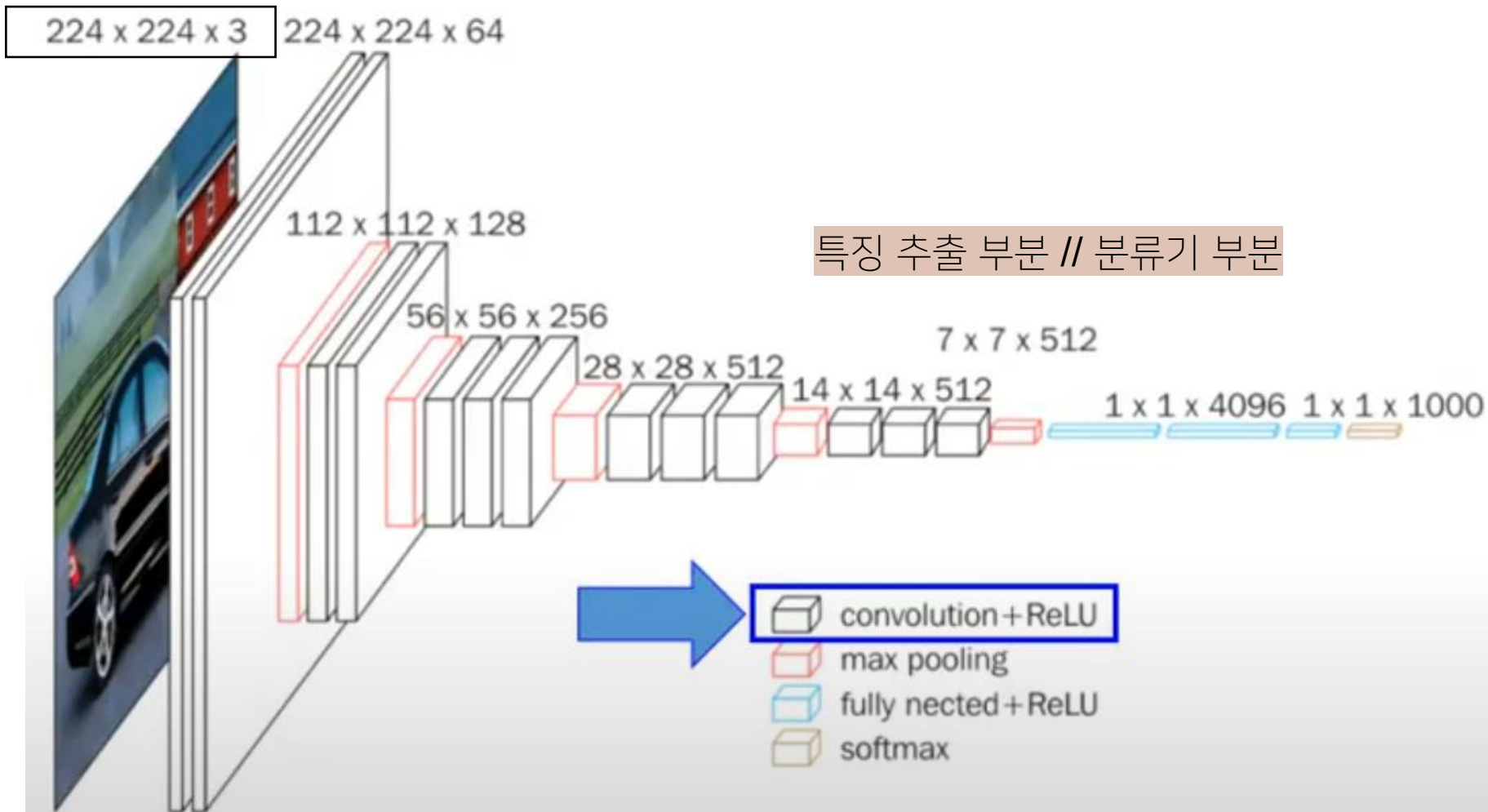


4번째 미팅발표

1. CNN - convolution layer form (+입출력 데이터의 채널)
2. Fully connected layer 들어가기 전에 flatten 설명
3. BeatGAN, FRSKD model 코드 리뷰
4. 결합할 부분 : FRSKD의 knowledge distillation 코드 리뷰

CNN

이미지 크기 224x224
컬러 RGB로 3채널

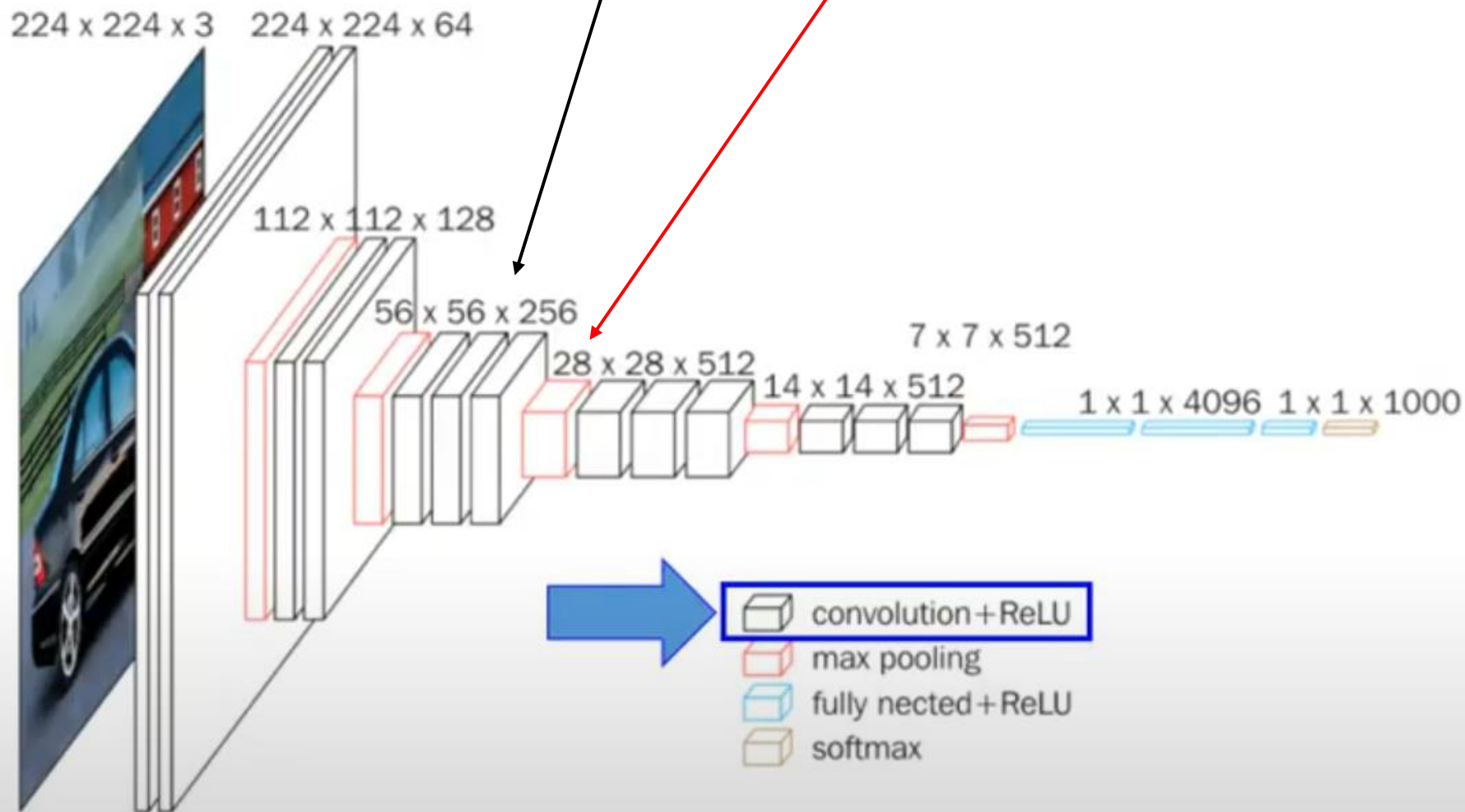


Output	Ground Truth (정답지)	
0.0001	0	오리
0.0009	0	고양이
0.0014	0	사람
0.9274	1	자동차
0.0001	0	사과
⋮	⋮	⋮
0.0001	0	연필
nx1	nx1	

Loss 계산 후 Update
(ex. Cross entropy loss)

CNN

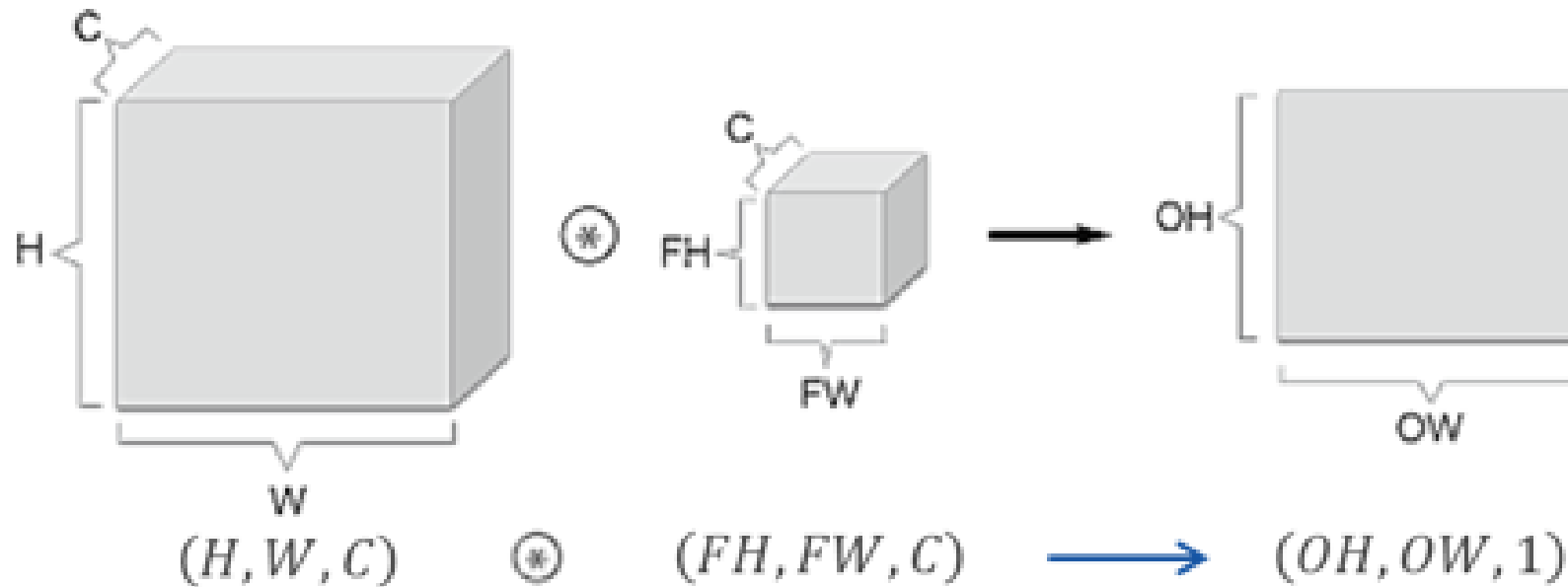
특징 추출 부분 : 컨볼루션 레이어 + 풀링 레이어 반복 구성



Output	Ground Truth (정답지)	
0.0001	0	오리
0.0009	0	고양이
0.0014	0	사람
0.9274	1	자동차
0.0001	0	사과
⋮	⋮	⋮
0.0001	0	연필
nx1	nx1	

Loss 계산 후 Update
(ex. Cross entropy loss)

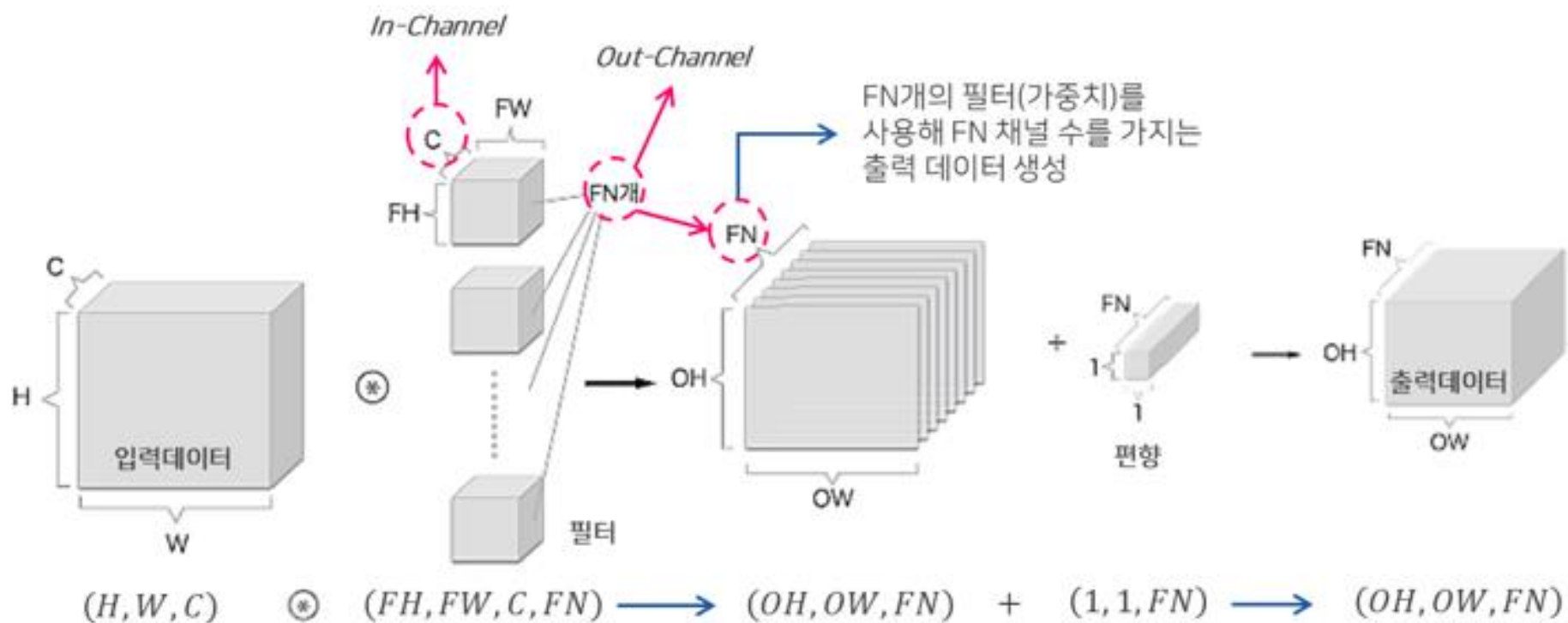
3차원 데이터의 합성곱



3차원의 합성곱 연산을 볼록으로 생각!

3차원 데이터의 모양은 (높이, 너비, 채널) = (Height, Width, Channel) 순으로 표현
이때 **필터의 채널 수(C)**은 **입력 데이터의 채널 수**와 같아야 하며
그림과 같이 **필터의 개수**가 1개인 경우, **출력 데이터의 채널 수**는 1이 된다.

3차원 데이터의 합성곱



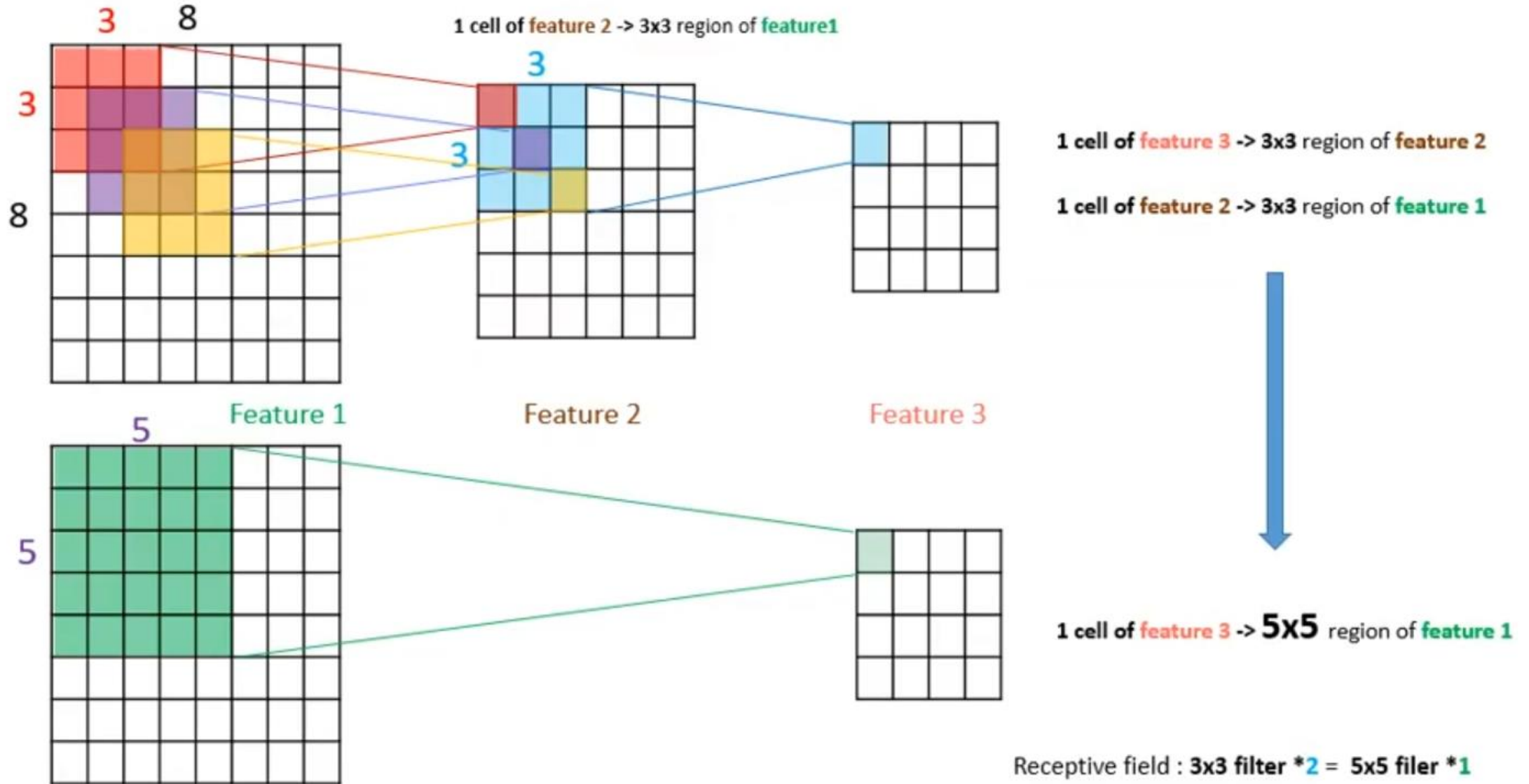
필터가 이번에 1개가 아닌, 다수의 **FN개**가 있다고 하자
 이 경우 **출력 데이터의 채널 수**는 **FN개**가 된다.
 합성곱 연산에도 편향이 쓰여서 그림과 같이 편향을 더해주면 맨 오른쪽 그림처럼 된다.

Multi Channel CNN 정리

- 입력 데이터의 채널 수와 필터의 채널 수는 같아야 한다.
(3채널 데이터가 입력 → 필터도 3채널)
- 입력 데이터의 채널 수와 관계없이
필터의 개수가 출력 데이터의 채널 수로 결정된다.

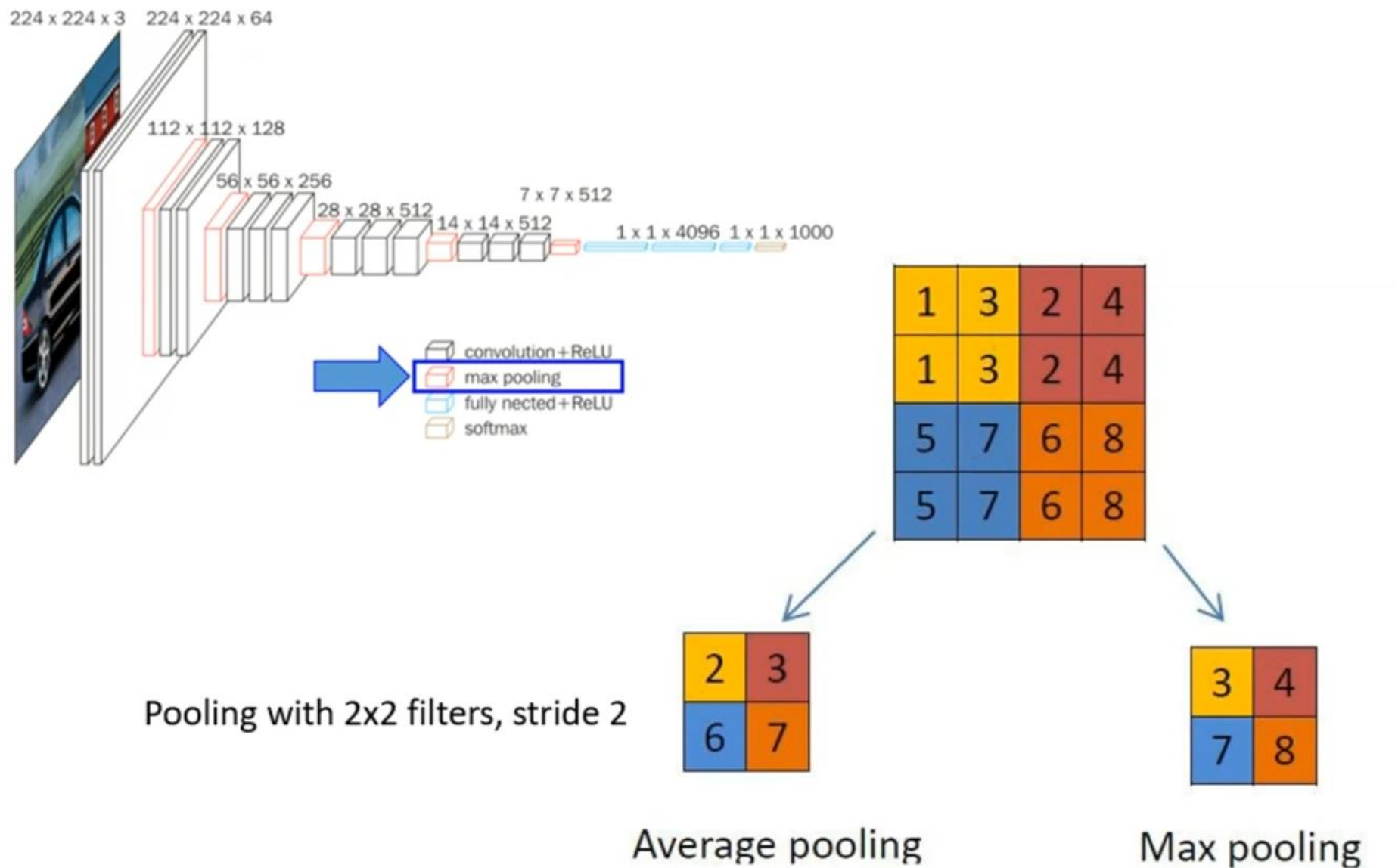
CNN (Receptive field)

filter가 한번에 보는 영역으로,
일반적인 3x3 filter size는 receptive field가 3x3 이미지
Receptive field가 늘어난다는 것은 output계산할 때 사용하는 정보의 양이 증가



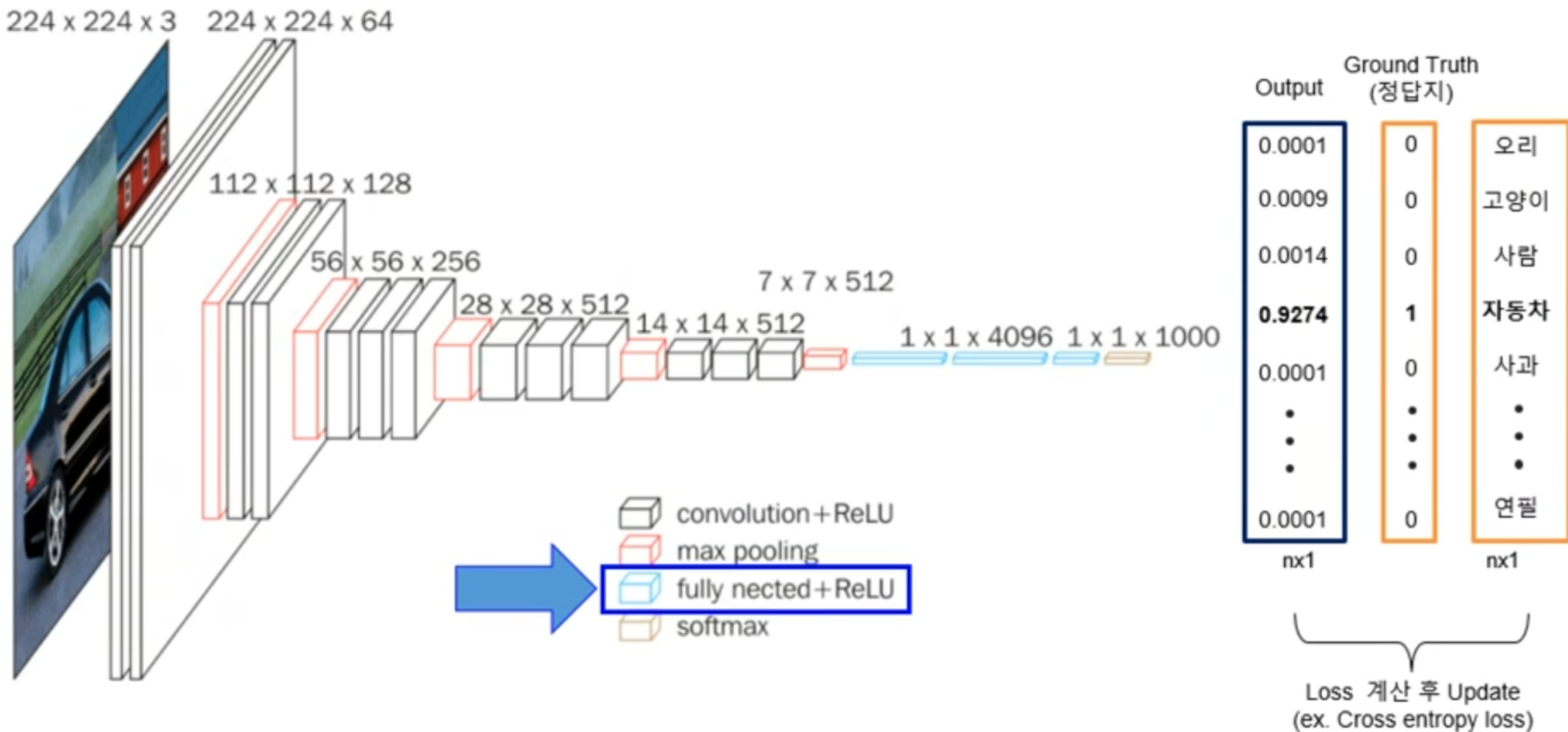
Pooling

차레로 처리되는 데이터의 크기를 줄인다.

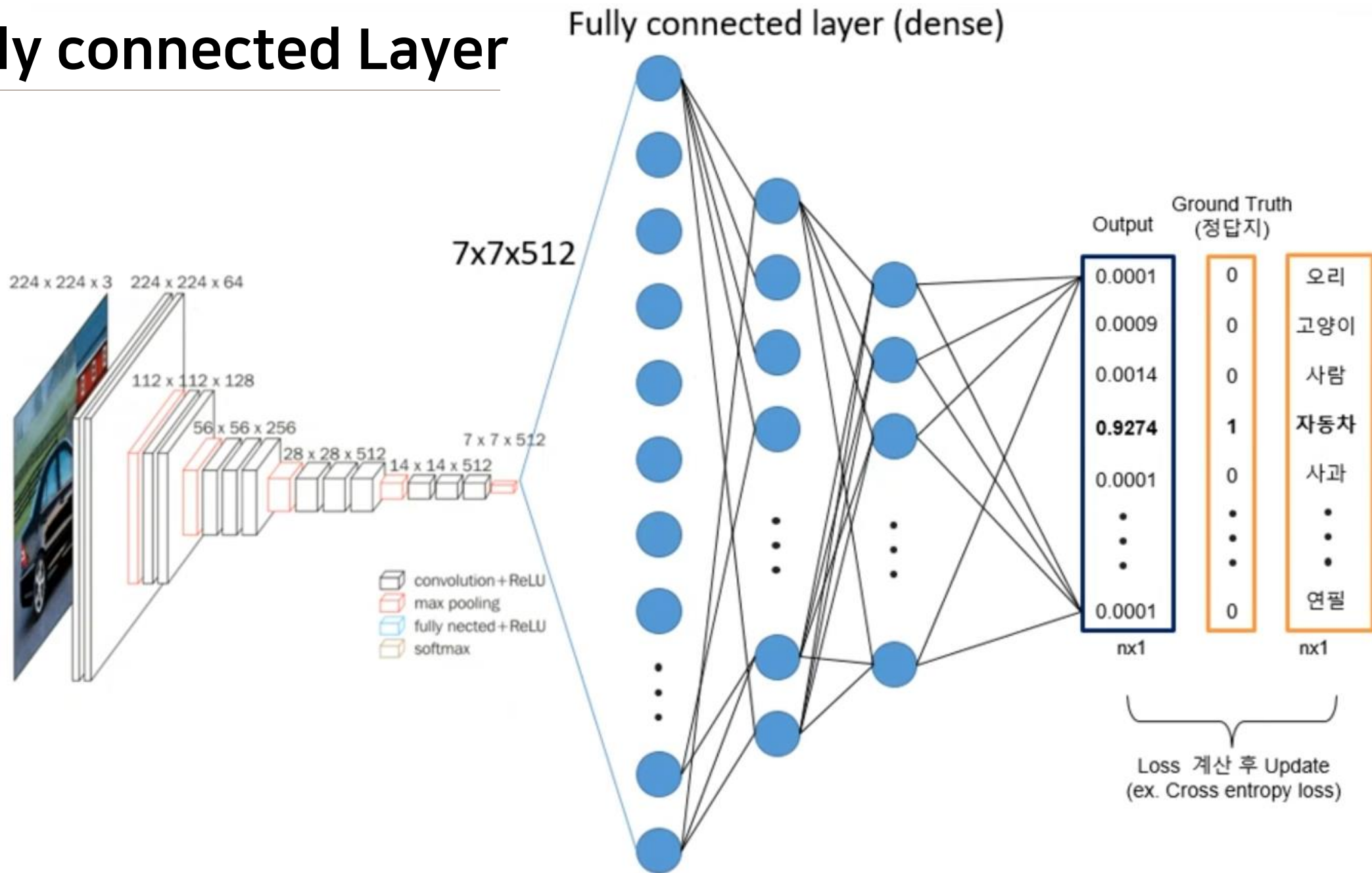


Flatten Layer

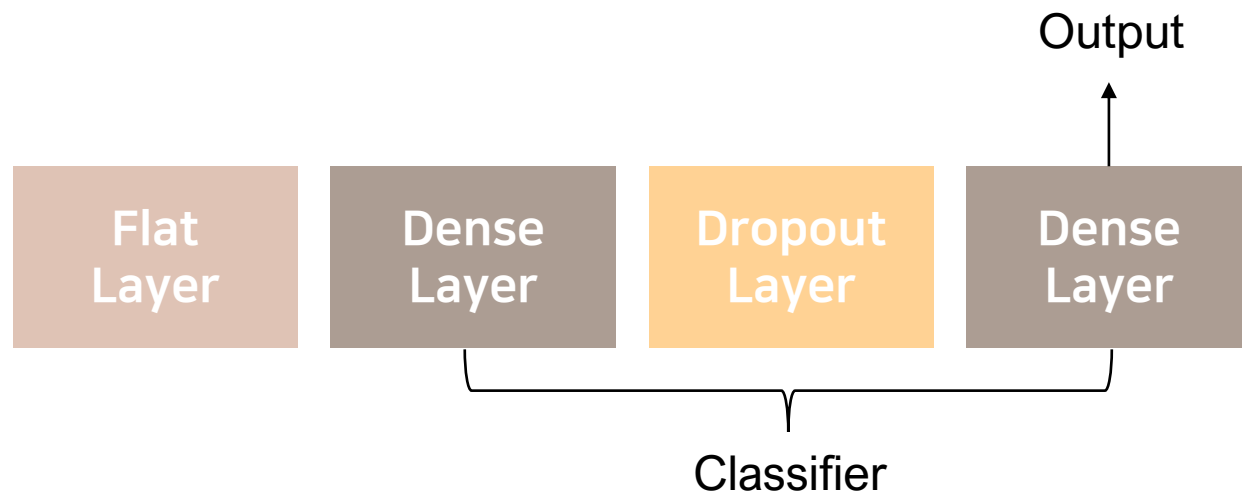
컨볼루션 레이어, 풀링 레이어 반복적으로 거치면 주요 특징만 추출되며,
이는 다차원의 데이터로 이루어져 있지만 (이미지는 3차원)
분류를 위한 학습 레이어에서는 **1차원 데이터**로 바꿔 학습이 되어야 한다.



Fully connected Layer



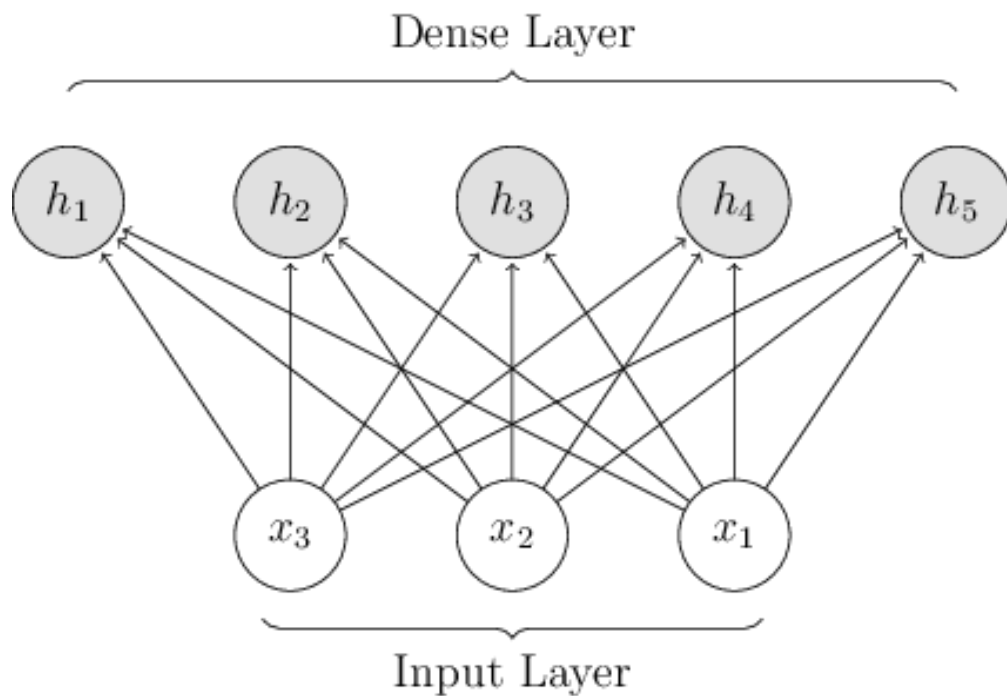
Classifier 구성



Dense Layer

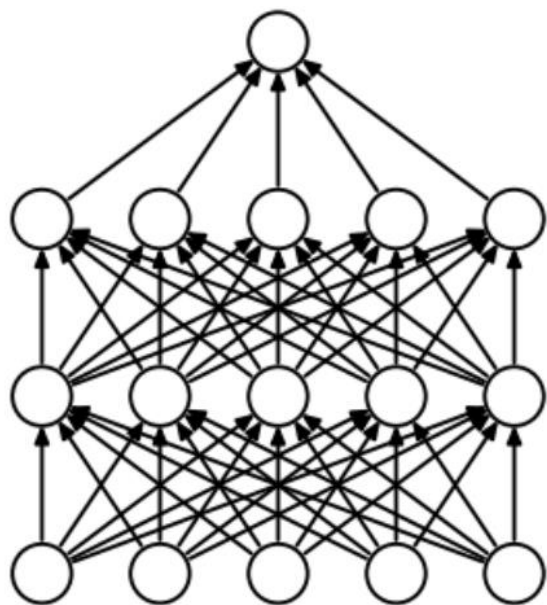
입력과 출력을 모두 연결해준다.

예를 들어 입력 뉴런이 3개, 출력 뉴런이 5개라고 할 때,
총 연결선은 $3 \times 5 = 15$ 개가 된다.

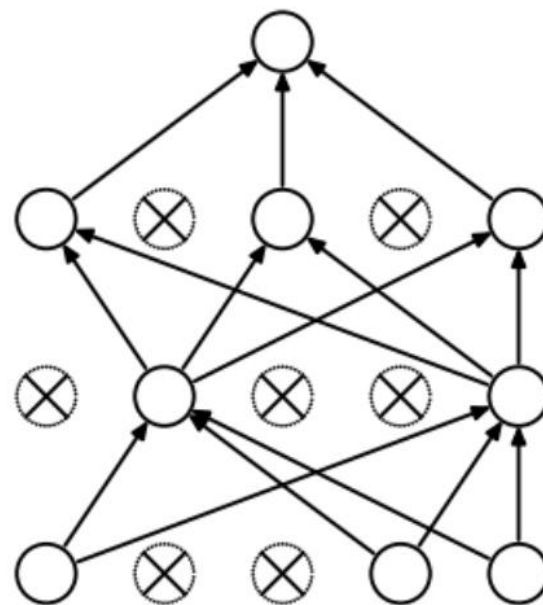


Dropout Layer

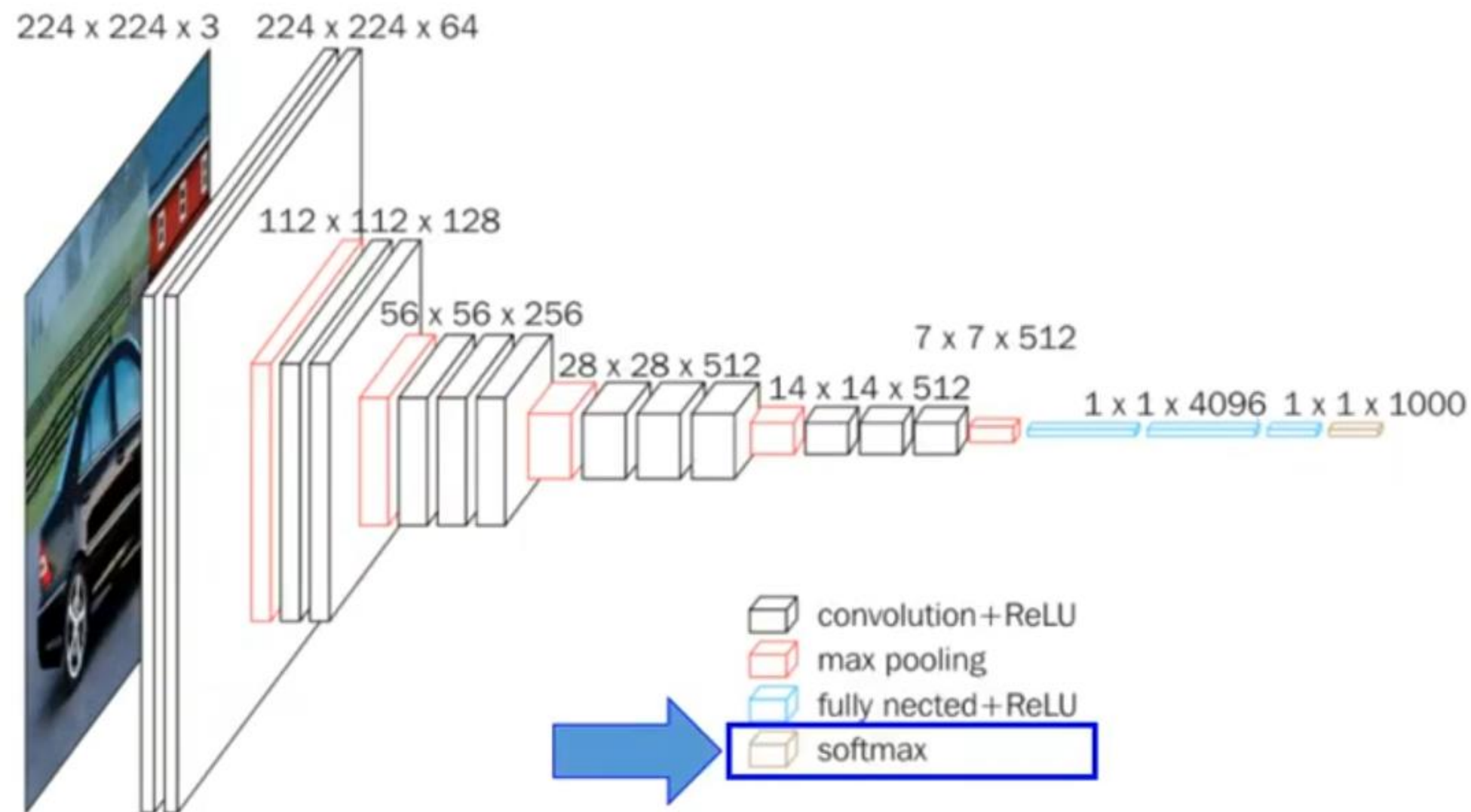
네트워크가 **과적합**되는 경우를 **방지**하기 위해 만들어진 레이어로,
학습 시 **무작위**로 부분적인 뉴런을 **제거**하는 방식이다.
테스트에서는 모든 값을 포함하여 계산한다.



(a) Standard Neural Net



(b) After applying dropout.



Output	Ground Truth (정답지)	
0.0001	0	오리
0.0009	0	고양이
0.0014	0	사람
0.9274	1	자동차
0.0001	0	사과
⋮	⋮	⋮
0.0001	0	연필
nx1	nx1	

Loss 계산 후 Update
(ex. Cross entropy loss)

Softmax

Softmax :

Class 분류를 위해 마지막 단계에서 출력값에 대한 정규화를 해주는 함수
특정 레이블에 대한 확률을 구할 수 있다.

구한 확률 값은 해당 데이터의 종류가 무엇인지 추론하게 한다.

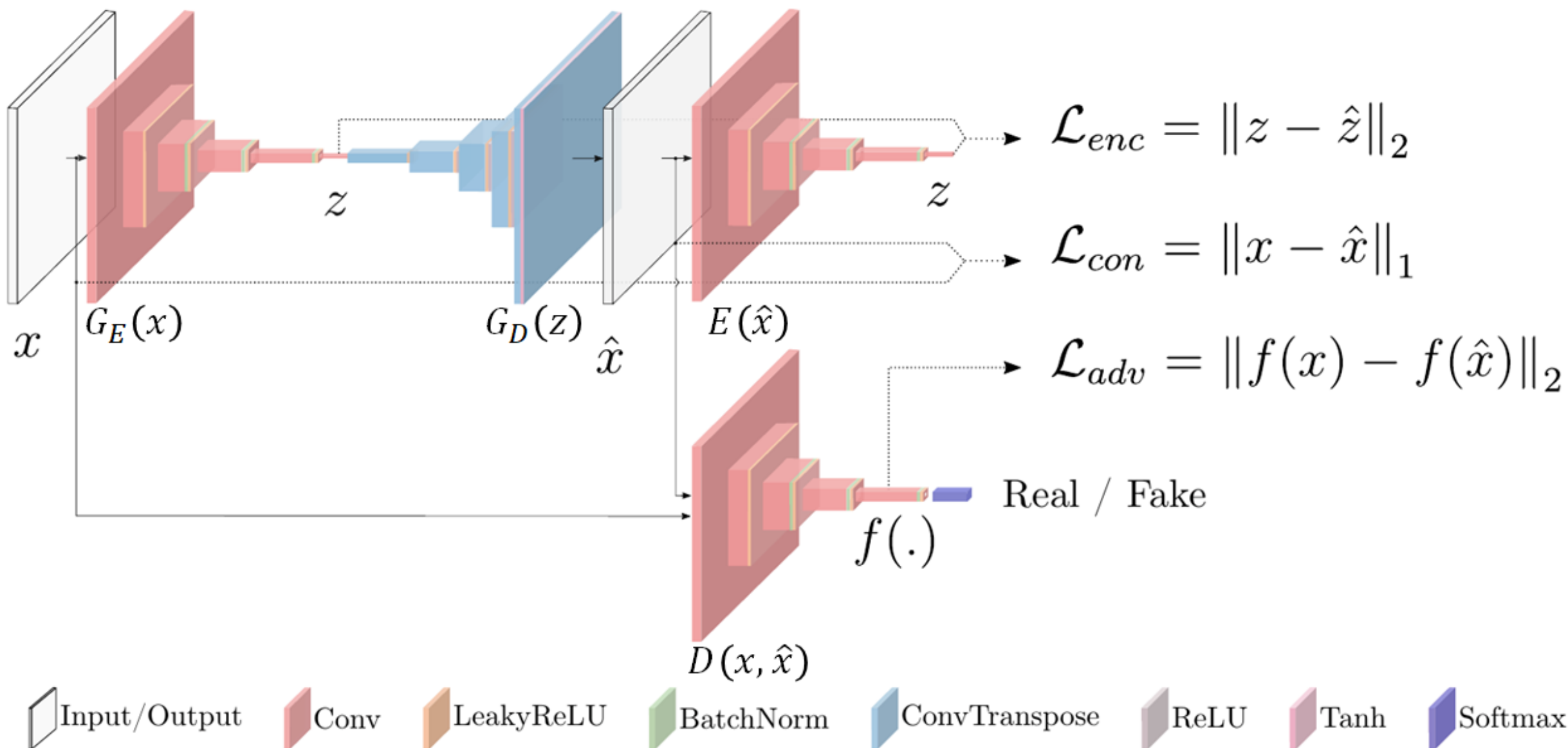
출력값들의 총 합은 항상 1

Softmax 간단한 예제

```
1  import numpy as np
2  import pandas as pd
3
4  a = np.random.uniform(low=0.0, high=10.0, size=3)
5
6  def softmax(arr):
7      m = np.argmax(arr)
8      arr = arr - m
9      arr = np.exp(arr)
10     return arr / np.sum(arr)
11
12  y = softmax(a)
13
14  print(y)
15
16  print(y.sum())
```

```
$ python softmax_ex.py
[0.01737411 0.91275863 0.06986726]
1.0
```

예시) GAN기반 모델 - GANomaly



Setting, 코드실행결과

1. RTX3090 setting
 2. Local setting
 3. 로컬에서 BeatGAN 코드실행결과 및 비교표
 4. 로컬에서 FRSKD 코드실행
-

RTX3090 setting

OS : Ubuntu 18.04.3 LTS bionic x86_64

VGA : RTX 3090

VGA Driver : 460.27.04

CUDA ver : 11.0

CUDNN ver : 8.0.4

Anaconda : Anaconda3-2021.05-Linux-x86_64.sh

Python ver : 3.8

PyTorch ver : 1.7.1

RTX3090 setting



```
ERROR: cuda failure (no CUDA-capable device is detected) in error_util.h:91
Aborting...
```

결론 : 다시 로컬에서 하기로 했습니다.

코드서버... 가끔 무한 로딩되는 에러가 걸리면 창 닫히고 새로 띄우는데,
이때 가상환경 세팅해둔 것이 모두!! 날라가고 처음부터 전부 설치해야 했습니다..
(Reloading VS code kills the terminal, it's like closing a terminal and opening a new one.)

cuda나 conda env, zshrc 등등 전부 /usr/local에 설치되었는데.
아마도 볼륨에 경로 지정하여 설치해야 하지 않았을까 합니다. (하지만 이마저도 에러!)
가상환경 세팅, cuda, cudnn, pytorch 설치로 세팅을 전부 완료해놓아도
cuda인식이 불가능한 에러는 결국 고치지 못했습니다....

Local setting

OS : Window10

VGA : GTX1660super

VGA Driver : 472.12

CUDA ver : 11.0

CUDNN ver : 8.0.5

Anaconda : Anaconda3-2021.05-Windows-x86_64.exe

Python ver : 3.7.6

PyTorch ver : 1.7.1

VS Code (Terminal : Git Bash)

Local setting

```
(py37) C:\#Windows#system32>python
Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.cuda.is_available()
True
>>> torch.cuda.current_device()
0
>>> torch.cuda.device_count()
1
>>> torch.cuda.get_device_name(0)
'NVIDIA GeForce GTX 1660 SUPER'
>>> _
```

자세한 설치 방법은 [구글독스](#)에 정리했습니다. (슬라이드 노트/메모 참고)

RTX3090으로 꼭 해보려고 리눅스 기반으로 며칠간 삽질해보다가 계속 **CUDA** 인식을 실패해서 로컬에서 하니... 윈도우는 비교적 아주 쉽게 설치가 완료되었습니다...

로컬에서 BeatGAN코드 실행결과

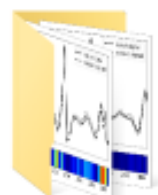
```
#####  
##### Result #####  
ap:0.9040209933041002  
auc:0.9380470672183427  
best th:0.006629863753914833 --> best f1:0.8045258299836235  
(py37)  
edin@DESKTOP-FF4AVFF MINGW64 ~/Desktop/4-2/캡|1/코드/BeatGAN-master  
$ █
```


로컬에서 BeatGAN코드 실행결과

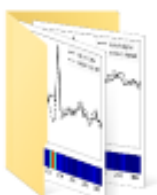
> output > beatgan > ecg > test > 0 >



0 검색



F



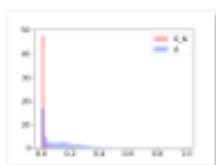
Q



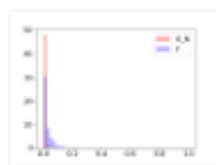
S



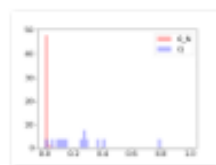
V



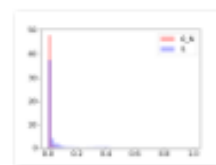
dist0_NA.png



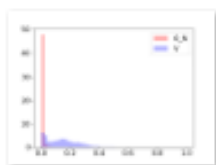
dist0_NF.png



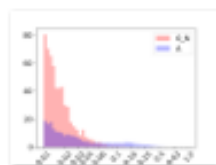
dist0_NQ.png



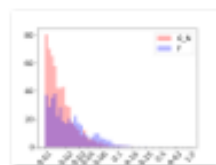
dist0_NS.png



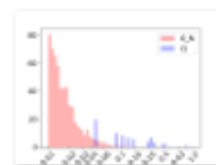
dist0_NV.png



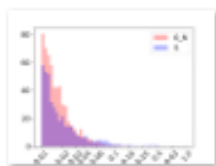
logdist0_NA.png



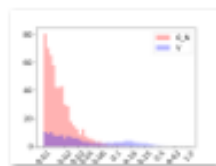
logdist0_NF.png



logdist0_NQ.png



logdist0_NS.png



logdist0_NV.png



res-record.txt

BeatGAN코드 실행결과

Colab 실행결과

Method	AUC	AP
BeatGAN	0.93804699...	0.904020909...
BeatGAN(reproduce)	0.93804698...	0.904020906...

로컬에서 실행결과

Method	AUC	AP
BeatGAN	0.93804706...	0.904020993...
BeatGAN(reproduce)	0.9380470672183427...	0.9040209933041002...



RuntimeError: freeze_support() 해결

윈도우에서 파이토치 돌릴 때 발생할 수 있는 에러로,
해결 방법 중 num_worker의 값을 0으로 세팅

C:\Users\edin\Desktop\4-2캡디1코드\BeatGAN-master\experiments\ecg\data.py	
120	#num_workers=int(opt.workers),
121	num_workers=0,
127	#num_workers=int(opt.workers),
128	num_workers=0,
134	#num_workers=int(opt.workers),
135	num_workers=0,
141	#num_workers=int(opt.workers),
142	num_workers=0,
148	#num_workers=int(opt.workers),
149	num_workers=0,
155	#num_workers=int(opt.workers),
156	num_workers=0,
162	#num_workers=int(opt.workers),
163	num_workers=0,

RuntimeError: freeze_support() 해결

```
experiments > ecg > options.py > Options > __init__
12
13
14 def __init__(self):
15     ##
16     #
17     self.parser = argparse.ArgumentParser(formatter_class=argparse.ArgumentDefaultsHelpFormatter)
18
19     ##
20     # Base
21     self.parser.add_argument('--dataset', default='ecg', help='ecg dataset')
22     self.parser.add_argument('--dataroot', default='', help='path to dataset')
23     self.parser.add_argument('--batchsize', type=int, default=64, help='input batch size')
24     self.parser.add_argument('--workers', type=int, help='number of data loading workers', default=1)
25     self.parser.add_argument('--isize', type=int, default=320, help='input sequence size.')
26     self.parser.add_argument('--nc', type=int, default=1, help='input sequence channels')
27     self.parser.add_argument('--nz', type=int, default=50, help='size of the latent z vector')
28     self.parser.add_argument('--ngf', type=int, default=32)
29     self.parser.add_argument('--ndf', type=int, default=32)
30     self.parser.add_argument('--device', type=str, default='gpu', help='Device: gpu | cpu')
31     self.parser.add_argument('--gpu_ids', type=str, default='0', help='gpu ids: e.g. 0 0,1,2, 0,2. use -1 for CPU')
32     self.parser.add_argument('--ngpu', type=int, default=1, help='number of GPUs to use')
33     self.parser.add_argument('--model', type=str, default='beatgan', help='choose model')
34     self.parser.add_argument('--outf', default='./output', help='output folder')
35
```

```
116     dataloader = {"train": DataLoader(  
117         dataset=train_dataset, # torch TensorDataset format  
118         batch_size=opt.batchsize, # mini batch size  
119         shuffle=True,  
120         #num_workers=int(opt.workers),  
121         num_workers=0,  
122         drop_last=True),  
123     "val": DataLoader(  
124         dataset=val_dataset, # torch TensorDataset format  
125         batch_size=opt.batchsize, # mini batch size  
126         shuffle=True,  
127         #num_workers=int(opt.workers),  
128         num_workers=0,  
129         drop_last=False),  
130     "test_N": DataLoader(  
131         dataset=test_N_dataset, # torch TensorDataset format  
132         batch_size=opt.batchsize, # mini batch size  
133         shuffle=True,  
134         #num_workers=int(opt.workers),  
135         num_workers=0,  
136         drop_last=False),  
137     "test_S": DataLoader(  
138         dataset=test_S_dataset, # torch TensorDataset format  
139         batch_size=opt.batchsize, # mini batch size  
140         shuffle=True,  
141         #num_workers=int(opt.workers),  
142         num_workers=0,  
143         drop_last=False),  
144     "test_V": DataLoader(  
145         dataset=test_V_dataset, # torch TensorDataset format  
146         batch_size=opt.batchsize, # mini batch size  
147         shuffle=True,  
148         #num_workers=int(opt.workers),  
149         num_workers=0,  
150         drop_last=False),  
151     "test_F": DataLoader(  
152         dataset=test_F_dataset, # torch TensorDataset format  
153         batch_size=opt.batchsize, # mini batch size  
154         shuffle=True,  
155         #num_workers=int(opt.workers),  
156         num_workers=0,  
157         drop_last=False),
```

로컬에서 FRSKD코드 실행?..

```
RuntimeError: CUDA out of memory. Tried to allocate 16.00 MiB (GPU 0; 6.00 GiB total capacity; 4.33 GiB already allocated; 0 bytes free; 4.40 GiB reserved in total by PyTorch)
```

batch size를 128에서 64로 줄이니 코드가 돌아갔지만
작업관리자에서 확인해보니 **GPU**는 일하지 않고 **CPU**에서 돌아가는 현상이 있어서 급하게 실행 종료했습니다,,

BeatGAN Review

1. `run_ecg.sh`
 2. `main.py`
 3. `options.py`
 4. `data.py`
 5. `model.py`
-

BeatGAN – run_ecg.sh

```
#!/bin/bash

cd experiments/ecg

test=1
threshold=0.02 # the model, 1 means evaluate the model
fold_cnt=1

dataroot="./dataset/preprocessed/ano0/"
model="beatgan"

w_adv=1
niter=100
lr=0.0001
n_aug=0

outf="./output"

for (( i=0; i<$fold_cnt; i+=1))
do
    echo "#####"
    echo "##### Folder $i #####"
    if [ $test = 0 ]; then
        python -u main.py \
            --dataroot $dataroot \
            --model $model \
            --niter $niter \
            --lr $lr \
            --outf $outf \
            --folder $i
    else
        python -u main.py \
            --dataroot $dataroot \
            --model $model \
            --niter $niter \
            --lr $lr \
            --outf $outf \
            --folder $i \
            --istest \
            --threshold $threshold
    fi
done
```

```
1  #!/bin/bash
2
3  cd experiments/ecg
4
5  test=1    # 0 means train the model, 1 means evaluate the model
6  threshold=0.02
7  fold_cnt=1
8
9  dataroot="./dataset/preprocessed/ano0/"
10 model="beatgan"
11
12 w_adv=1
13 niter=100
14 lr=0.0001
15 n_aug=0
16
17 outf="./output"
18
```

인자값 초기 설정

test=1 : evaluate the model

BeatGAN – run_ecg.sh

```
#!/bin/bash

cd experiments/ecg

test=1
threshold=0.02n the model, 1 means evaluate the model
fold_cnt=1

dataroot="./dataset/preprocessed/ano0/"
model="beatgan"

w_adv=1
niter=100
lr=0.0001
n_aug=0

outf="./output"

for (( i=0; i<$fold_cnt; i+=1))
do
    echo "#####"
    echo "##### Folder $i #####"
    if [ $test = 0 ]; then
        python -u main.py \
            --dataroot $dataroot \
            --model $model \
            --niter $niter \
            --lr $lr \
            --outf $outf \
            --folder $i

    else
        python -u main.py \
            --dataroot $dataroot \
            --model $model \
            --niter $niter \
            --lr $lr \
            --outf $outf \
            --folder $i \
            --istest \
            --threshold $threshold

    fi
done
```

```
19 for (( i=0; i<$fold_cnt; i+=1))
20 do
21     echo "#####"
22     echo "##### Folder $i #####"
23     if [ $test = 0 ]; then
24         python -u main.py \
25             --dataroot $dataroot \
26             --model $model \
27             --niter $niter \
28             --lr $lr \
29             --outf $outf \
30             --folder $i
31
32     else
33         python -u main.py \
34             --dataroot $dataroot \
35             --model $model \
36             --niter $niter \
37             --lr $lr \
38             --outf $outf \
39             --folder $i \
40             --outf $outf \
41             --istest \
42             --threshold $threshold
43     fi
44
45 done
```

main.py로 파이썬 파일 실행
앞서 설정된 인자값 전달

BeatGAN – main.py

```
1 import os
2 os.environ["CUDA_VISIBLE_DEVICES"] = "0"
3 import torch
4 from options import Options
5
6 from data import load_data
7
8 # from dcgan import DCGAN as myModel
9
10
11 device = torch.device("cuda:0" if
12 torch.cuda.is_available() else "cpu")
13
14
15
16
17 opt = Options().parse()
18 print(opt)
19 dataloader=load_data(opt)
20 print("load data success!!!")
21
22 if opt.model == "beatgan":
23     from model import BeatGAN as MyModel
24
25 else:
26     raise Exception("no this model :{}".format(opt.model))
27
28
29 model=MyModel(opt,dataloader,device)
30
31 if not opt.istest:
32     print("##### Train #####")
33     model.train()
34 else:
35     print("##### Eval #####")
36     model.load()
37     model.test_type()
38     # model.test_time()
39     # model.plotTestFig()
40     # print("threshold:{}\tf1-score:{}\tauc:{}".format( th, f1, auc))
```

```
1 import os
2 os.environ["CUDA_VISIBLE_DEVICES"] = "0"
3 import torch
4 from options import Options
5
6 from data import load_data
7
8 # from dcgan import DCGAN as myModel
9
10
11 device = torch.device("cuda:0" if
12 torch.cuda.is_available() else "cpu")
```

- line 2 : 선택한 GPU에서만 메모리 할당하여 코드가 실행되게 함 (로컬에서 현재 0번째가 돌아감)
- line 4, 6 : option.py에서 Options 클래스를,
data.py에서 load_data 클래스를 import함
- line 11~12 : cuda를 사용할 수 있으면 0번째 gpu를 device에 할당

****cuda란** : 엔비디아에서 개발한 GPU 개발툴로, 많은 양의 연산을 동시에 처리할 수 있게 하여 딥러닝을 쉽게 사용가능하게 함

BeatGAN – main.py

```
1 import os
2 os.environ["CUDA_VISIBLE_DEVICES"] = "0"
3 import torch
4 from options import Options
5
6 from data import load_data
7
8 # from dcgan import DCGAN as myModel
9
10
11 device = torch.device("cuda:0" if
12 torch.cuda.is_available() else "cpu")
13
14
15
16
17 opt = Options().parse()
18 print(opt)
19 dataloader=load_data(opt)
20 print("load data success!!!")
21
22 if opt.model == "beatgan":
23     from model import BeatGAN as MyModel
24
25 else:
26     raise Exception("no this model :{}".format(opt.model))
27
28
29 model=MyModel(opt,dataloader,device)
30
31 if not opt.istest:
32     print("##### Train #####")
33     model.train()
34 else:
35     print("##### Eval #####")
36     model.load()
37     model.test_type()
38     # model.test_time()
39     # model.plotTestFig()
40     # print("threshold:{}\tf1-score:{}\tauc:{}".format( th, f1, auc))
```

```
1 opt = Options().parse()
2 print(opt)
3 dataloader=load_data(opt)
4 print("load data success!!!")
5
6 if opt.model == "beatgan":
7     from model import BeatGAN as MyModel
8
9 else:
10     raise Exception("no this model :{}".format(opt.model))
11
12
13 model=MyModel(opt,dataloader,device)
14
15 if not opt.istest:
16     print("##### Train #####")
17     model.train()
18 else:
19     print("##### Eval #####")
20     model.load()
21     model.test_type()
```

- option 설정 : options.py의 Options 클래스에서 정해짐
파이썬의 argparse 모듈을 사용 (자세한 인자설명은 다른 페이지에서)
- 파이토치는 dataloader를 사용하여 dataset 만들어 사용
(자세한건 data.py에서 이어 설명)

BeatGAN – main.py

```
1 import os
2 os.environ["CUDA_VISIBLE_DEVICES"] = "0"
3 import torch
4 from options import Options
5
6 from data import load_data
7
8 # from dcgan import DCGAN as myModel
9
10
11 device = torch.device("cuda:0" if
12 torch.cuda.is_available() else "cpu")
13
14
15
16
17 opt = Options().parse()
18 print(opt)
19 dataloader=load_data(opt)
20 print("load data success!!!")
21
22 if opt.model == "beatgan":
23     from model import BeatGAN as MyModel
24
25 else:
26     raise Exception("no this model :{}".format(opt.model))
27
28
29 model=MyModel(opt,dataloader,device)
30
31 if not opt.istest:
32     print("##### Train #####")
33     model.train()
34 else:
35     print("##### Eval #####")
36     model.load()
37     model.test_type()
38
39 # model.test_time()
40 # model.plotTestFig()
41 # print("threshold:{}\tf1-score:{}\tauc:{}".format( th, f1, auc))
```

```
1 opt = Options().parse()
2 print(opt)
3 dataloader=load_data(opt)
4 print("load data success!!!")
5
6 if opt.model == "beatgan":
7     from model import BeatGAN as MyModel
8
9 else:
10     raise Exception("no this model :{}".format(opt.model))
11
12
13 model=MyModel(opt,dataloader,device)
14
15 if not opt.istest:
16     print("##### Train #####")
17     model.train()
18 else:
19     print("##### Eval #####")
20     model.load()
21     model.test_type()
```

- run_ecg.sh에서 model="beatgan"이라 했으므로, if조건에 부합하여 model.py에서 BeatGAN(MyModel) 클래스를 import한다. 이후에 model 객체를 만든다.
- opt.istest는 option.py의 parser에 의해 true로 설정되어 있다. 따라서 Train없이 Eval로 넘어가서 line20, line21 수행

BeatGAN – options.py

```
import argparse
import os
import torch
```

```
class Options():
    """Options class

    Returns:
    [argparse]: argparse containing train and test options
    """

    def __init__(self):
        self.parser = argparse.ArgumentParser(formatter_class=argparse.
            ArgumentDefaultsHelpFormatter)

        # Base
        self.parser.add_argument('--dataset', default='ecg', help=
            'ecg dataset')
        self.parser.add_argument('--dataroot', default='', help=
            'path to dataset')
        self.parser.add_argument('--batchsize', type=int, default=64, help=
            'input batch size')
        self.parser.add_argument('--workers', type=int, help=
            'number of data loading workers', default=1)
        self.parser.add_argument('--isize', type=int, default=320, help=
            'input sequence size.')
        self.parser.add_argument('--nc', type=int, default=1, help=
            'input sequence channels')
        self.parser.add_argument('--ngf', type=int, default=50, help=
            'size of the latent z vector')
        self.parser.add_argument('--ndf', type=int, default=32, help=
            'number of layers in discriminator')
        self.parser.add_argument('--device', type=str, default='gpu', help=
            'Device: gpu | cpu')
        self.parser.add_argument('--gpu_ids', type=str, default='0', help=
            'gpu ids: e.g. 0 1,2,3, use -1 for CPU')
        self.parser.add_argument('--ngpu', type=int, default=1, help=
            'number of GPUs to use')
        self.parser.add_argument('--model', type=str, default='beatgan',
            help='choose model')
        self.parser.add_argument('--outf', default='./output', help=
            'output folder')

        # Train
        self.parser.add_argument('--print_freq', type=int, default=100,
            help='frequency of showing training results on console')
        self.parser.add_argument('--niter', type=int, default=100, help=
            'number of epochs to train for')
        self.parser.add_argument('--beta1', type=float, default=0.5, help=
            'betan1 for adam')
        self.parser.add_argument('--lr', type=float, default=0.001, help=
            'initial learning rate for adam')
        self.parser.add_argument('--decay', type=float, default=1, help=
            'decay factor')
        self.parser.add_argument('--valen', type=int, default=0, help=
            'valen index 0-4')
        self.parser.add_argument('--log', type=int, default=0, help=
            'log batch size')

        # Test
        self.parser.add_argument('--dataset', action='store_true', help=
            'load model on test mode')
        self.parser.add_argument('--threshold', type=float, default=0.05,
            help='threshold score for anomaly')

        self.doc = None

    def parse(self):
        # Parse arguments.
        opt, args = self.parser.parse_known_args()

        self.doc = self.parser.format_help()

        str_ids = self.opt.gpu_ids.split(',')
        self.opt.gpu_ids = []
        for str_id in str_ids:
            id = int(str_id)
            if id >= 0:
                self.opt.gpu_ids.append(id)

        # set gpu ids
        if len(self.opt.gpu_ids) > 0:
            torch.cuda.set_device(self.opt.gpu_ids[0])

        args = vars(args)
        # print('===== Options =====')
        # for k, v in vars(opt).items():
        #     # print('opt.%s (%s): %s' % (k, type(v), v))
        #     # print('===== End =====')

        # save to the disk
        self.opt.name = '%s/%s' % (self.opt.model, self.opt.dataset)
        exp_dir = os.path.join(self.opt.outf, self.opt.name, 'train')
        test_dir = os.path.join(self.opt.outf, self.opt.name, 'test')

        if not os.path.isdir(exp_dir):
            os.makedirs(exp_dir)
        if not os.path.isdir(test_dir):
            os.makedirs(test_dir)

        file_name = os.path.join(exp_dir, 'opt.txt')
        with open(file_name, 'w') as opt_file:
            opt_file.write('Options\n')
            for k, v in sorted(vars(opt).items()):
                opt_file.write(' %s: %s\n' % (str(k), str(v)))
            opt_file.write('\n')
        return self.opt
```

```
import argparse
```

```
import os
```

```
import torch
```

```
class Options():
```

```
    """Options class
```

```
    Returns:
```

```
    [argparse]: argparse containing train and test options
```

```
    """
```

```
    def __init__(self):
```

```
        ##
```

```
        #
```

```
        self.parser = argparse.ArgumentParser(formatter_class=argparse.ArgumentDefaultsHelpFormatter)
```

```
        ##
```

```
        # Base
```

```
        self.parser.add_argument('--dataset', default='ecg', help='ecg dataset')
```

```
        self.parser.add_argument('--dataroot', default='', help='path to dataset')
```

```
        self.parser.add_argument('--batchsize', type=int, default=64, help='input batch size')
```

```
        self.parser.add_argument('--workers', type=int, help='number of data loading workers', default=1)
```

```
        self.parser.add_argument('--isize', type=int, default=320, help='input sequence size.')
```

```
        self.parser.add_argument('--nc', type=int, default=1, help='input sequence channels')
```

```
        self.parser.add_argument('--nz', type=int, default=50, help='size of the latent z vector')
```

```
        self.parser.add_argument('--ngf', type=int, default=32)
```

```
        self.parser.add_argument('--ndf', type=int, default=32)
```

```
        self.parser.add_argument('--device', type=str, default='gpu', help='Device: gpu | cpu')
```

```
        self.parser.add_argument('--gpu_ids', type=str, default='0', help='gpu ids: e.g. 0 0,1,2, 0,2. use -1 for CPU')
```

```
        self.parser.add_argument('--ngpu', type=int, default=1, help='number of GPUs to use')
```

```
        self.parser.add_argument('--model', type=str, default='beatgan', help='choose model')
```

```
        self.parser.add_argument('--outf', default='./output', help='output folder')
```


BeatGAN – options.py

[illegible]

```
# Train
self.parser.add_argument('--print_freq', type=int, default=100, help='frequency of showing training results on console')
self.parser.add_argument('--niter', type=int, default=100, help='number of epochs to train for')
self.parser.add_argument('--beta1', type=float, default=0.5, help='momentum term of adam')
self.parser.add_argument('--lr', type=float, default=0.0001, help='initial learning rate for adam')
self.parser.add_argument('--w_adv', type=float, default=1, help='parameter')
self.parser.add_argument('--folder', type=int, default=0, help='folder index 0-4')
self.parser.add_argument('--n_aug', type=int, default=0, help='aug data times')
```

```
## Test
self.parser.add_argument('--istest', action='store_true', help='train model or test model')
self.parser.add_argument('--threshold', type=float, default=0.05, help='threshold score for anomaly')

self.opt = None
```

sh파일 실행할때 출력되는 옵션들

```
Namespace(batchsize=64, beta1=0.5,  
dataroot='./dataset/preprocessed/ano0/', dataset='ecg', device='gpu',  
folder=0, gpu_ids=[0], isize=320, istest=True, lr=0.0001,  
model='beatgan', n_aug=0, name='beatgan/ecg', nc=1, ndf=32,  
ngf=32, ngpu=1, niter=100, nz=50, outf='./output', print_freq=100,  
threshold=0.02, w_adv=1, workers=1)
```

sh파일 실행할때 출력되는 옵션들

#Base #Train #Test

```
batchsize=64
beta1=0.5
dataroot='./dataset/preprocessed/ano0/'
dataset='ecg'
device='gpu'
folder=0
gpu_ids=[0]
isize=320
istest=True
lr=0.0001
model='beatgan'
n_aug=0
```

```
name='beatgan/ecg'
nc=1
ndf=32
ngf=32
ngpu=1
niter=100
nz=50
outf='./output'
print_freq=100
threshold=0.02
w_adv=1
workers=1)
```


Options

```
batchsize=64
dataroot='./dataset/preprocessed/ano0/'
dataset='ecg'
device='gpu'
gpu_ids=[0]
isize=320 → input sequence size
istest=True → train model or test model
model='beatgan'
name='beatgan/ecg'
nc=1 → input sequence channels
ndf=32 → number of filters in the discriminator
ngf=32 → number of filters in the generator
ngpu=1 → number of GPUs to use
nz=50 → size of the latent z vector
outf='./output'
threshold=0.02 → threshold score for anomaly
workers=1 → number of data loading workers (에러해결을 위해 다른 파일에서 0으로 수정함)
```

data.py 구성

```
1
2 import os
3 import numpy as np
4 import torch
5 from torch.utils.data import DataLoader, TensorDataset
6 from sklearn.model_selection import train_test_split
7
8 np.random.seed(42)
9
10 > def normalize(seq): ...
17
18 > def load_data(opt): ...
167
168
169 > def getFloderK(data, folder, label): ...
186
187 > def getPercent(data_x, data_y, percent, seed): ...
190
191 > def get_full_data(dataloader): ...
211
212
213 > def data_aug(train_x, train_y, times=2): ...
231
232 > def aug_ts(x): ...
```

← np.random.seed(42)

난수를 생성할 때,
일종의 기준인 **seed**가 존재
(기본 **seed** : 현재 날짜나 시간 등)

seed가 같을 경우 다음 생성될 난수가 예측됨
따라서 임의로 설정하면 난수 생성에 편리

data.py

```
def normalize(seq):  
    '''  
    normalize to [-1,1]  
    :param seq:  
    :return:  
    '''  
    return 2*(seq-np.min(seq))/(np.max(seq)-np.min(seq))-1
```

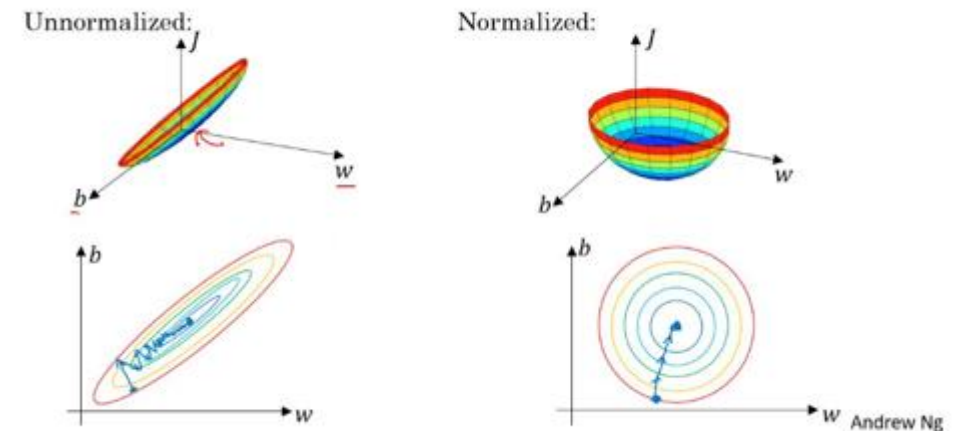
```
# normalize all  
for i in range(N_samples.shape[0]):  
    for j in range(opt.nc):  
        N_samples[i][j]=normalize(N_samples[i][j][:])  
N_samples=N_samples[:, :opt.nc, :]  
  
for i in range(S_samples.shape[0]):  
    for j in range(opt.nc):  
        S_samples[i][j] = normalize(S_samples[i][j][:])  
S_samples = S_samples[:, :opt.nc, :]  
  
for i in range(V_samples.shape[0]):  
    for j in range(opt.nc):  
        V_samples[i][j] = normalize(V_samples[i][j][:])  
V_samples = V_samples[:, :opt.nc, :]  
  
for i in range(F_samples.shape[0]):  
    for j in range(opt.nc):  
        F_samples[i][j] = normalize(F_samples[i][j][:])  
F_samples = F_samples[:, :opt.nc, :]  
  
for i in range(Q_samples.shape[0]):  
    for j in range(opt.nc):  
        Q_samples[i][j] = normalize(Q_samples[i][j][:])  
Q_samples = Q_samples[:, :opt.nc, :]
```

To normalize in $[-1, 1]$ you can use:

$$x'' = 2 \frac{x - \min x}{\max x - \min x} - 1$$

입력 데이터에 대해 정규화 :
모든 데이터 포인트가 동일한 정도의 스케일로
반영되도록 해주는 게 목표.

normalize가 unnormalized보다
경사하강법에서 수렴속도가 더 빠르다.



data.py

```
10 > def normalize(seq): ...
17
18 > def load_data(opt): ...
167
168
169 > def getFolderK(data,folder,label): ...
186
187 > def getPercent(data_x,data_y,percent,seed): ...
190
191 > def get_full_data(dataloader): ...
211
212
213 > def data_aug(train_x,train_y,times=2): ...
231
232 > def aug_ts(x): ...
```

Dataloader은 batch기반의 딥러닝모델 학습을 위해 mini batch를 만들어주는 역할을 한다.

서버에서 돌릴 때는 num_worker를 조절해서 load속도를 올릴 수 있지만, PC에서는 default로 설정해야 오류가 나지 않는다.

****num_worker :**

학습 도중 CPU의 작업을 몇 개의 코어로 사용해서 진행할지 설정

```
dataloader = {"train": DataLoader(
    dataset=train_dataset, # torch TensorDataset format
    batch_size=opt.batchsize, # mini batch size
    shuffle=True,
    #num_workers=int(opt.workers),
    num_workers=0,
    drop_last=True),
    "val": DataLoader(
    dataset=val_dataset, # torch TensorDataset format
    batch_size=opt.batchsize, # mini batch size
    shuffle=True,
    #num_workers=int(opt.workers),
    num_workers=0,
    drop_last=False),
    "test_N": DataLoader(
    dataset=test_N_dataset, # torch TensorDataset format
    batch_size=opt.batchsize, # mini batch size
    shuffle=True,
    #num_workers=int(opt.workers),
    num_workers=0,
    drop_last=False),
    "test_S": DataLoader(
    dataset=test_S_dataset, # torch TensorDataset format
    batch_size=opt.batchsize, # mini batch size
    shuffle=True,
    #num_workers=int(opt.workers),
    num_workers=0,
    drop_last=False),
```

```
    "test_V": DataLoader(
    dataset=test_V_dataset, # torch TensorDataset format
    batch_size=opt.batchsize, # mini batch size
    shuffle=True,
    #num_workers=int(opt.workers),
    num_workers=0,
    drop_last=False),
    "test_F": DataLoader(
    dataset=test_F_dataset, # torch TensorDataset format
    batch_size=opt.batchsize, # mini batch size
    shuffle=True,
    #num_workers=int(opt.workers),
    num_workers=0,
    drop_last=False),
    "test_Q": DataLoader(
    dataset=test_Q_dataset, # torch TensorDataset format
    batch_size=opt.batchsize, # mini batch size
    shuffle=True,
    #num_workers=int(opt.workers),
    num_workers=0,
    drop_last=False),
    }
return dataloader
```

model.py

```
1  import time,os,sys
2
3  import numpy as np
4  import torch
5  import torch.nn as nn
6  import torch.optim as optim
7
8  from network import Encoder,Decoder,AD_MODEL,weights_init,print_network
9
10 dirname=os.path.dirname
11 sys.path.insert(0,dirname(dirname(os.path.abspath(__file__))))
12
13 from metric import evaluate
14
15
16 ##
17 > class Discriminator(nn.Module): ...
35
36 ##
37 > class Generator(nn.Module): ...
48
49
50 > class BeatGAN(AD_MODEL): ...
444
```

torch.nn.module은

모든 뉴럴 네트워크 모듈의 기본 클래스

일반적인 모델들은

이 클래스를 상속받아야 하고,

일반적으로 두가지 메소드는 반드시 **override**(재정의)해야한다.

- **__init__(self)** : 모델에서 사용될 **module**, **activation function** 등 정의
- **forward(self,x)** : 모델에서 실행되어야 하는 계산 정의.

model.py

```
1  import time,os,sys
2
3  import numpy as np
4  import torch
5  import torch.nn as nn
6  import torch.optim as optim
7
8  from network import Encoder,Decoder,AD_MODEL,weights_init,print_network
9
10 dirname=os.path.dirname
11 sys.path.insert(0,dirname(dirname(os.path.abspath(__file__))))
12
13 from metric import evaluate
14
15
16 ##
17 > class Discriminator(nn.Module): ...
35
36 ##
37 > class Generator(nn.Module): ...
48
49
50 > class BeatGAN(AD_MODEL): ...
444
```

PyTorch로 신경망 설계할 때
두가지 방법이 있다.

- (1) 사용자 정의 nn모듈
- (2) nn.Module을 상속한 클래스 이용

nn.Module은 신경망을 만드는 데
필요한 많은 일을 대신 해준다!

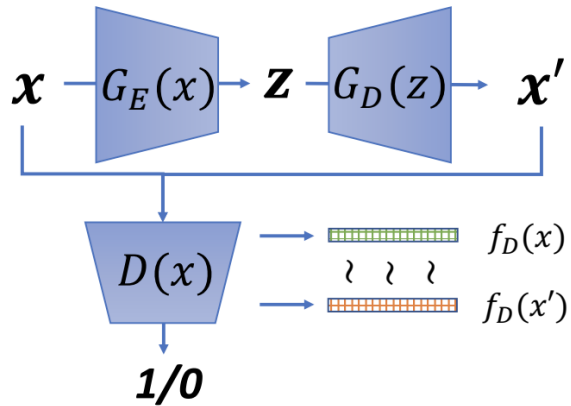
torch.nn.module은
모든 뉴럴 네트워크 모듈의 기본 클래스

일반적인 모델들은
이 클래스를 상속받아야 하고,
일반적으로 두가지 메소드는 반드시 **override**(재정의)해야한다.

- **__init__(self)** : 모델에서 사용될 **module**, **activation function** 등 정의
- **forward(self,x)** : 모델에서 실행되어야 하는 계산 정의.

model.py (class Discriminator)

```
17 > class Discriminator(nn.Module): ...
35
36 ##
37 > class Generator(nn.Module): ...
48
49
50 > class BeatGAN(AD_MODEL): ...
444
```



```
class Discriminator(nn.Module):

    def __init__(self, opt):
        super(Discriminator, self).__init__()
        model = Encoder(opt.ngpu, opt, 1)
        layers = list(model.main.children())

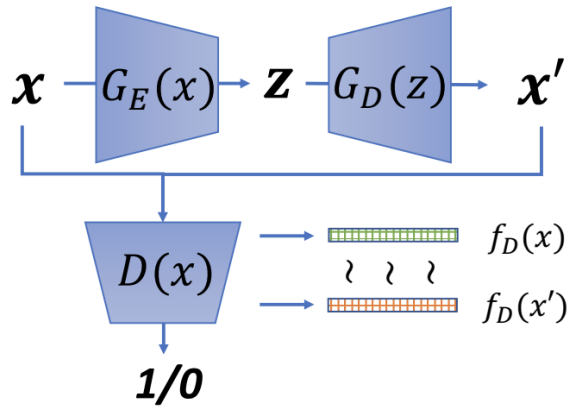
        self.features = nn.Sequential(*layers[:-1])
        self.classifier = nn.Sequential(layers[-1])
        self.classifier.add_module('Sigmoid', nn.Sigmoid())

    def forward(self, x):
        features = self.features(x)
        features = features
        classifier = self.classifier(features)
        classifier = classifier.view(-1, 1).squeeze(1)

        return classifier, features
```

model.py (class Generator)

```
17 > class Discriminator(nn.Module): ...  
35  
36 ##  
37 > class Generator(nn.Module): ...  
48  
49  
50 > class BeatGAN(AD_MODEL): ...  
444
```



```
class Generator(nn.Module):  
  
    def __init__(self, opt):  
        super(Generator, self).__init__()  
        self.encoder1 = Encoder(opt.ngpu, opt, opt.nz)  
        self.decoder = Decoder(opt.ngpu, opt)  
  
    def forward(self, x):  
        latent_i = self.encoder1(x)  
        gen_x = self.decoder(latent_i)  
        return gen_x, latent_i
```


network.py – class Encoder

BeatGAN 논문에서,
Evaluation on ECG Dataset의 Experimental setup

We use 5 1D transposed convolutional layers followed by batch-norm and leaky ReLU activation, with slope of the leak set to 0.2. The transposed convolutional kernel's size and number of each layer are 512(10/1)-216(4/2)-128(4/2)-64(4/2)-32(4/2): e.g. 512(10/1) means that the number of filters is 512, the size of filter is 10 and the stride is 1.

```
30 class Encoder(nn.Module):
31     def __init__(self, ngpu, opt, out_z):
32         super(Encoder, self).__init__()
33         self.ngpu = ngpu
34         self.main = nn.Sequential(
35             # input is (nc) x 320
36             nn.Conv1d(opt.nc, opt.ndf, 4, 2, 1, bias=False),
37             nn.LeakyReLU(0.2, inplace=True),
38             # state size. (ndf) x 160
39             nn.Conv1d(opt.ndf, opt.ndf * 2, 4, 2, 1, bias=False),
40             nn.BatchNorm1d(opt.ndf * 2),
41             nn.LeakyReLU(0.2, inplace=True),
42             # state size. (ndf*2) x 80
43             nn.Conv1d(opt.ndf * 2, opt.ndf * 4, 4, 2, 1, bias=False),
44             nn.BatchNorm1d(opt.ndf * 4),
45             nn.LeakyReLU(0.2, inplace=True),
46             # state size. (ndf*4) x 40
47             nn.Conv1d(opt.ndf * 4, opt.ndf * 8, 4, 2, 1, bias=False),
48             nn.BatchNorm1d(opt.ndf * 8),
49             nn.LeakyReLU(0.2, inplace=True),
50             # state size. (ndf*8) x 20
51             nn.Conv1d(opt.ndf * 8, opt.ndf * 16, 4, 2, 1, bias=False),
52             nn.BatchNorm1d(opt.ndf * 16),
53             nn.LeakyReLU(0.2, inplace=True),
54             # state size. (ndf*16) x 10
55
56             nn.Conv1d(opt.ndf * 16, out_z, 10, 1, 0, bias=False),
57             # state size. (nz) x 1
58         )
59
60     def forward(self, input):
61         if input.is_cuda and self.ngpu > 1:
62             output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
63         else:
64             output = self.main(input)
65
66         return output
```

Transposed convolution

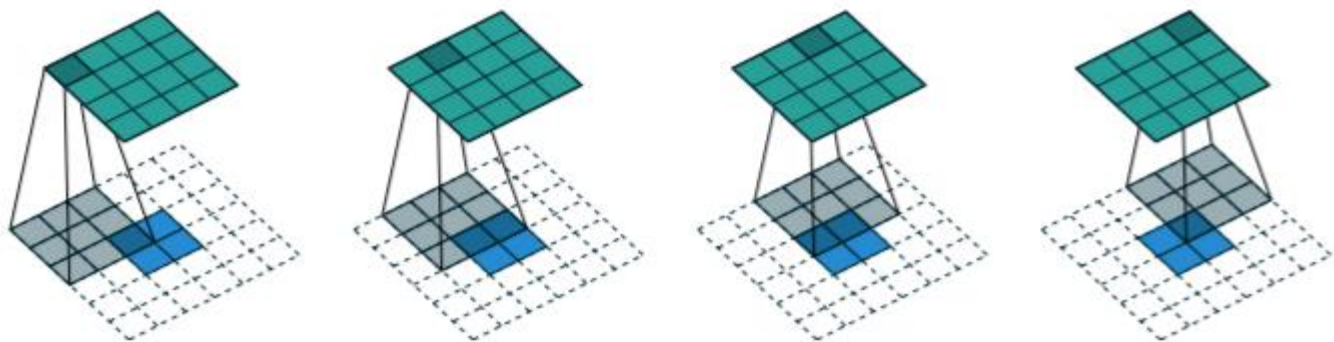


Figure 4.1: The transpose of convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$). It is equivalent to convolving a 3×3 kernel over a 2×2 input padded with a 2×2 border of zeros using unit strides (i.e., $i' = 2$, $k' = k$, $s' = 1$ and $p' = 2$).

```
30 class Encoder(nn.Module):
31     def __init__(self, ngpu, opt, out_z):
32         super(Encoder, self).__init__()
33         self.ngpu = ngpu
34         self.main = nn.Sequential(
35             # input is (nc) x 320
36             nn.Conv1d(opt.nc, opt.ndf, 4, 2, 1, bias=False),
37             nn.LeakyReLU(0.2, inplace=True),
38             # state size. (ndf) x 160
39             nn.Conv1d(opt.ndf, opt.ndf * 2, 4, 2, 1, bias=False),
40             nn.BatchNorm1d(opt.ndf * 2),
41             nn.LeakyReLU(0.2, inplace=True),
42             # state size. (ndf*2) x 80
43             nn.Conv1d(opt.ndf * 2, opt.ndf * 4, 4, 2, 1, bias=False),
44             nn.BatchNorm1d(opt.ndf * 4),
45             nn.LeakyReLU(0.2, inplace=True),
46             # state size. (ndf*4) x 40
47             nn.Conv1d(opt.ndf * 4, opt.ndf * 8, 4, 2, 1, bias=False),
48             nn.BatchNorm1d(opt.ndf * 8),
49             nn.LeakyReLU(0.2, inplace=True),
50             # state size. (ndf*8) x 20
51             nn.Conv1d(opt.ndf * 8, opt.ndf * 16, 4, 2, 1, bias=False),
52             nn.BatchNorm1d(opt.ndf * 16),
53             nn.LeakyReLU(0.2, inplace=True),
54             # state size. (ndf*16) x 10
55
56             nn.Conv1d(opt.ndf * 16, out_z, 10, 1, 0, bias=False),
57             # state size. (nz) x 1
58         )
59
60     def forward(self, input):
61         if input.is_cuda and self.ngpu > 1:
62             output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
63         else:
64             output = self.main(input)
65
66         return output
```

파라미터의 조정으로 원하는 크기의 이미지를 만들 수 있다!
위의 사진은 2x2 이미지에 Transposed Conv를 적용시켜서 4x4 이미지를 만들
3x3 filter, 그리고 padding을 이용해서 Upsampling을 수행하는 Layer을 만든 것

예시) CNN을 사용한 Encoder-Decoder 구조의 Autoencoder
Encoder에서 Pooling 등을 통해 이미지를 축소시키면서 데이터를 압축했다면,
그 데이터를 다시 원래의 이미지로 복원하기 위해 이미지를 크게 만들어야 할 때 사용할 수 있다.

1D Convolution

CNN은 일반적으로 이미지에서 계층적 특징 추출을 위해 사용된다.
이러한 장점을 활용하여 2차원 이미지가 아닌 **1차원의 sequential 데이터**에도 CNN이 사용된다. 주어진 sequence data에서 중요한 정보를 추출해낼 수 있다.

여|스|)

```
# input is (nc) x 320
nn.Conv1d(opt.nc, opt.ndf, 4, 2, 1, bias=False),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf) x 160
nn.Conv1d(opt.ndf, opt.ndf * 2, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 2),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*2) x 80
nn.Conv1d(opt.ndf * 2, opt.ndf * 4, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 4),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*4) x 40
nn.Conv1d(opt.ndf * 4, opt.ndf * 8, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 8),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*8) x 20
nn.Conv1d(opt.ndf * 8, opt.ndf * 16, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 16),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*16) x 10

nn.Conv1d(opt.ndf * 16, out_z, 10, 1, 0, bias=False),
# state size. (nz) x 1
```

[illegible]

[PyTorch에서 Conv1d의 파라미터]

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the convolving kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1
- **padding** (*int, tuple or str, optional*) – Padding added to both sides of the input. Default: 0
- **padding_mode** (*string, optional*) – 'zeros', 'reflect', 'replicate' or 'circular'. Default: 'zeros'
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1
- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool, optional*) – If `True`, adds a learnable bias to the output. Default: `True`

1D Convolution

```
# input is (nc) x 320
nn.Conv1d(opt.nc, opt.ndf, 4, 2, 1, bias=False),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf) x 160
nn.Conv1d(opt.ndf, opt.ndf * 2, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 2),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*2) x 80
nn.Conv1d(opt.ndf * 2, opt.ndf * 4, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 4),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*4) x 40
nn.Conv1d(opt.ndf * 4, opt.ndf * 8, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 8),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*8) x 20
nn.Conv1d(opt.ndf * 8, opt.ndf * 16, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 16),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*16) x 10

nn.Conv1d(opt.ndf * 16, out_z, 10, 1, 0, bias=False),
# state size. (nz) x 1
```

option에 따르면

nc=1 (채널 수)

ndf=32 (generator에 있는 필터의 수)

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the convolving kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1
- **padding** (*int, tuple or str, optional*) – Padding added to both sides of the input. Default: 0

Bach Normalization

```
# input is (nc) x 320
nn.Conv1d(opt.nc,opt.ndf,4,2,1,bias=False),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf) x 160
nn.Conv1d(opt.ndf, opt.ndf * 2, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 2),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*2) x 80
nn.Conv1d(opt.ndf * 2, opt.ndf * 4, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 4),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*4) x 40
nn.Conv1d(opt.ndf * 4, opt.ndf * 8, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 8),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*8) x 20
nn.Conv1d(opt.ndf * 8, opt.ndf * 16, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 16),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*16) x 10

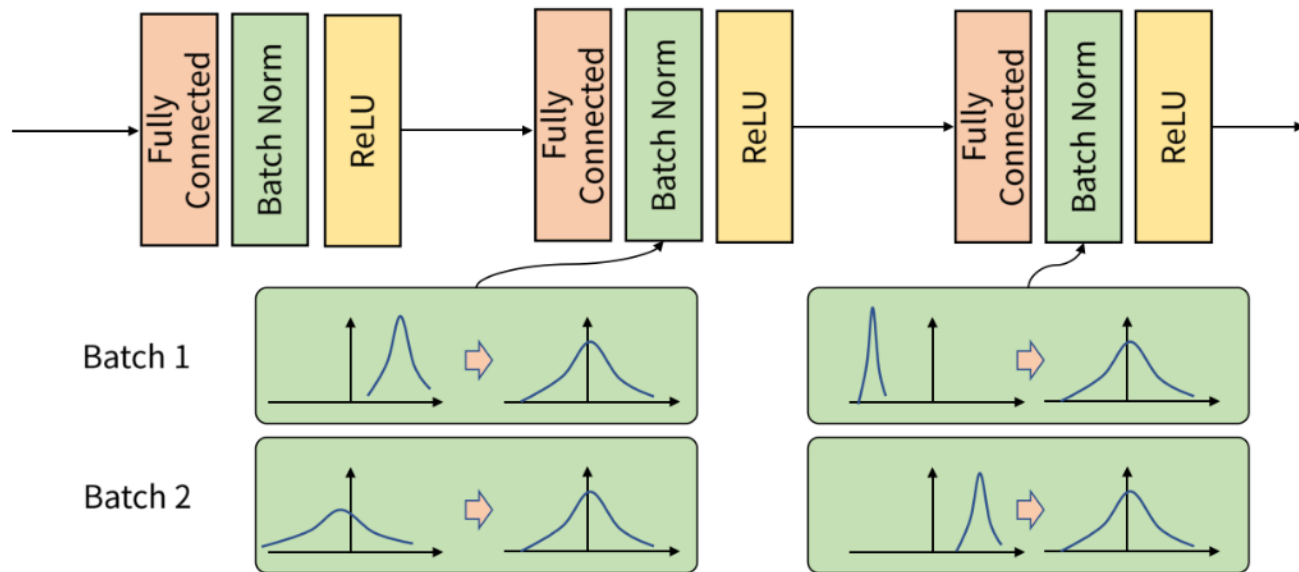
nn.Conv1d(opt.ndf * 16, out_z, 10, 1, 0, bias=False),
# state size. (nz) x 1
```

Batch단위로 학습을 하게 되면 계층 별로 입력의 데이터 분포가 달라지는 현상이 있다. (Internal Covariant Shift)

이 문제를 개선하기 위한 개념이 Batch Normalization이다.

각 배치별로 평균과 분산을 이용해 정규화하는 것

이미지 예시)



-활성화 함수-

ReLU (rectified linear unit)


```
# input is (nc) x 320
nn.Conv1d(opt.nc,opt.ndf,4,2,1,bias=False),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf) x 160
nn.Conv1d(opt.ndf, opt.ndf * 2, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 2),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*2) x 80
nn.Conv1d(opt.ndf * 2, opt.ndf * 4, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 4),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*4) x 40
nn.Conv1d(opt.ndf * 4, opt.ndf * 8, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 8),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*8) x 20
nn.Conv1d(opt.ndf * 8, opt.ndf * 16, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 16),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*16) x 10

nn.Conv1d(opt.ndf * 16, out_z, 10, 1, 0, bias=False),
# state size. (nz) x 1
```

딥러닝 네트워크에서 노드에 입력된 값들을 **비선형 함수**에 통과시킨 후 다음 레이어로 전달하는데, 이 때 사용하는 함수를 활성화 함수 (Activation Function)라고 한다.

선형 함수가 아니라 비선형 함수를 사용하는 이유는 딥러닝 모델의 레이어 층을 깊게 가져갈 수 있기 때문이다.

Sigmoid


$$y = \frac{1}{1 + e^{-x}}$$

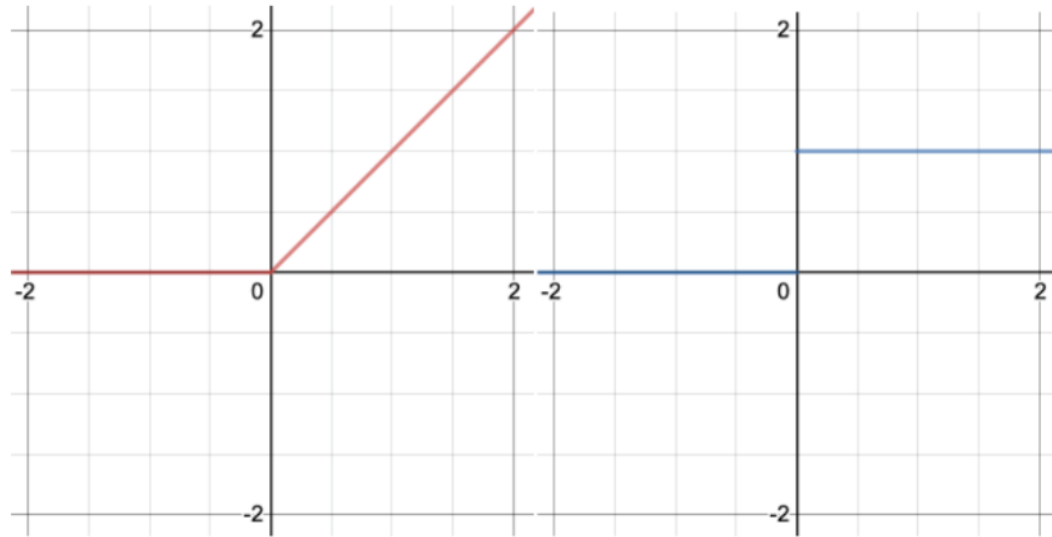
-활성화 함수-

ReLU (rectified linear unit)

```
# input is (nc) x 320
nn.Conv1d(opt.nc,opt.ndf,4,2,1,bias=False),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf) x 160
nn.Conv1d(opt.ndf, opt.ndf * 2, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 2),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*2) x 80
nn.Conv1d(opt.ndf * 2, opt.ndf * 4, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 4),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*4) x 40
nn.Conv1d(opt.ndf * 4, opt.ndf * 8, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 8),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*8) x 20
nn.Conv1d(opt.ndf * 8, opt.ndf * 16, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 16),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*16) x 10

nn.Conv1d(opt.ndf * 16, out_z, 10, 1, 0, bias=False),
# state size. (nz) x 1
```

$$f(x) = \begin{cases} x & (x \geq 0) \\ 0 & (x < 0) \end{cases} \quad \tilde{f}(x) = \begin{cases} 1 & (x \geq 0) \\ 0 & (x < 0) \end{cases}$$



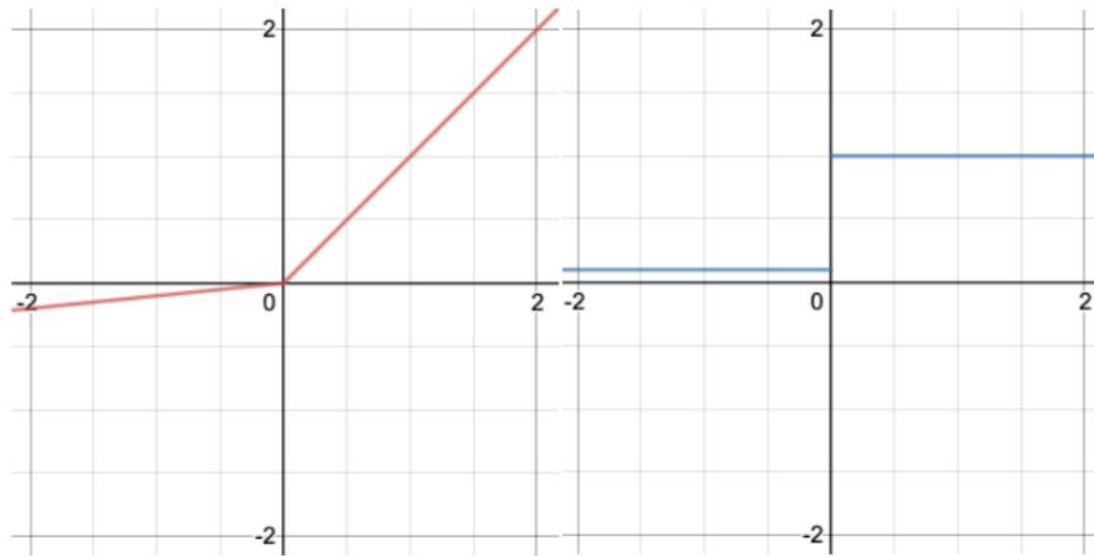
- 특징: 0 이하의 값은 다음 레이어에 전달하지 않습니다. 0이상의 값은 그대로 출력합니다.
- 사용처: CNN을 학습시킬 때 많이 사용됩니다.
- 한계점: 한번 0 활성화 값을 다음 레이어에 전달하면 이후의 뉴런들의 출력값이 모두 0이 되는 현상이 발생합니다. 이를 **dying ReLU**라 부릅니다. 이러한 한계점을 개선하기 위해 음수 출력 값을 소량이나마 다음 레이어에 전달하는 방식으로 개선한 활성화 함수들이 등장합니다.

-활성화 함수- LeakyReLU

```
# input is (nc) x 320
nn.Conv1d(opt.nc,opt.ndf,4,2,1,bias=False),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf) x 160
nn.Conv1d(opt.ndf, opt.ndf * 2, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 2),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*2) x 80
nn.Conv1d(opt.ndf * 2, opt.ndf * 4, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 4),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*4) x 40
nn.Conv1d(opt.ndf * 4, opt.ndf * 8, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 8),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*8) x 20
nn.Conv1d(opt.ndf * 8, opt.ndf * 16, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 16),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*16) x 10

nn.Conv1d(opt.ndf * 16, out_z, 10, 1, 0, bias=False),
# state size. (nz) x 1
```

$$f(x) = \begin{cases} x & (x > 0) \\ \alpha x & (x \leq 0) \end{cases} \quad f^*(\alpha, x) = \begin{cases} 1 & (x > 0) \\ \alpha & (x \leq 0) \end{cases}$$



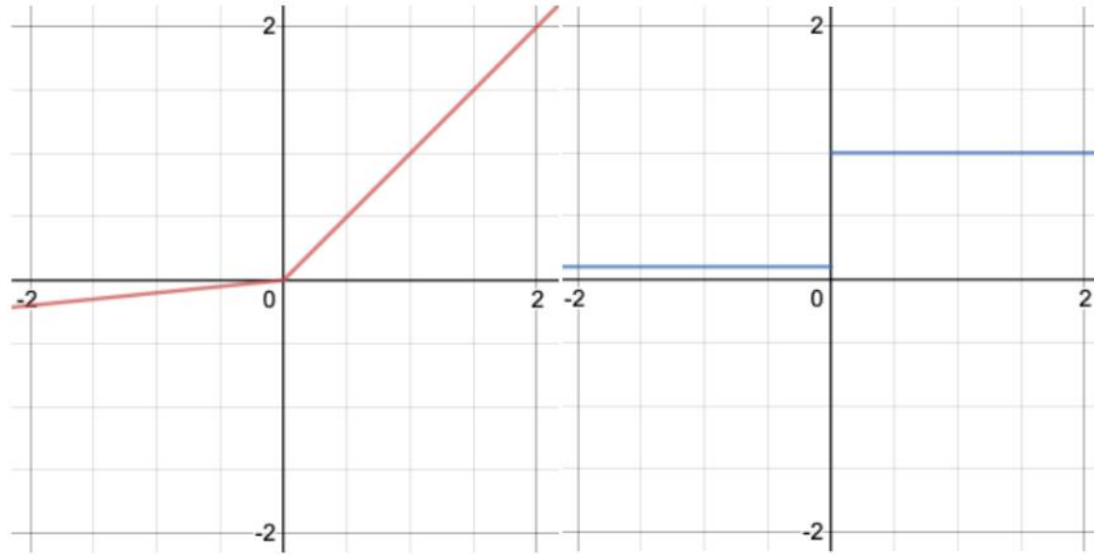
특징: ReLU와 거의 비슷한 형태를 갖습니다.
입력 값이 음수일 때 완만한 선형 함수를 그려줍니다.
일반적으로 알파를 **0.01**로 설정합니다.
(위 그래프에서는 시각화 편의상 알파를 **0.1**로 설정)

-활성화 함수- LeakyReLU

```
# input is (nc) x 320
nn.Conv1d(opt.nc,opt.ndf,4,2,1,bias=False),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf) x 160
nn.Conv1d(opt.ndf, opt.ndf * 2, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 2),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*2) x 80
nn.Conv1d(opt.ndf * 2, opt.ndf * 4, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 4),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*4) x 40
nn.Conv1d(opt.ndf * 4, opt.ndf * 8, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 8),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*8) x 20
nn.Conv1d(opt.ndf * 8, opt.ndf * 16, 4, 2, 1, bias=False),
nn.BatchNorm1d(opt.ndf * 16),
nn.LeakyReLU(0.2, inplace=True),
# state size. (ndf*16) x 10

nn.Conv1d(opt.ndf * 16, out_z, 10, 1, 0, bias=False),
# state size. (nz) x 1
```

$$f(x) = \begin{cases} x & (x > 0) \\ \alpha x & (x \leq 0) \end{cases} \quad f^*(\alpha, x) = \begin{cases} 1 & (x > 0) \\ \alpha & (x \leq 0) \end{cases}$$



특징: ReLU와 거의 비슷한 형태를 갖습니다.
입력 값이 음수일 때 완만한 선형 함수를 그려줍니다.
일반적으로 알파를 **0.01**로 설정합니다.
(위 그래프에서는 시각화 편의상 알파를 **0.1**로 설정)

Encoder, Decoder

```
class Encoder(nn.Module):
    def __init__(self, ngpu,opt,out_z):
        super(Encoder, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is (nc) x 320
            nn.Conv1d(opt.nc,opt.ndf,4,2,1,bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf) x 160
            nn.Conv1d(opt.ndf, opt.ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*2) x 80
            nn.Conv1d(opt.ndf * 2, opt.ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*4) x 40
            nn.Conv1d(opt.ndf * 4, opt.ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*8) x 20
            nn.Conv1d(opt.ndf * 8, opt.ndf * 16, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ndf * 16),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*16) x 10

            nn.Conv1d(opt.ndf * 16, out_z, 10, 1, 0, bias=False),
            # state size. (nz) x 1
        )

    def forward(self, input):
        if input.is_cuda and self.ngpu > 1:
            output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
        else:
            output = self.main(input)

        return output
```

```
class Decoder(nn.Module):
    def __init__(self, ngpu,opt):
        super(Decoder, self).__init__()
        self.ngpu = ngpu
        self.main=nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose1d(opt.nz,opt.ngf*16,10,1,0,bias=False),
            nn.BatchNorm1d(opt.ngf*16),
            nn.ReLU(True),
            # state size. (ngf*16) x10
            nn.ConvTranspose1d(opt.ngf * 16, opt.ngf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf * 8),
            nn.ReLU(True),
            # state size. (ngf*8) x 20
            nn.ConvTranspose1d(opt.ngf * 8, opt.ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf * 4),
            nn.ReLU(True),
            # state size. (ngf*2) x 40
            nn.ConvTranspose1d(opt.ngf * 4, opt.ngf*2, 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf*2),
            nn.ReLU(True),
            # state size. (ngf) x 80
            nn.ConvTranspose1d(opt.ngf * 2, opt.ngf , 4, 2, 1, bias=False),
            nn.BatchNorm1d(opt.ngf ),
            nn.ReLU(True),
            # state size. (ngf) x 160
            nn.ConvTranspose1d(opt.ngf , opt.nc, 4, 2, 1, bias=False),
            nn.Tanh()
            # state size. (nc) x 320
        )

    def forward(self, input):
        if input.is_cuda and self.ngpu > 1:
            output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
        else:
            output = self.main(input)

        return output
```

model.py (class Generator)

```
17 > class Discriminator(nn.Module): ...
35
36 ##
37 > class Generator(nn.Module): ...
48
49
50 > class BeatGAN(AD_MODEL): ...
444
```

class BeatGAN의 메소드

- __init__(self,opt,dataloader,device):
- train
- train_epoch
- set_input
- optimize
- update_netd
- reinitialize_netd
- update_netg
- get_errors
- get_generated_x
- validate
- predict
- predict_for_right
- test_type
- test_time