 분산에 편향이 생기는 것을 완화시켜주는 연산
  - $E[x]$, $V[x]$ : 훈련 데이터에서 얻은 이동 평균과 이동 분산
  - $\beta$, $\gamma$ : 훈련된 offset 및 scale 값
  - $\varepsilon$ : Division-by-0를 방지하기 위한 값 (1e-5)

$$output = \beta + \frac{(input - E[x]) * \gamma}{\sqrt{V[x] + \varepsilon}}$$

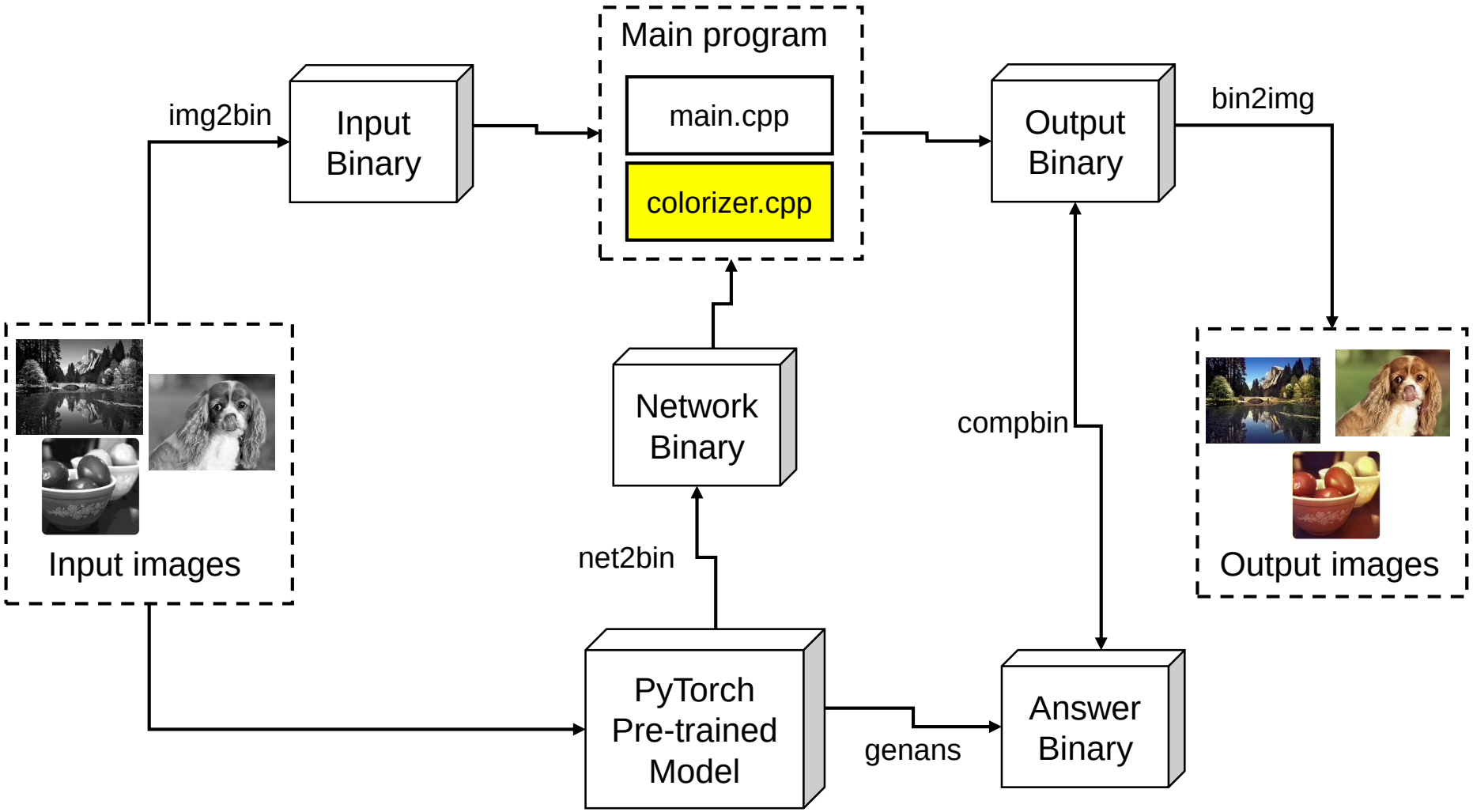
# Miscellaneous Operations

- NormalizeL : $y = (x - 50)/100$
- ReLU : $y = \max(x, 0)$
- Softmax : $y = \frac{e^x}{\sum e^x}$
- Upsample : Bilinear Interpolation
- UnnormalizeAB : $y = x * 110$

- 뼈대코드 참고

# Skeleton Code

# Overview


Main program
main.cpp
colorizer.cpp
img2bin
Input Binary
Output Binary
bin2img
Input images
Network Binary
compbin
Output images
net2bin
PyTorch Pre-trained Model
genans
Answer Binary

# 목표

- colorizer.cpp 를 병렬화 하여 빠르게 하는 것
- 계산 노드 3~4개의 자원 모두 사용 (CPUs + 12~16 GPUs)

# 뼈대 코드 구조

- `seq/`
  - `timer.cpp, timer.h, util.cpp, util.h, main.cpp:` 수정 불가능
  - **`colorizer.cpp, colorizer.h`: 병렬화 대상**
  - **`Makefile, run.sh`: 적절하게 수정해서 사용**
- `common/`
  - `bins/`: 예제 입력 및 출력 데이터
  - `bin2img, compbin, img2bin`: 보조 프로그램
  - `imgs`: 이미지 예제
  - `network.bin`: Pre-trained network binary
- `misc/` (optional; PyTorch required)
  - `genans, net2bin`: 보조 프로그램

# Makefile

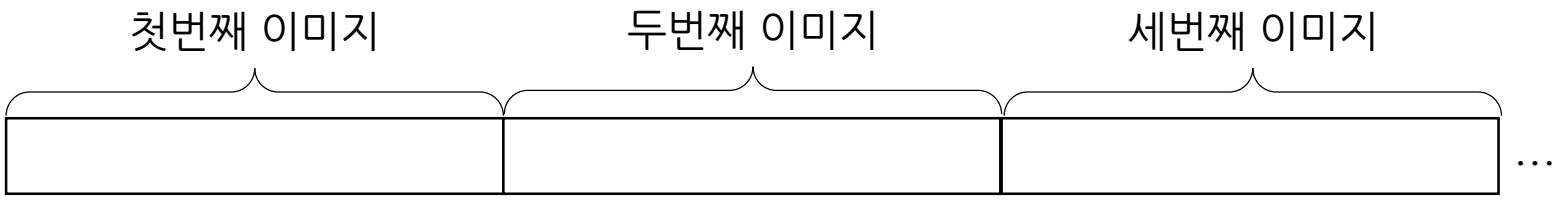
- 자유롭게 수정 가능
- **MPI를 사용하고자 하는 경우 CXXFLAGS에 -DUSE_MPI를 추가하여야 main.cpp가 올바르게 동작함**
- make 로 컴파일이 되어야 함
- make run 으로 실행이 되어야 함

# colorizer.cpp

- `ColorizerInit`
  - 초기화 작업 구현
  - `mat_mul_init`을 생각하면 됨
  - 시간 측정에 들어가지 않음
- `ColorizerFinalize`
  - 마무리 작업 구현
  - `mat_mul_finalize`를 생각하면 됨
  - 시간 측정에 들어가지 않음

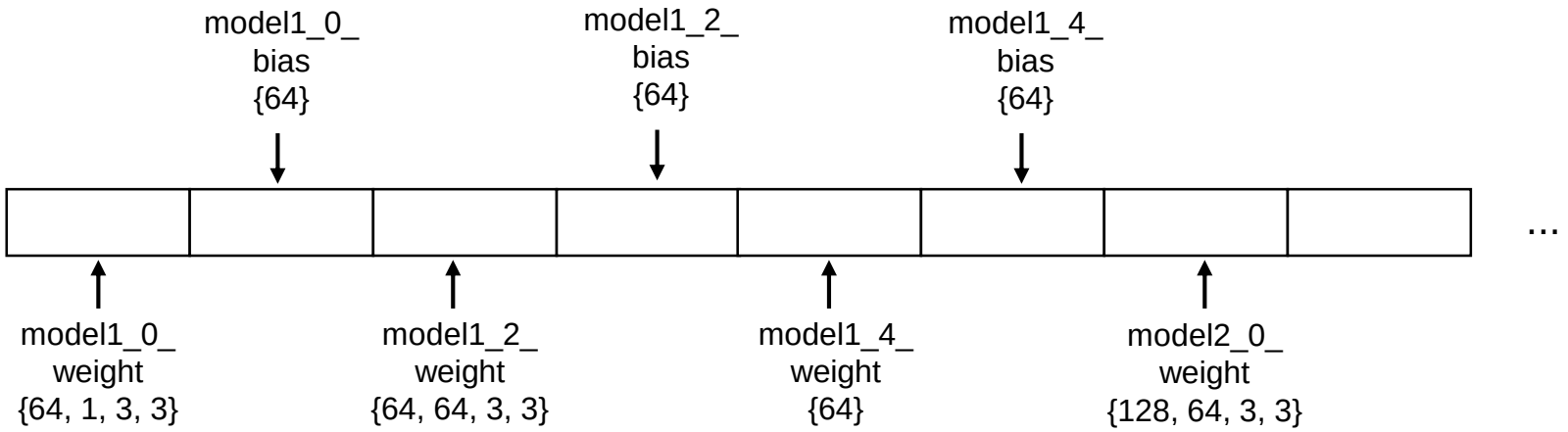
# colorizer.cpp - Colorize

- `void Colorizer(float* input, float* network, float* output, int N)`
  - N: 이미지 개수
  - input: 입력 이미지 데이터
    - 크기 : N * 256 * 256



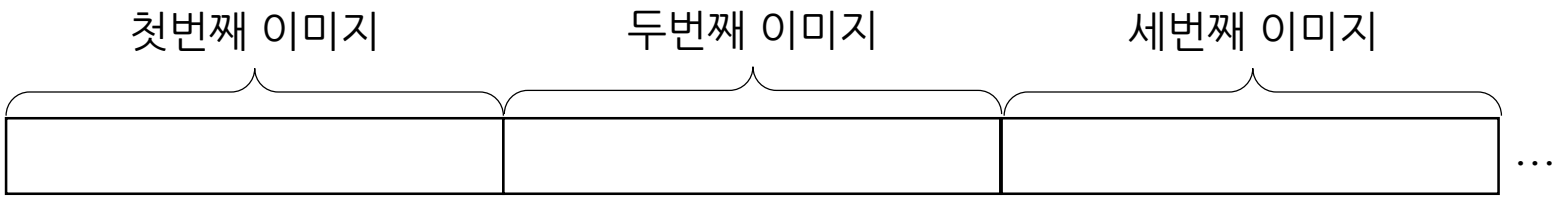
# colorizer.cpp - Colorize

- `void Colorizer(float* input, float* network, float* output, int N)`
  - network : 네트워크 파라미터 데이터
    - 포인터 연산을 이용하여 각 레이어의 파라미터로 분리
    - 각 파라미터의 순서와 크기는 코드 참고



# colorizer.cpp - Colorize

- `void Colorizer(float* input, float* network, float* output, int N)`
  - output : 출력 이미지 데이터
    - 크기 : N * **2** * 256 * 256



# 프로그램

- img2bin [흑백 이미지 목록] [출력 바이너리]
  - 이미지 여러 장을 main.cpp가 이해할 수 있는 바이너리 형태로 변환
- main [입력 바이너리] [네트워크 바이너리] [출력 바이너리]
  - 입력 바이너리와 네트워크를 받아 출력 바이너리 생성
- compbin [비교할 바이너리1] [비교할 바이너리2]
  - 출력 바이너리 2개를 비교해서 차이의 최대값을 출력
  - **차이의 최대값이 0.001 (1e-3) 이하여야만 답으로 인정**
- bin2img [원본 흑백 이미지] [출력 바이너리]
  - 원본 흑백 이미지를 출력 바이너리에 담긴 컬러 정보를 이용하여 컬러 이미지 생성
  - 컬러 이미지는 원본 흑백 이미지가 있는 폴더에 생성됨
  - robot.png -> robot_colorized.png

# 프로그램 - PyTorch 필요

- 다음은 프로젝트를 하는데 있어 필수는 아니지만 PyTorch가 있다면 실행해볼 수 있는 프로그램임
- genans [흑백 이미지 목록] [출력 바이너리]
  - 이미지 여러 장을 PyTorch로 처리하여 출력 바이너리 생성
  - main과 genans로 생성된 출력 바이너리를 compbin으로 비교하여 답 검사 가능
- net2bin [네트워크 바이너리]
  - network.bin를 생성하는데 사용한 프로그램

# 실행 예시

- **반드시 계산 노드에서 실행할 것**

```
kjp4155@b1:~/project/project_skeleton/seq$ ./main ../common/bins/inputN1.bin ../common/network.bin outputN1.bin
Options:
  Input: ../common/bins/inputN1.bin
  Network: ../common/network.bin
  Output: outputN1.bin

Reading input...
Reading input done! (1 images, 0.025858 s)
Reading network...
Reading network done! (128964012 bytes, 10.968411 s)
Preparing output...
Preparing output done! (0.000028 s)
Initializing...
Initializing done! (0.000318 s)
Calculating...
Calculating done! (106.385733 s)
Performance: 0.009400 img/s
Writing output...
Writing output done! (0.231593 s)
Finalizing...
Finalizing done! (0.010222 s)
```

```
kjp4155@b1:~/project/project_skeleton/seq$ ../common/compbin ../common/bins/answerN1.bin outputN1.bin
Max of difference: 8.392333984375e-05
kjp4155@b1:~/project/project_skeleton/seq$
```

# 프로젝트 가이드

- 배운 병렬화 방법 모두 사용 가능
  - Pthread, OpenMP, OpenCL, MPI, CUDA, ...
- 외부 라이브러리 사용 금지
  - Intel MKL, clBLAS, cuDNN, cuBLAS 등 ...
- 새 파일을 추가 하거나 이름 변경은 가능함
  - kernel.cl 추가, colorizer.cpp를 colorizer.cu로 변경 등
  - Makefile 로 잘 컴파일 및 실행 되도록 유의
- main.cpp, util, timer는 수정 금지
  - 부득이하게 수정이 필요하다 생각되는 경우 문의

# 제약 조건

- 메모리 레이아웃 변경, 루프 순서 변경, 패딩 데이터/연산 추가, 레이어 병합 등의 최적화 기법은 허용

- 연산량을 줄이는 변경은 어떤 것이든 불허
  - 계산 복잡도가 다른 알고리즘을 써 연산의 양을 줄이거나, 일부 레이어를 누락하거나, 레이어 순서를 바꾸는 등의 변경

- 애매한 것은 조교에게 문의할 것

# 일정

- 목표: seq 코드 성능 측정 (N = 1, 2, 8, 16, 32, ..., 1024)
  - **seq 코드 이해하기**
  - inputN.bin, outputN.bin 만들기
  - 답 검증 & 성능 측정 하기

- 중간점검1: 5월 31일 (화)
  - 목표: 1GPU 에서 최소한의 효율로 돌아가도록
  - seq 코드 성능 측정 (N = 1, 2, 8, 16, 32,..., 1024)
  - 1GPU 사용 시 성능 측정

- 중간점검2: 6월 7일 (화)

- 최종 제출: 6월 14일

- 계산노드: b1, b2, b3
- slurm 사용 안함
- ssh 접속 및 실행
  - login 노드에서 ssh b1, ssh b2
  - 작업이 겹치지 않도록 유의