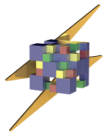
Term Project

Face Generator



THUNDER Research Group
Seoul National University
서울대학교 천둥 연구실

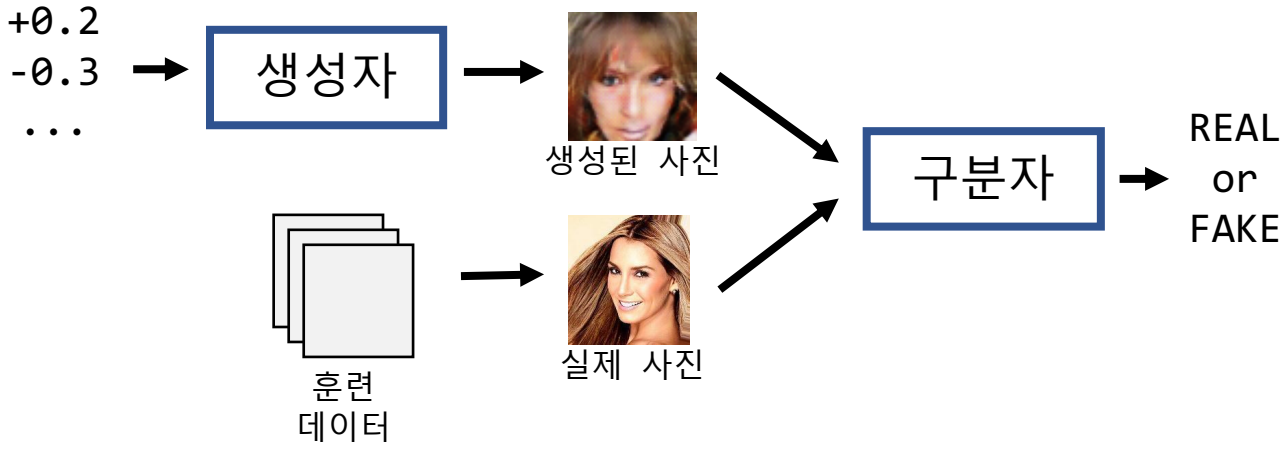
프로젝트 개요

- 임의의 노이즈를 입력받아 얼굴을 생성하는 프로그램

```
100
0.235 -0.889 0.592 ...
... 0.464 -0.007 -0.198
```

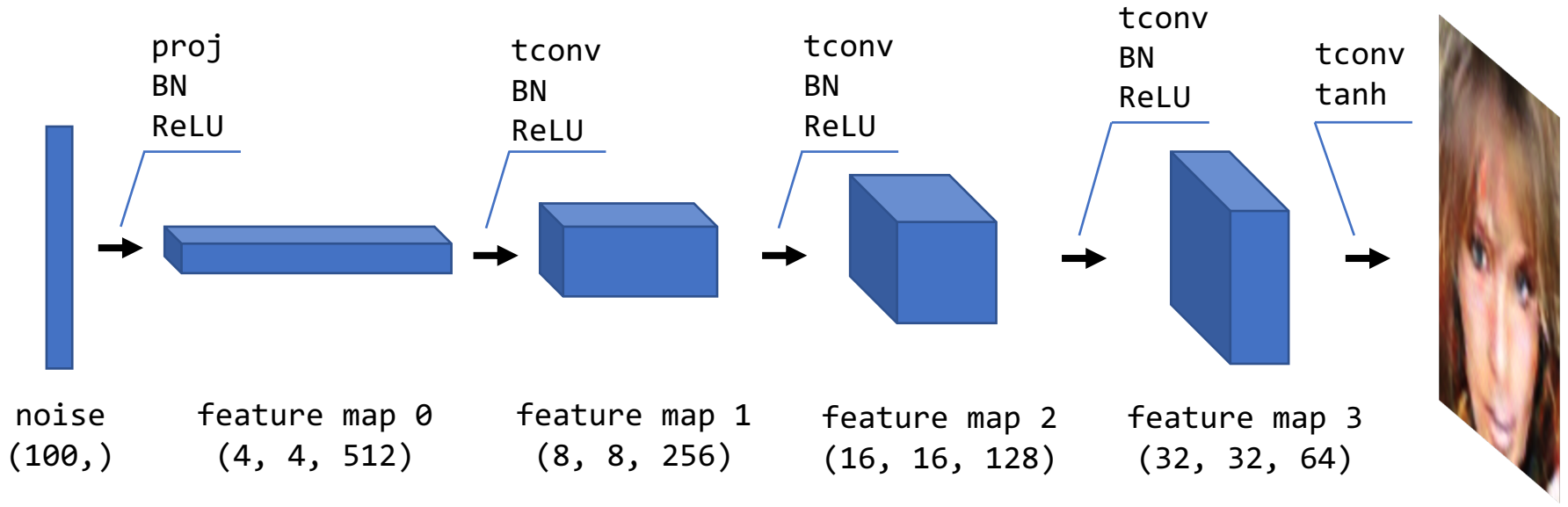
생성 알고리즘

- Generative Adversarial Network (GAN)
  - 뉴럴 네트워크의 일종으로 생성자(generator)와 구분자(discriminator)로 구성
  - 생성자는 노이즈를 입력 받아 실제와 비슷한 이미지를 생성하도록 훈련
  - 구분자는 이미지가 실제인지 생성된 것인지 구분하도록 훈련
  - 훈련이 끝나면 생성자는 실제와 유사한 이미지를 생성할 수 있게 됨



생성 알고리즘

- Deep Convolutional GAN (DCGAN)
  - 생성자와 구분자로 Convolutional Neural Network 사용
  - 본 프로젝트에서는 생성자만 구현

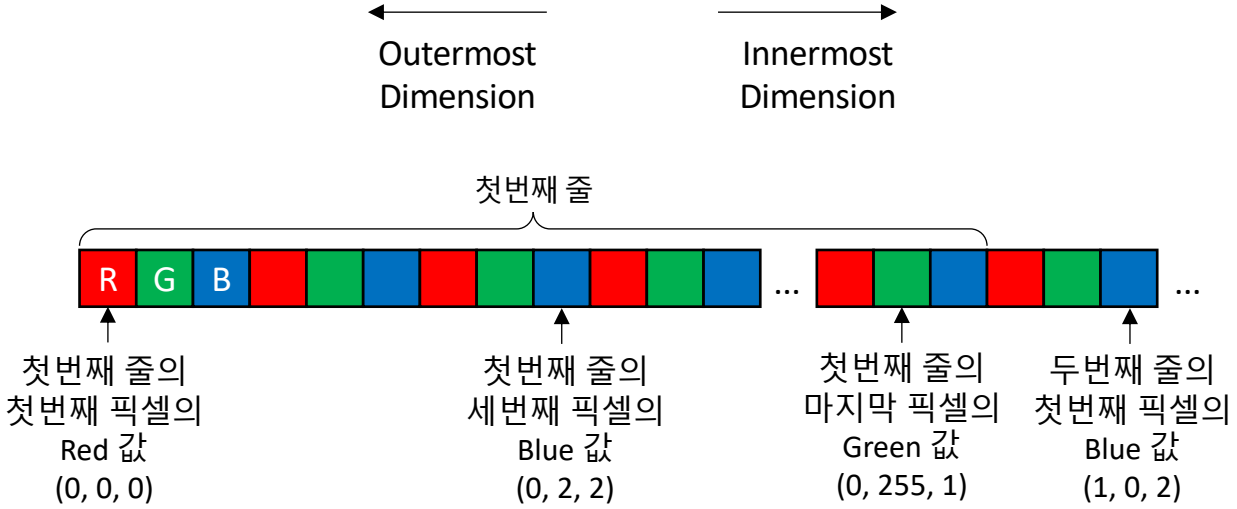


** tconv = transposed convolution, proj = linear projection, BN = batch normalization*

Data Layout

- 예) 가로 256 픽셀, 세로 256 픽셀, RGB 채널을 가진 이미지 데이터

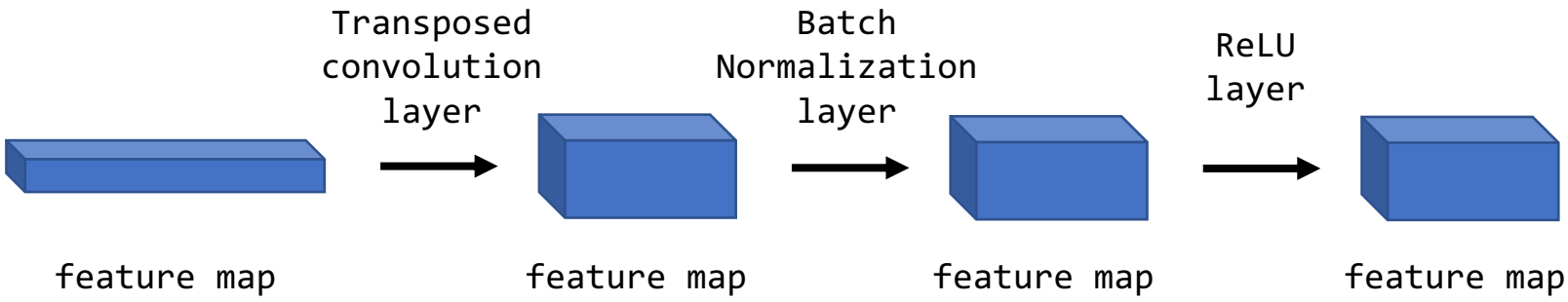
(Height, Width, Channel) = (256, 256, 3)



뉴럴 네트워크 기초

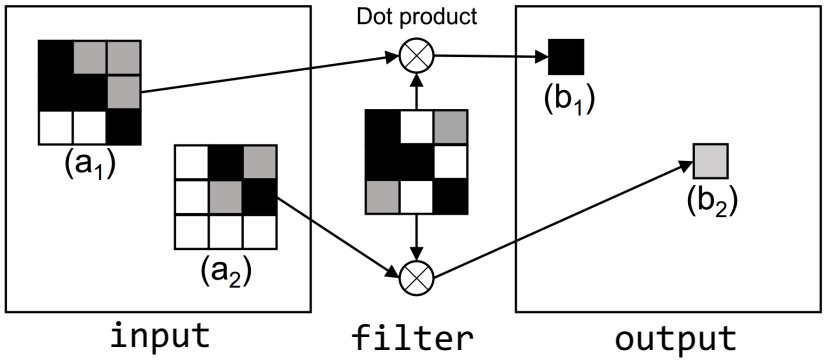
- 뉴런이 모여서 feature map을 이룸
  - 일반적으로 (높이, 너비, 채널)의 레이아웃을 가짐

- Feature map이 연산 레이어를 통과하여 새로운 feature map이 생성되고 이것이 다음 연산 레이어의 입력이 됨
  - 연산에 따라 레이아웃이 달라지기도 함



Convolution

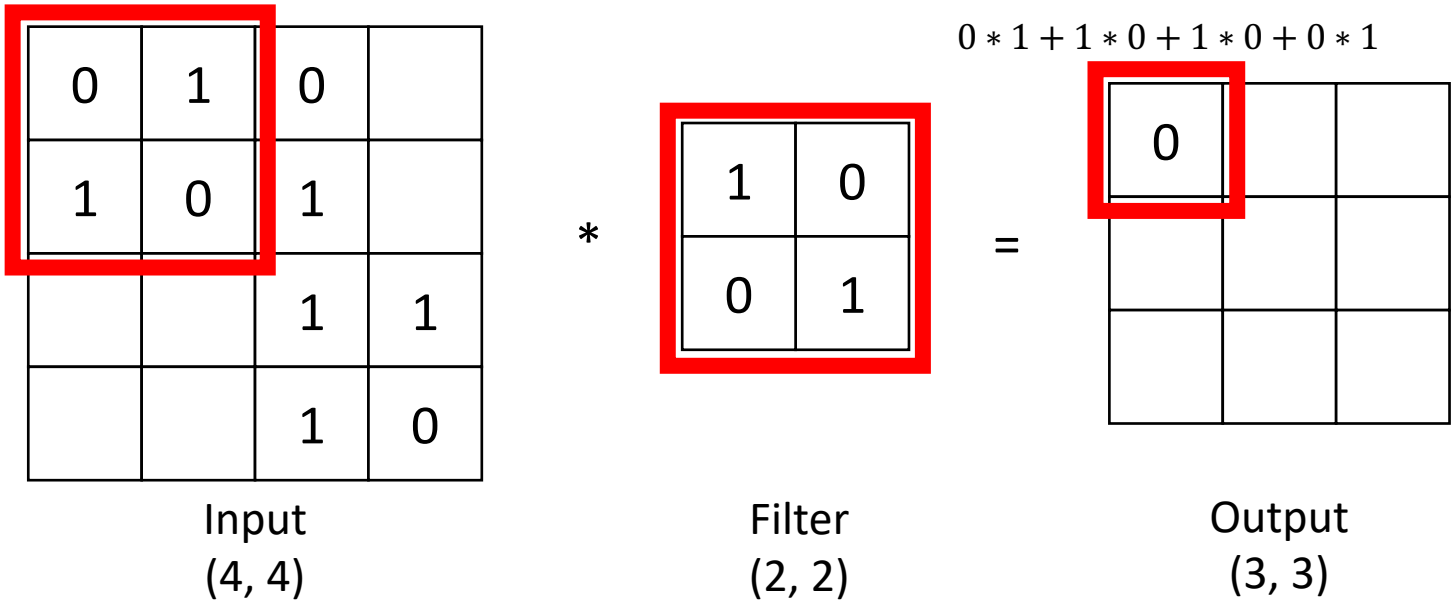
- 이미지 내에서 필터와 유사한 패턴을 찾아내는 연산
  - Input의 특정 영역이 필터와 유사하다면, 대응되는 output의 값이 높게 나옴




```
function convolution(input, filter, output):
  for i in image_height
    for j in image_width
      for fi in filter_height
        for fi in filter_width
          output[i][j] += filter[fi][fj] * input[i + fi][j + fj]
```

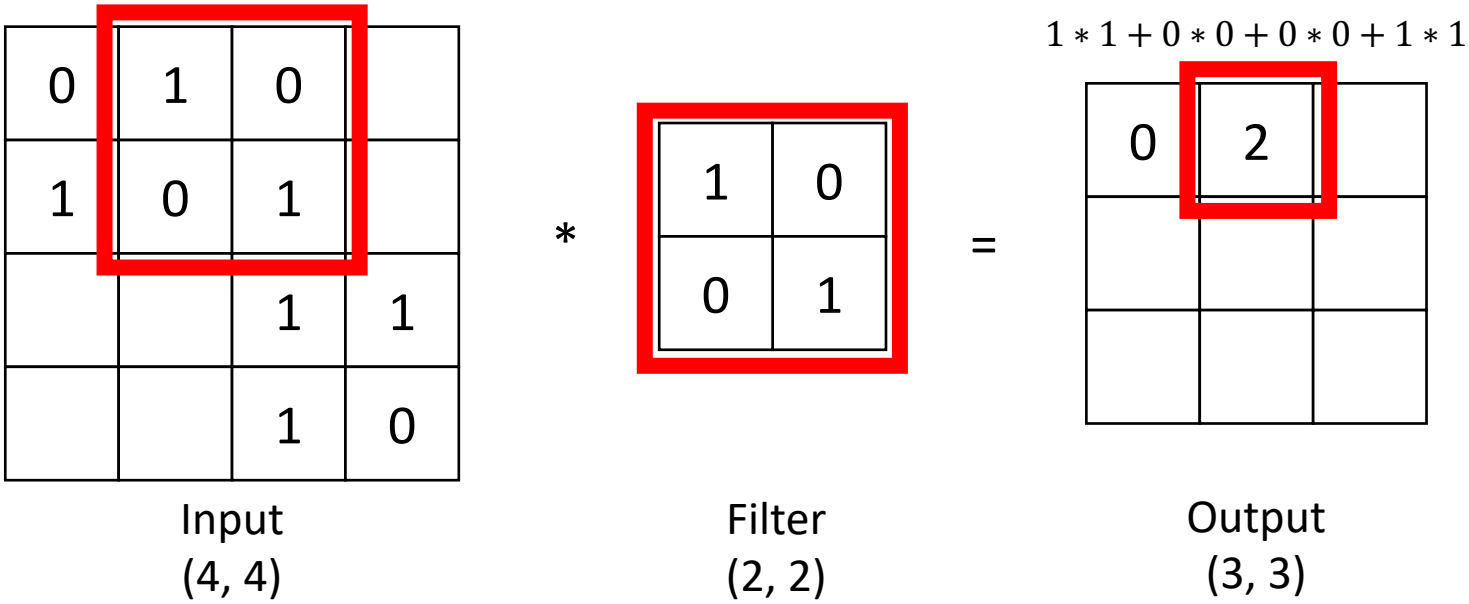
Convolution

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



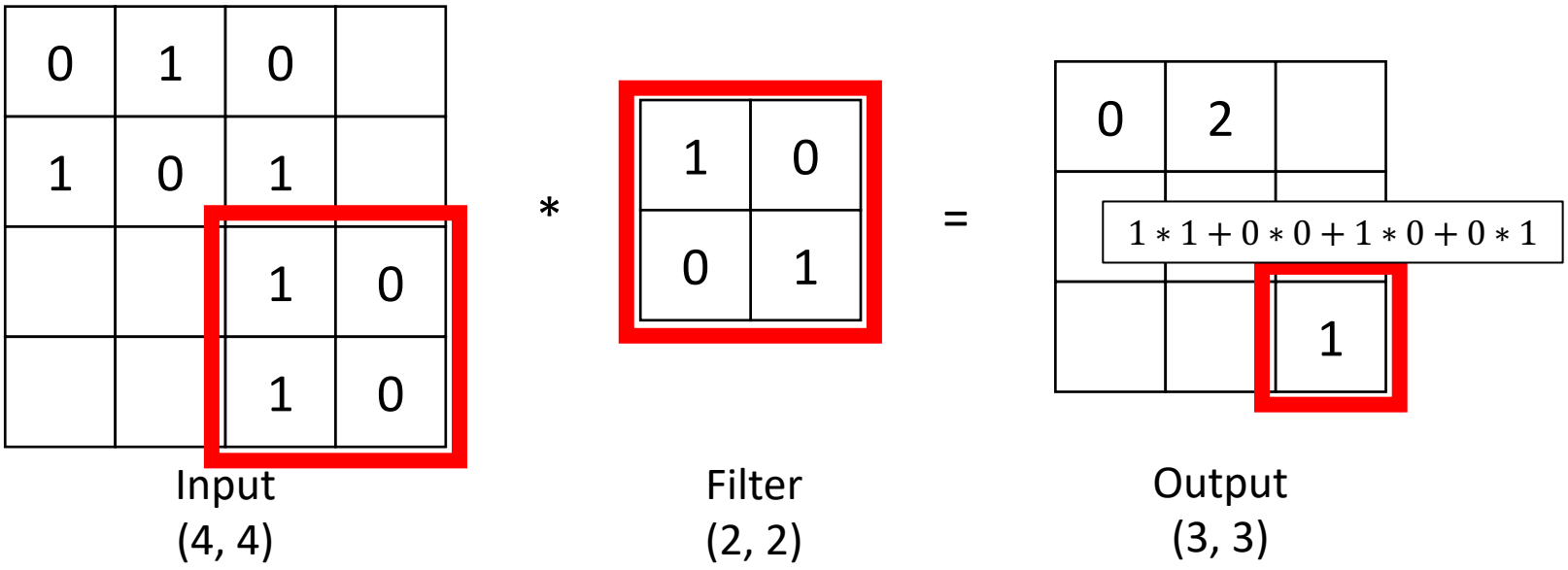
Convolution

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



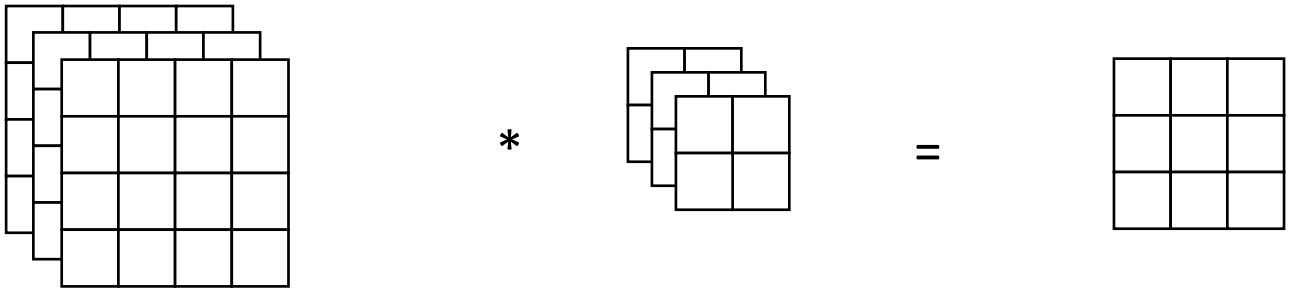
Convolution

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



Convolution

- 입력의 채널이 여러 개라면?



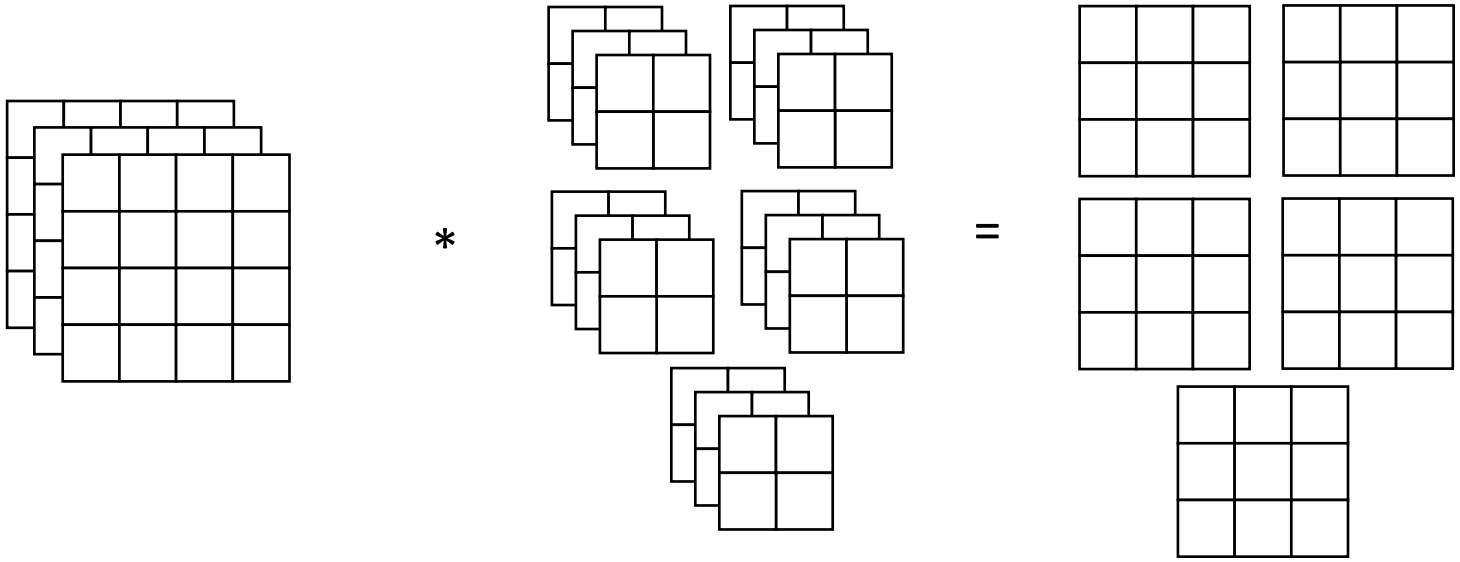
Input
(4, 4, **3**)

Filter
(2, 2, **3**)

Output
(3, 3)

Convolution

- 출력의 채널이 여러 개라면?



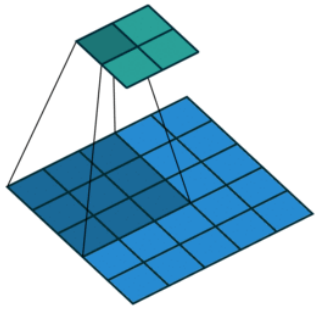
Input
(4, 4, 3)

Filter
(2, 2, 3, **5**)

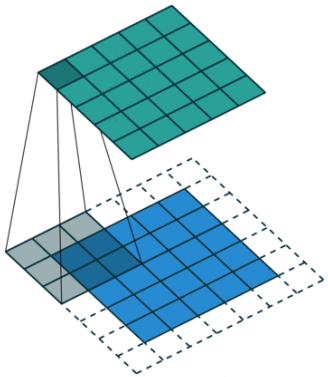
Output
(3, 3, **5**)

Convolution

- Stride
  - 필터가 움직이는 간격



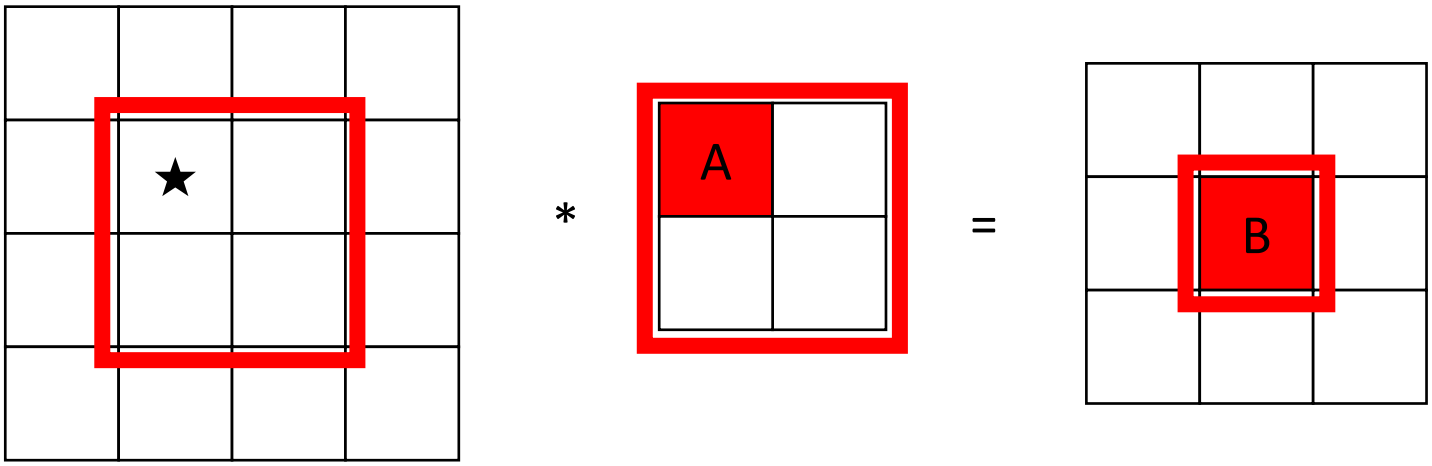
- Pad
  - 입력 가장자리에 0으로 채운 행/열을 추가하여 출력 크기 조절



Source: https://github.com/vdumoulin/conv_arithmetic

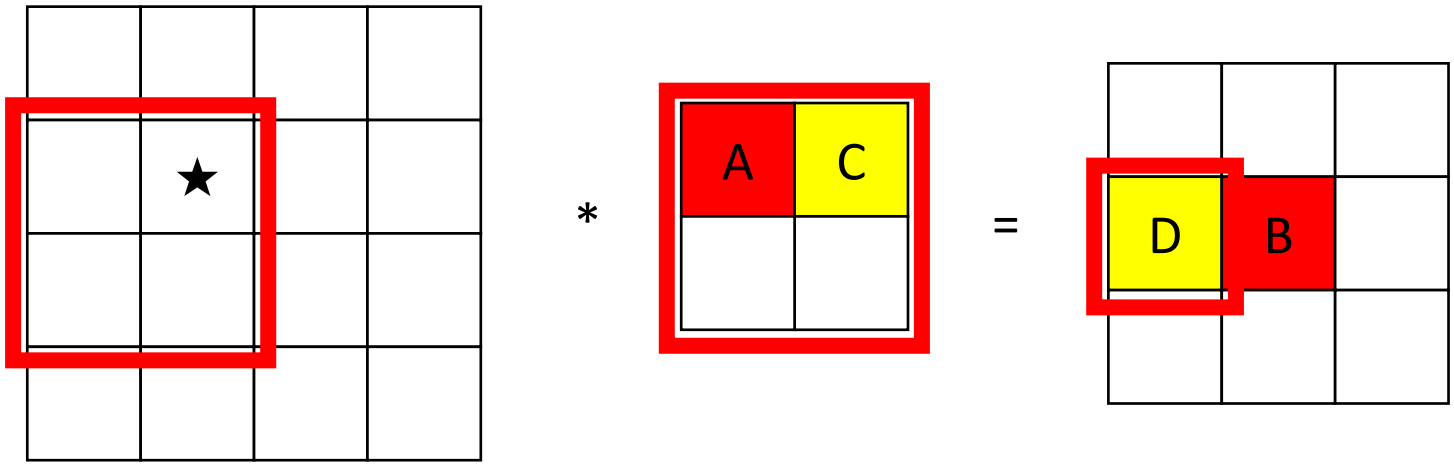
Transposed Convolution

- 입력(★)에 곱해진 필터 위치(A)와 그 값이 더해진 출력 위치(B)를 찾음



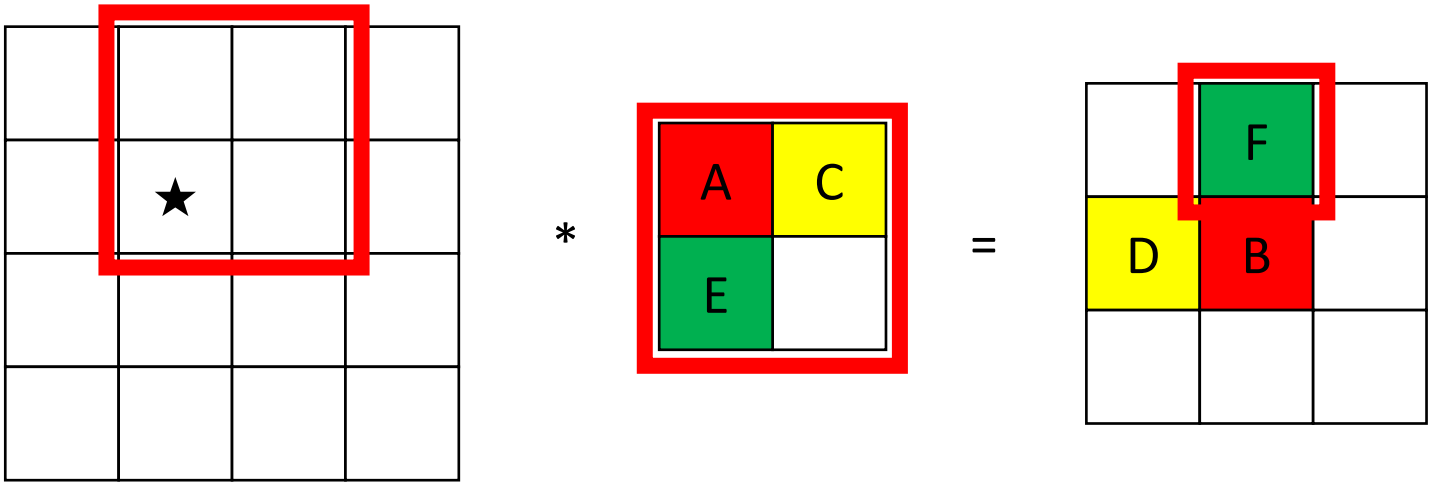
Transposed Convolution

- 입력(★)에 곱해진 필터 위치(C)와 그 값이 더해진 출력 위치(D)를 찾음



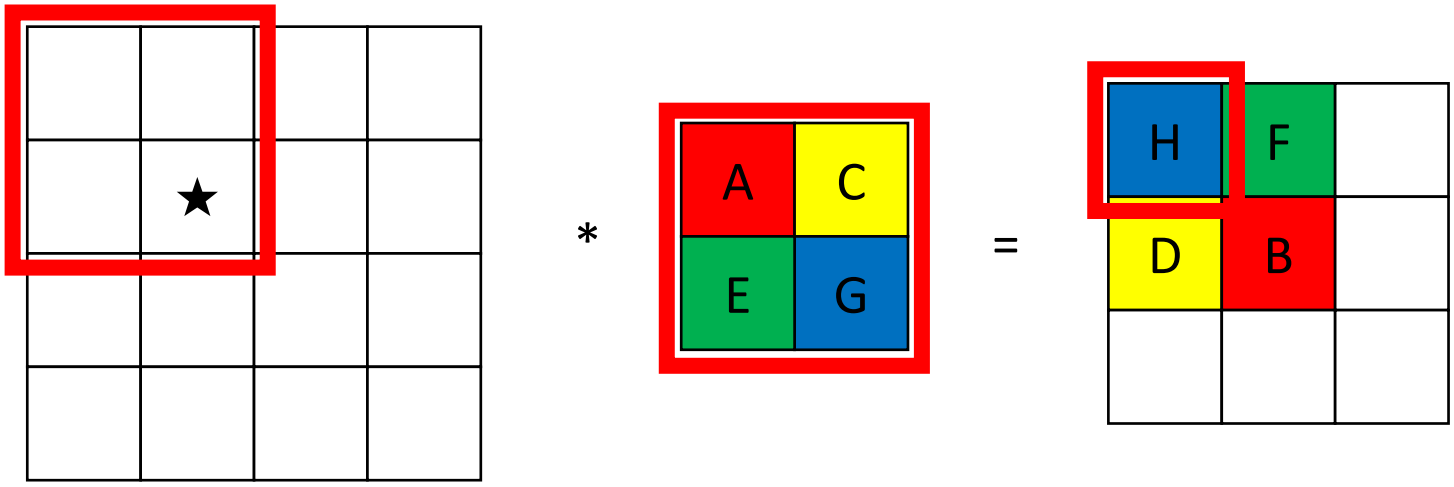
Transposed Convolution

- 입력(★)에 곱해진 필터 위치(E)와 그 값이 더해진 출력 위치(F)를 찾음



Transposed Convolution

- 입력(★)에 곱해진 필터 위치(G)와 그 값이 더해진 출력 위치(H)를 찾음



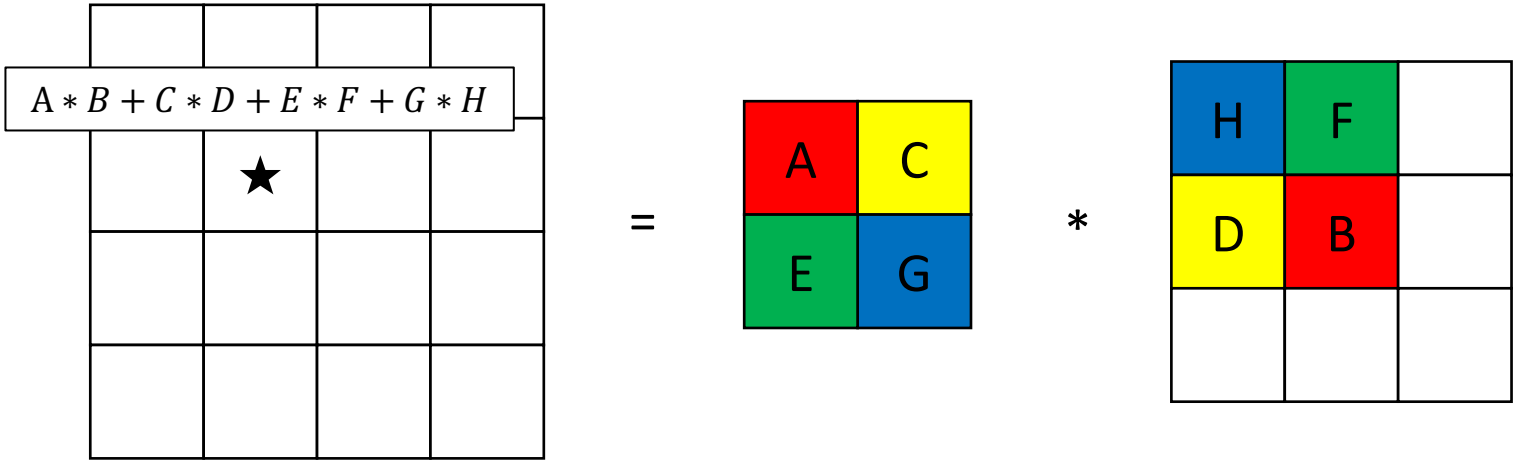
Input
(4, 4)

Filter
(2, 2)

Output
(3, 3)

Transposed Convolution

- 대응하는 값끼리 내적
  - 어렵지만, 코드 및 시각화 자료를 보면 이해가 갈 것



Output
(4, 4)

Filter
(2, 2)

Input
(3, 3)

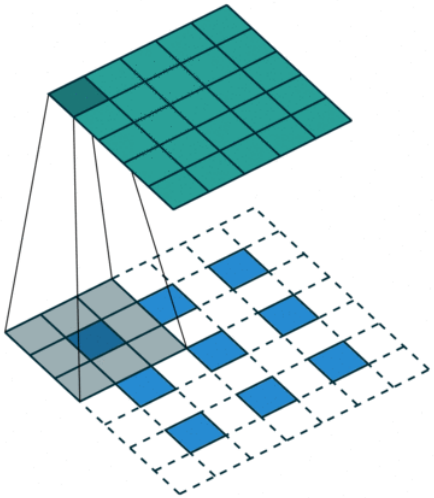
Transposed Convolution

- 대응하는 값끼리 내적
  - 어렵지만, 코드 및 시각화 자료를 보면 이해가 갈 것

Output (5, 5)

Filter (3, 3)

Input (3, 3)

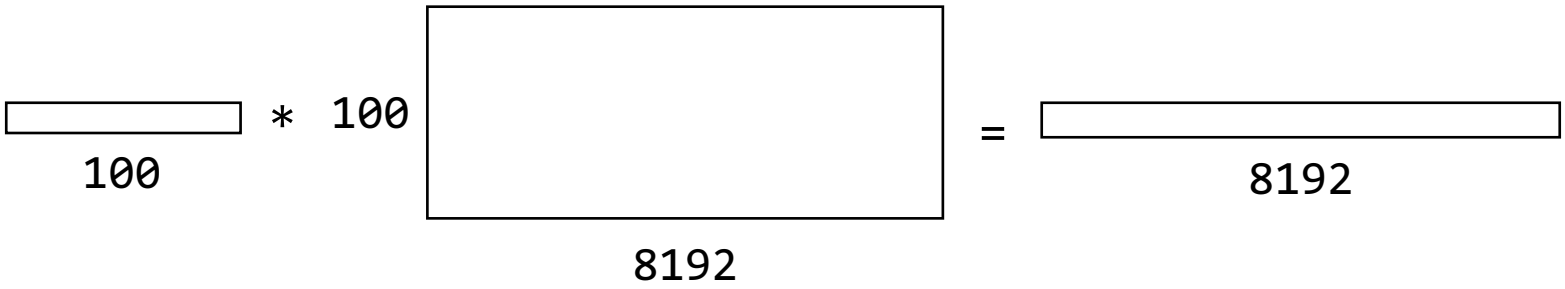


Stride 1
Pad 1

Source: https://github.com/vdumoulin/conv_arithmetic

Linear Projection

- Input을 벡터, filter를 행렬로 한 행렬-벡터 곱



Batch Normalization

- 입력이 뉴럴 네트워크를 통과하면서 평균이나 분산에 편향이 생기는 것을 완화시켜주는 연산
  - $E[x], V[x]$: 훈련 데이터로부터 얻은 평균과 분산
  - β, γ : 훈련으로 익힌 bias와 scale 값
  - ε : 0으로 나누는 것을 방지하기 위한 값

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

ReLU, tanh

- 단순히 각 원소에 해당 연산을 적용하는 레이어
  - ReLU : y = max(x, 0)
  - tanh : y = tanh(x)

프로젝트 목표

- 주어진 순차 코드를 CPU, GPU를 활용하도록 병렬화
  - GPU의 경우 CUDA만 사용

- celebA dataset으로 사전에 훈련된 네트워크 파일 제공

프로젝트 뼈대 코드

- facegen.h
- main.c 파일 입출력, 수행 시간 측정 등을 담당
- facegen_seq.c 얼굴을 생성하는 순차 코드
- facegen_parallel.cu 얼굴을 생성하는 병렬화 코드
- Makefile
- network.bin 훈련된 네트워크 파일
- inputX.txt 예제 입력 파일 (X=1,2,…)
- answerX.txt 예제 정답 파일 (X=1,2,…)
- compare_result 결과 비교용 유틸리티

프로젝트 뼈대 코드

- 컴파일 방법
  - 주어진 Makefile을 사용
  - `make seq`
    - 순차 버전 코드를 컴파일하여 실행 파일(`facegen_seq`) 생성
  - `make parallel`
    - 병렬화 버전 코드를 컴파일하여 실행 파일(`facegen_parallel`) 생성

프로그램 실행

- 간단한 테스트 외에는 SLURM 커맨드를 이용하여 계산 노드(a08-a11)에서 실행할 것!

- `./facegen_seq <네트워크> <입력데이터> <출력데이터> <출력이미지>`
  - 예: `./facegen_seq network.bin input1.txt output1.txt output1.bmp`
- `./facegen_parallel <네트워크> <입력데이터> <출력데이터> <출력이미지>`
  - 예: `./facegen_parallel network.bin input1.txt output1.txt output1.bmp`

- 실행 시
  - `.stdout`으로 수행시간이 출력
    - 파일 I/O에 필요한 시간, 프로그래밍 모델 환경을 위한 초기화 시간은 제외되며, 연산 부분의 성능만 포함
    - stdout에 출력되는 시간을 기준으로 프로젝트 결과를 평가
    - 출력데이터에는 네트워크의 output을 출력, 결과 검증에 사용됨
    - 출력이미지는 참고용으로 채점 시에는 사용되지 않음

프로그램 실행

- SLURM을 통한 프로그램 실행 예시
  - 순차코드 실행
    - `salloc --nodes=1 --ntasks-per-node=1 --cpus-per-task=2 --partition=shpc mpirun ./facegen_seq network.bin input1.txt output1.txt output1.bmp`
  - 1노드를 사용하여 실행
    - `salloc --nodes=1 --ntasks-per-node=1 --cpus-per-task=64 --gres=gpu:4 --partition=shpc mpirun ./facegen_parallel network.bin input1.txt output1.txt output1.bmp`
  - 2노드를 사용하여 실행
    - `salloc --nodes=2 --ntasks-per-node=1 --cpus-per-task=64 --gres=gpu:4 --partition=shpc mpirun ./facegen_parallel network.bin input1.txt output1.txt output1.bmp`

유틸리티

- compare_result
  - 연산 순서에 따라 결과 값에 오차가 발생할 수 있음
  - 절대오차, 상대오차 1e-3까지 허용하여 두 출력데이터가 같은지 비교해줌
  - ./compare_result output1.txt answer1.txt

- eog
  - image viewer로 로컬에 X server가 설치되어 있다면 편리하게 출력 이미지를 볼 수 있음
  - eog output1.bmp
  - 없다면 scp, sftp 등을 이용하여 출력 이미지를 로컬로 복사하여 이미지 뷰어로 볼 것

프로젝트 구현 시 주의 사항

- `main.c, facegen_parallel.cu, Makefile`의 컴파일 옵션만 수정 가능
  - 그 외 파일은 수정 금지
- 반드시 모든 연산을 CUDA 커널로 구현할 필요는 없음
- 알고리즘 자체는 수정하지 말 것
- 배운 병렬화 모델 모두 사용 가능
  - Pthreads, MPI, OpenMP 등
  - CUDA를 사용하므로 OpenCL은 예외
- 별도의 library 사용 불가
  - Intel MKL, cuBLAS, cuDNN 등
  - 애매하면 조교에게 문의

제출 형태

- shpcXXX_project.zip (또는 다른 압축파일 확장자)
  - report.pdf 및 소스 코드 디렉토리
  - **디렉토리에는 코드만 남기고 용량이 큰 바이너리나 이미지는 제거**

제출 방법

- shpcXXX_project.zip을 ETL로 제출
- **프로젝트는 grace day 사용 불가**
- **제출 기한 : 12월 20일 (월) 23:59**

평가

- 보고서 (20%)
  - 병렬화 방법에 대한 설명
  - 자신이 측정한 성능
    - 조교가 측정한 성능과 차이가 나는 경우 2차 확인을 하기 위해서

평가

- 소스 코드 (80%)
  - 성능 기반 상대 평가
  - 수강생 중 가장 좋은 성능을 만점으로 하고, 성능이 2배 떨어질 때마다 10% 감점
  - e.g., 1등 프로그램이 1초가 걸린다면, 8초가 걸리는 프로그램은 70점
  - (적당히 큰 8의 배수)개의 이미지로 성능을 측정할 예정
  - **채점 시 SLURM 커맨드의 `ntasks-per-node` 값을 1로 설정하여 채점함**
    - **`ntasks-per-node` 값을 1보다 큰 값을 사용하도록 프로그래밍 한 경우 보고서에 실행법을 별도로 명시할 것**