

# Dacon 카메라 이미지 품질 향상 AI 경진대회

---

팀원 : A – CV 팀 김성민, 손상원, 박세정, 안지송, 장수명

---

A – CV Team  
Project

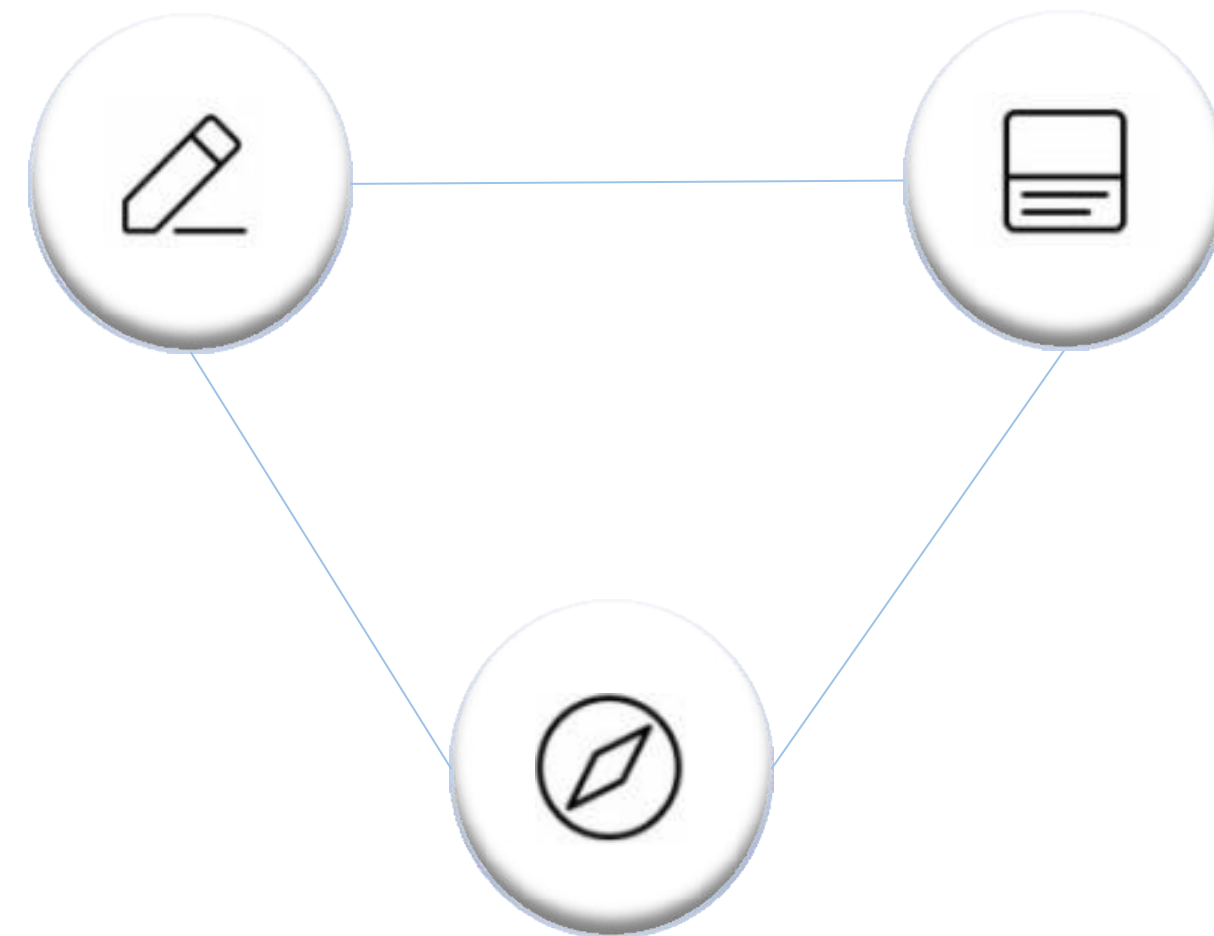
# CONTENTS

1. 빛번짐을 줄이자!

2. 데이터 전처리

3. Pix2Pix

4. Nfnet



CONTENTS

# 1. 빛 번짐을 줄이자!



빛 번짐으로 인한 이미지 품질 저하를  
해결하는 것이 목표

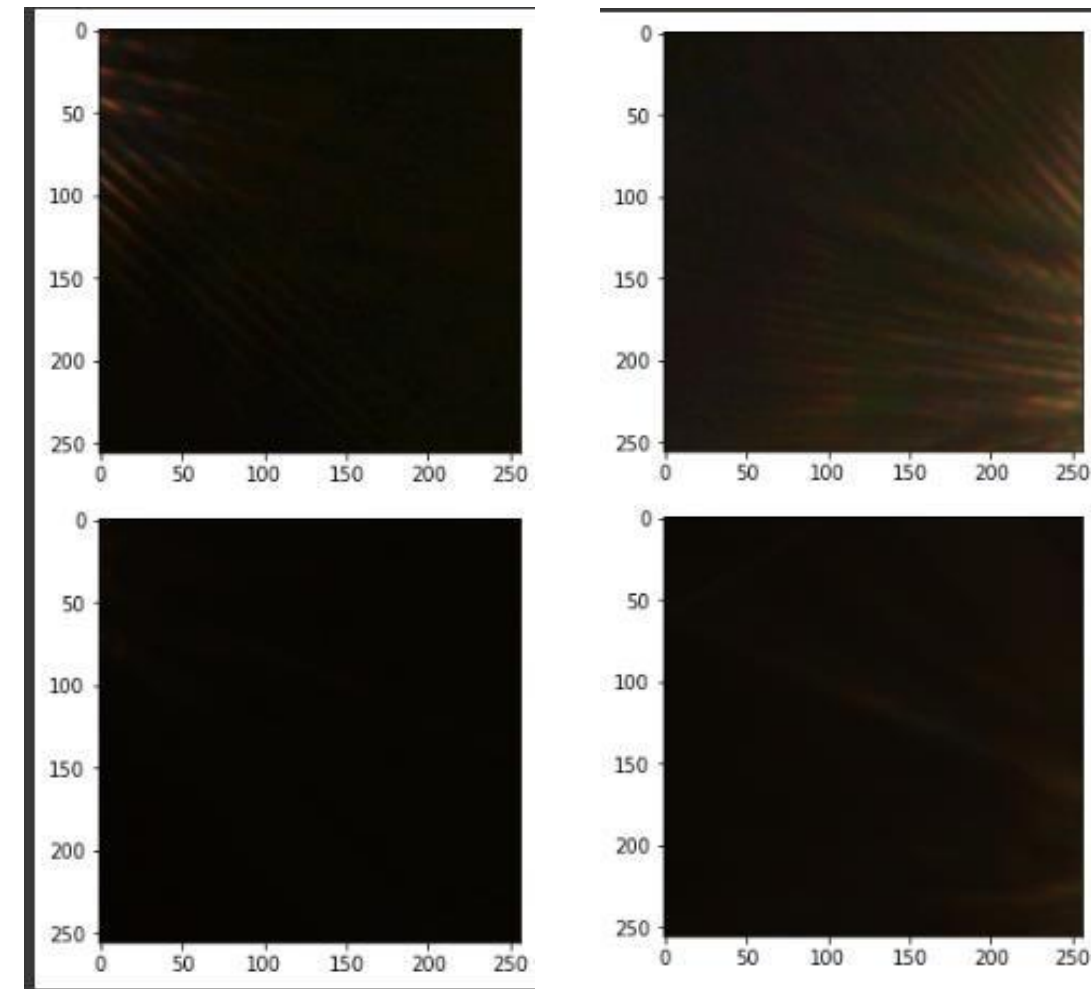
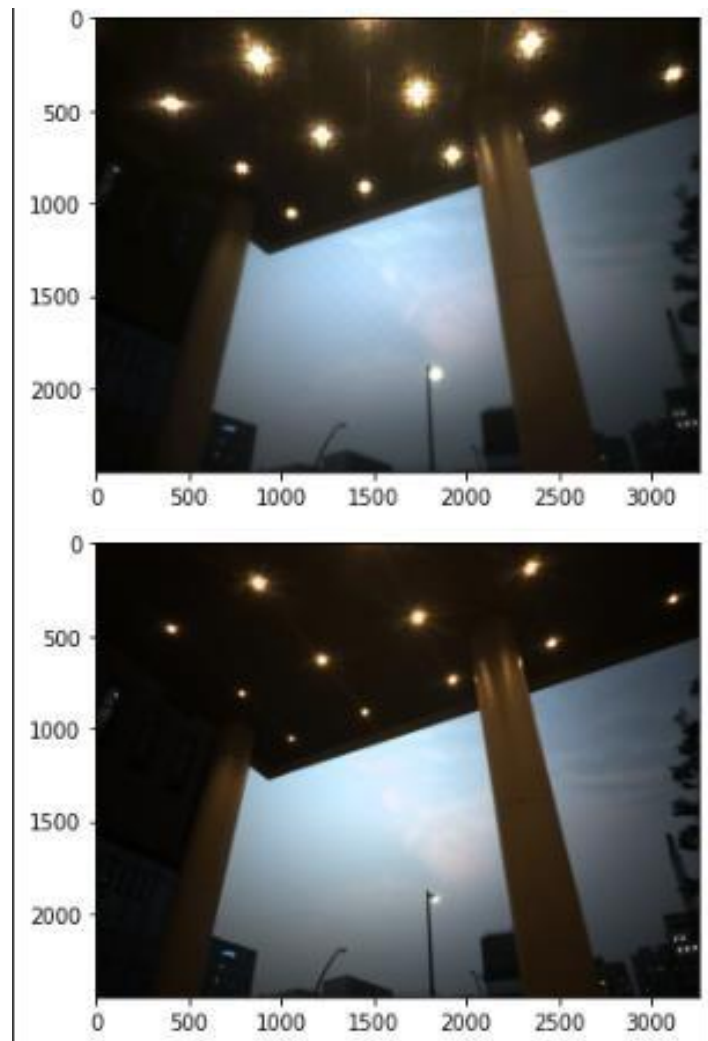
# 1. 빛 번짐을 줄이자!

1. 베이스라인 코드 이해
2. Pix2Pix와 Nfnnet을 기반으로 모델링

## 2. 데이터 전처리

기존의 이미지 파일을 조각내어(crop) 학습시키는 작업을 진행

왼쪽 이미지를 오른쪽 이미지로 쪼개어 진행. 추가적으로 flip 과 rotation을 추가





### 3. pix2pix

## Pix2Pix

Input 데이터와 label 데이터 모두 이미지로 주어져 학습하는 모델로 GAN의 일종

지도 학습이지만, Input 데이터는 보통 구하기 쉬운 반면,  
label 데이터는 구하기 어려운 경우 유용함

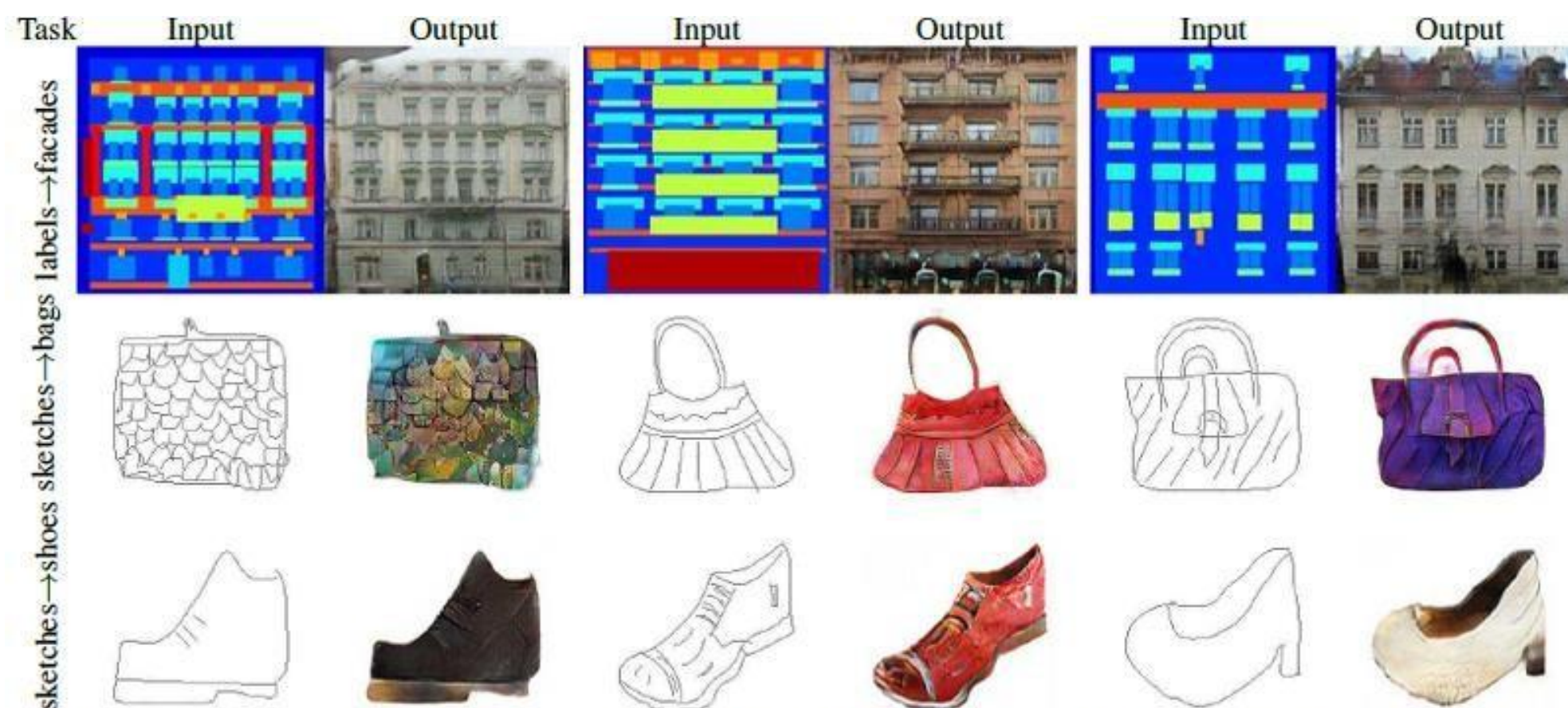


Figure 9: Results of our method on several tasks (data from [42] and [17]). Note that the sketch→photo results are generated by a model trained on automatic edge detections and tested on human-drawn sketches. Please see online materials for additional examples.

## 4. NFnet

---

### NFnet

2021년 2월 발표된 최신 논문이며 SOTA를 보이고 있다.

대부분 사용하는 배치정규화의 단점, 계산 복잡도 증가, 배치 사이즈가 작을 경우 성능 저하

등등의 문제로 배치정규화를 사용하지 않는다는 것에서 출발

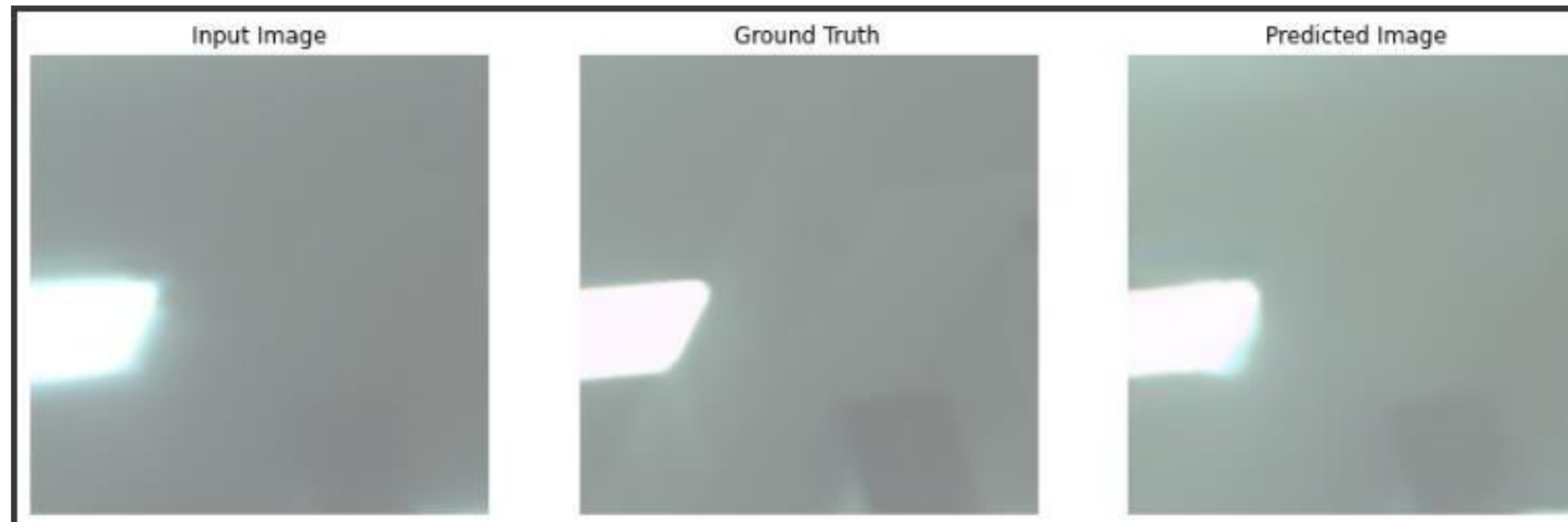


Adaptive Gradient Clipping 도입!

Gradient Clipping : 기울기가 특정 임계값을 초과하지 않도록 하여 모델 학습을 안정화 시키는 방법

## 5. 결과

### Pix2Pix



무언가 나오긴 했지만 아쉬운 결과...

예상 원인

1. 적은 Epoch (일반적 Pix2Pix는 수백번의 Epoch이 기본으로 보이지만 Colab 런타임에러로 인해 30번만 학습시킴)
2. 충분하지 못했던 하이퍼 파라미터 조정 및 다양한 데이터 전처리 방법 시도



## 5. 결과

### NFnet


NFnet과 backbone으로 Baseline 코드에서 제공하는 ResUnet을

사용 하지만 반복된 학습에도 loss가 nan을 기록하며 학습에 실패...

```
[ ] 1 start = time.time()
    2
    3 history = model.fit(train_ds, validation_data=val_ds, callbacks=train_callbacks, epochs=10)

Epoch 1/10
39810/39810 [=====] - 3481s 86ms/step - loss: nan - accuracy: 0.3542 - val_loss: nan - val_accuracy: 0.4010
Epoch 2/10
39810/39810 [=====] - 3427s 86ms/step - loss: nan - accuracy: 0.3542 - val_loss: nan - val_accuracy: 0.4010
Epoch 3/10
27918/39810 [=====>.....] - ETA: 15:46 - loss: nan - accuracy: 0.3714ERROR:root:Internal Python error in the inspect module.
```

예상 원인으로는 적절하지 못한 하이퍼 파라미터 설정 등이 있지만, 찾아내는 데에는 실패



# Deep Learning을 활용한 Starcraft2 Cinematic Video Super Resolution

A – CV팀 손상원, A – ML 이지웅

Feat. Colab Pro의 위대함

# CONTENTS



*Super resolution*  
이란?



*RDN & RDB?*



프로젝트 진행  
일대기  
(Feat. 맨땅에 헤딩해서 CV 프로젝트 하기)



# Super Resolution 이란?



저화질 ->  
고화질

# Super Resolution 이란?

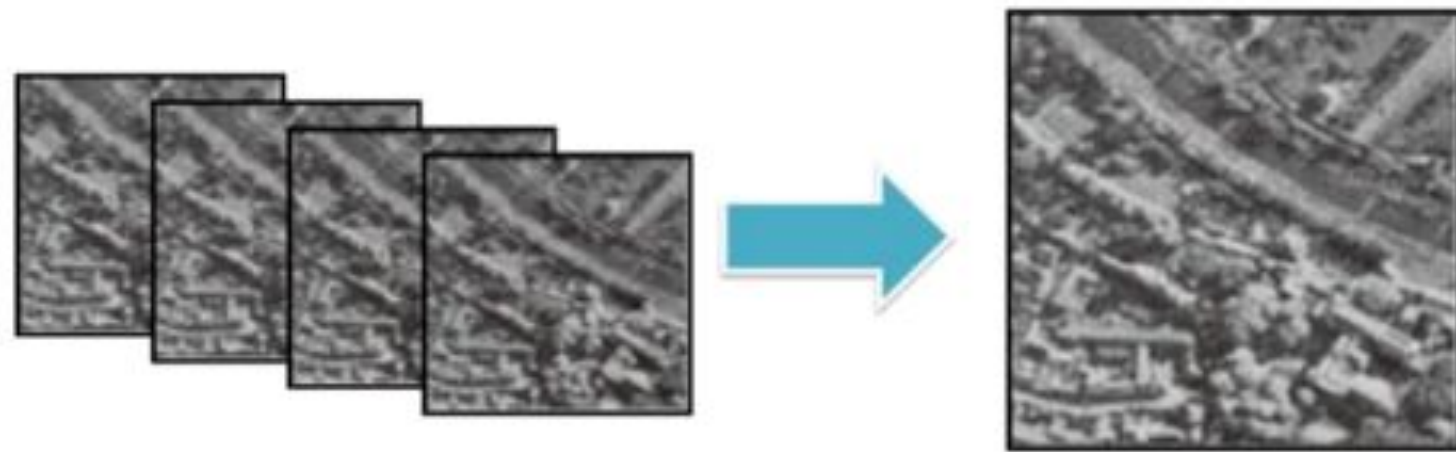


Fig 1: SR for satellite image [22]



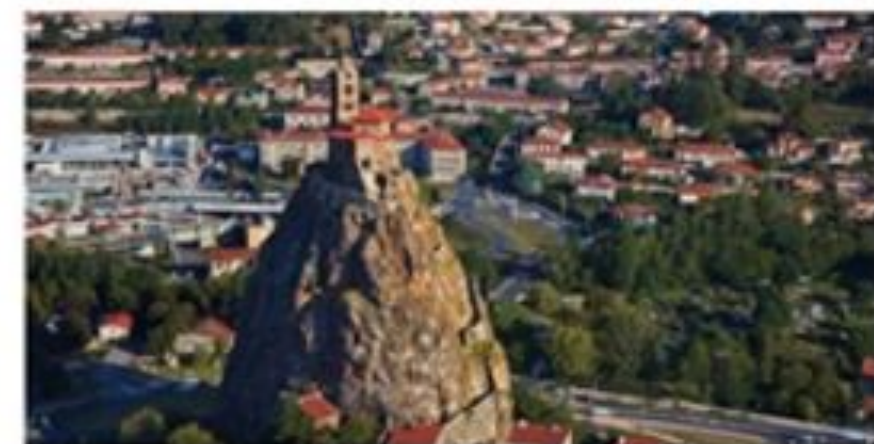
Fig 2: SR in Medical Imaging [23]

TV & Monitor



Before

HD(1280x720), FHD(1920x180)



AI Technology

UHD(3840x2160)



# Super Resolution을 평가하는 방법

## 1. PSNR

### PSNR (Peak Signal-to-Noise Ratio)

$$PSNR = 10 \log \frac{s^2}{MSE}$$

$s \rightarrow$  해당 영상의 최대값으로서, 해당 채널의 최대값에서 최소값을 빼서 구함  
8bit grayscale 영상의 경우 255(255-0) 이 된다.

- 최대 신호대 잡음비
- 신호가 가질 수 있는 최대 신호에 대한 잡음의 비를 나타냄
- 주로 영상 또는 동영상 손실 압축에서 화질 손실 정보를 평가할 때 사용
- 로그스케일에서 측정하기 때문에 단위는 [db] 이며, 손실이 적을수록 높은 값을 가짐
- 무손실 영상의 경우 MSE가 0이기 때문에 PSNR은 정의되지 않음

## 2. SSIM

### SSIM (Structural Similarity Index)

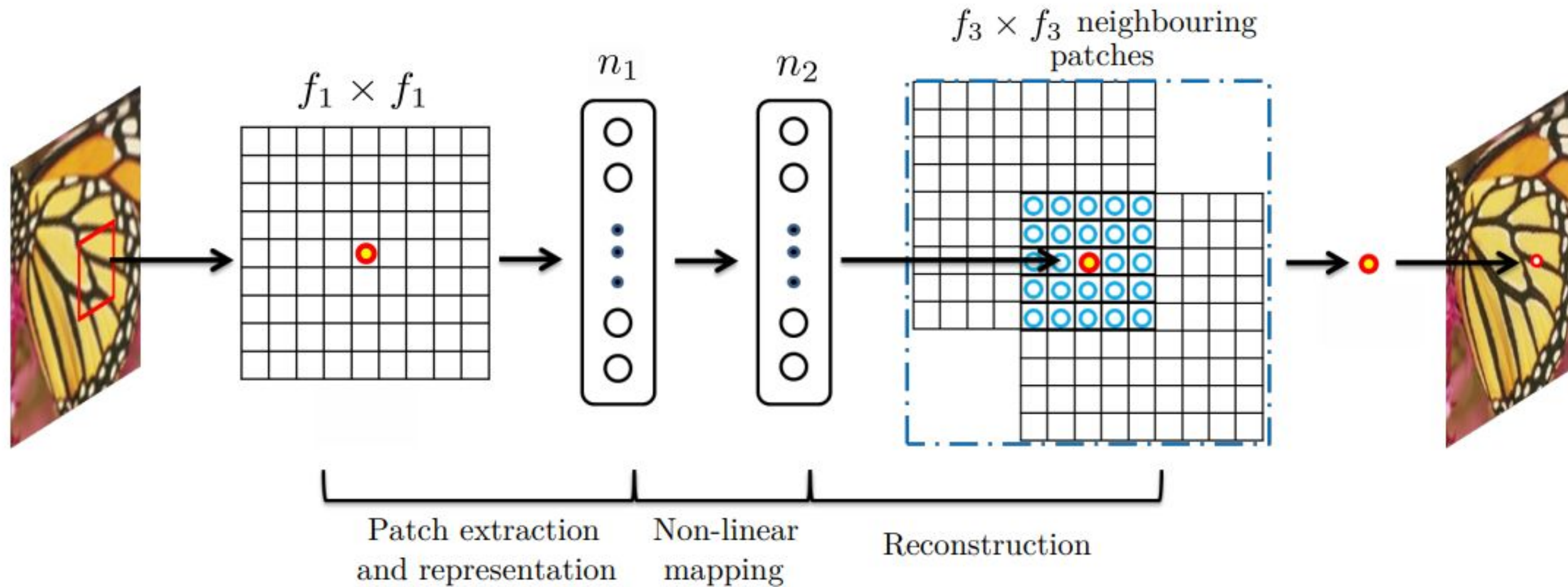
$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(2\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

- $\mu_x$  the average of  $x$ ;
- $\mu_y$  the average of  $y$ ;
- $\sigma_x^2$  the variance of  $x$ ;
- $\sigma_y^2$  the variance of  $y$ ;
- $\sigma_{xy}$  the covariance of  $x$  and  $y$ ;
- $c_1=(k_1 L)^2$ ,  $c_2=(k_2 L)^2$  two variables to stabilize the division with weak denominator;
- $L$  the dynamic range of the pixel-values (typically this is  $2^{\#bits \text{ per pixel}} - 1$ );
- $k_1=0.01$  and  $k_2=0.03$  by default.
- 구조적 유사 지수
- 압축 및 변환에 의해 발생하는 왜곡에 대하여 원본 영상에 대한 유사도를 측정하는 방법



# How to do SR with Deep Learning?

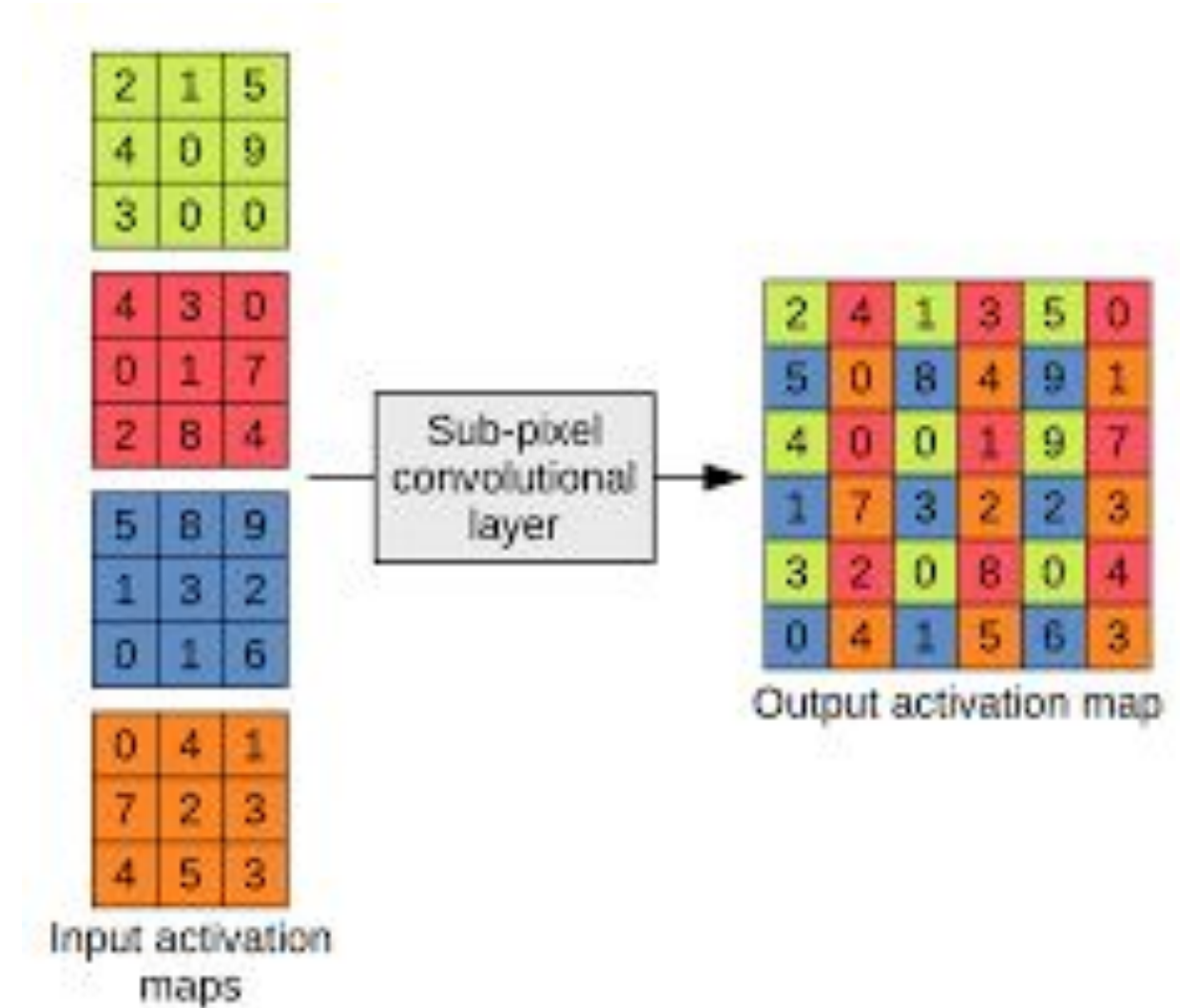
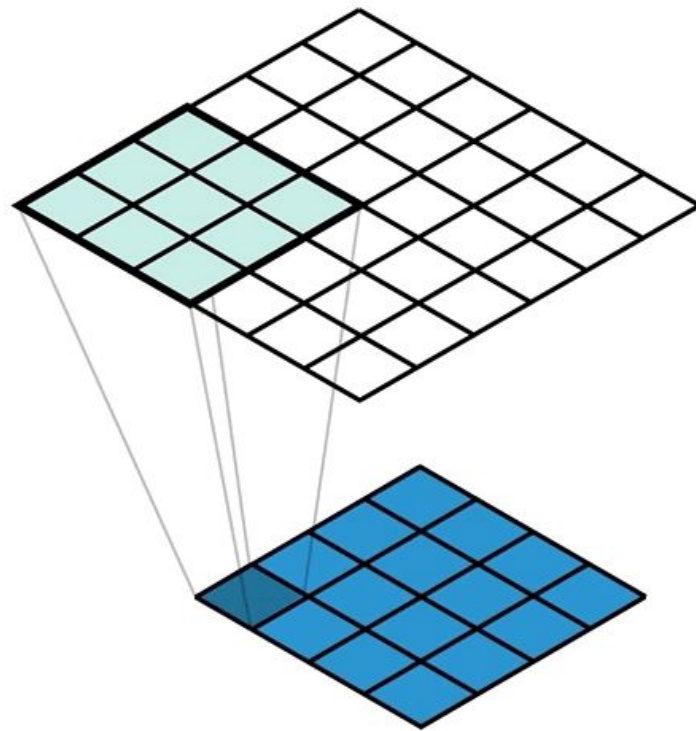
| SRCNN



1. Pooling을 사용하지 않음( 우리의 목적인 Upscaling과 무관할 뿐 아니라 정보의 손실을 유발 )
2. Deconvolutional Network를 통해 Upscale 효과를 부여  
(ex.  $64 \times 64$  크기의 이미지  $\rightarrow 128 \times 128$  크기의 이미지) (이 과정을 Reconstruction으로 부르기도 함)
3. 마지막 Convolution Layer에서는 Activation Function을 sigmoid로 사용하여 RGB 채널을 뽑아내는 형태로 구성 ( ex.  $128 \times 128 \times 32$  tensor  $\rightarrow 128 \times 128 \times 3$  image)

# About Deconvolutional Network

| Subpixel vs Transpose



1. Transpose layer은 행렬곱 연산을 통하여 upscale을 진행
2. Subpixel layer같은 경우에는 Convolution layer 에서 나온 filter의 정보를 재구성하여 upscale을 진행

# RDN : Residual Dense Network

| Total Structure

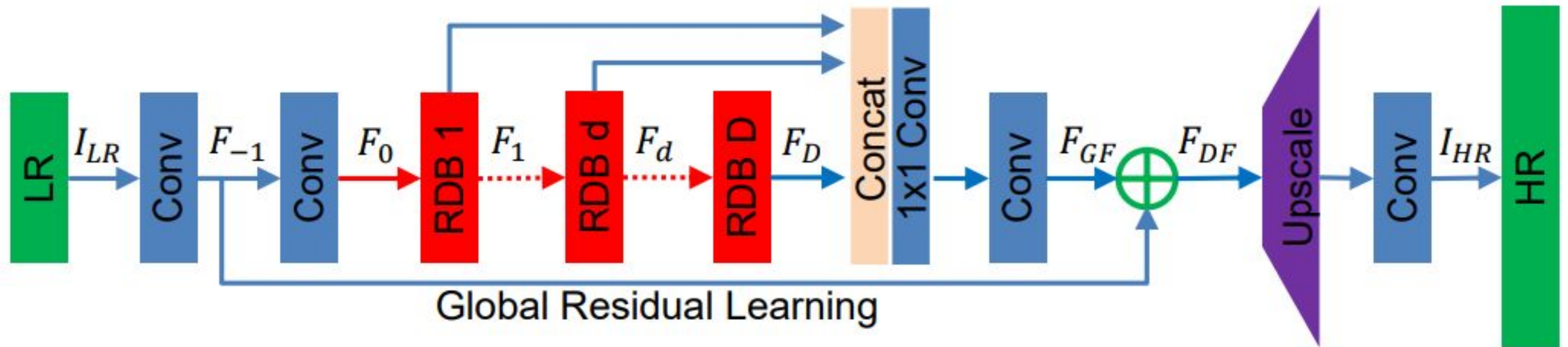
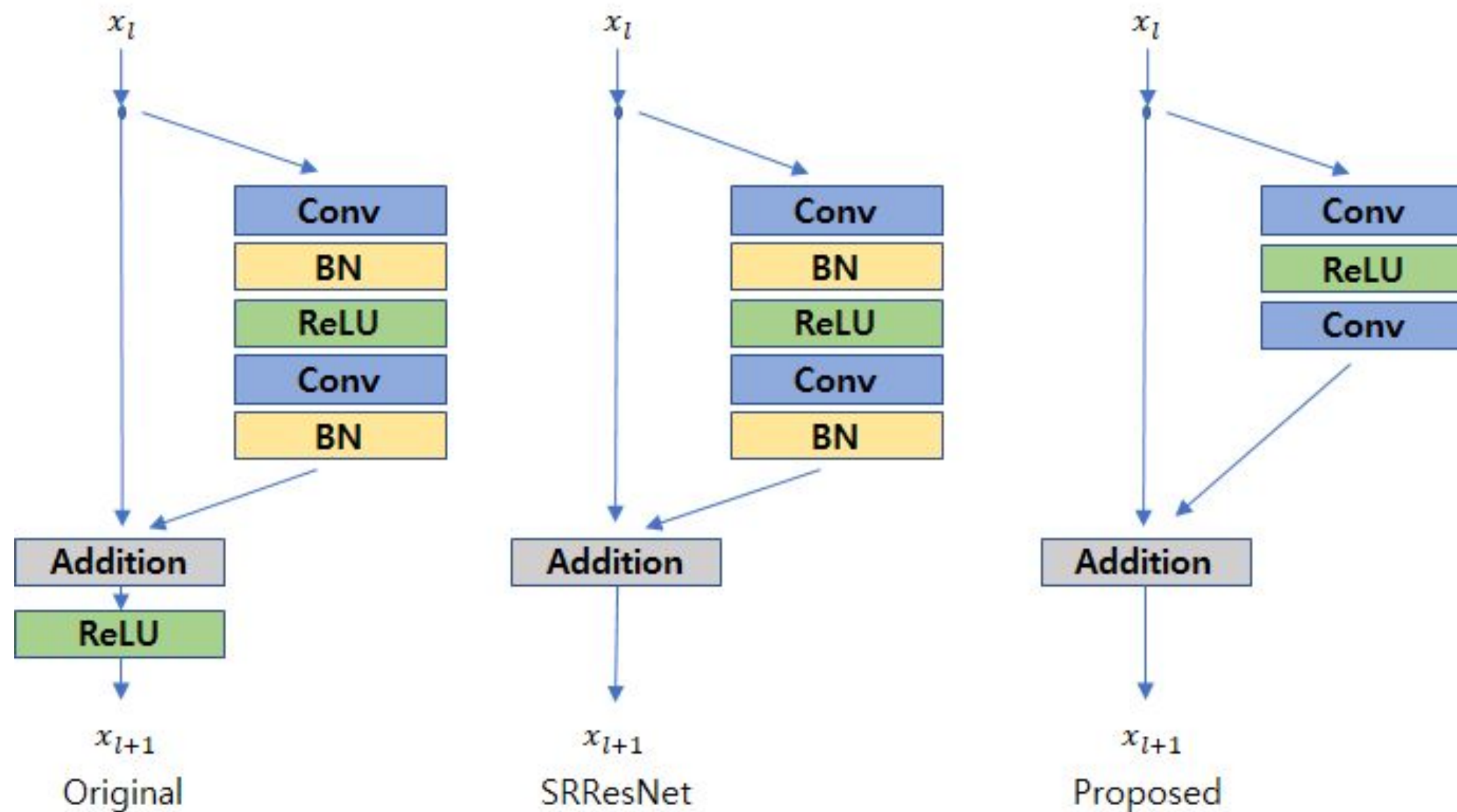


Figure 2. The architecture of our proposed residual dense network (RDN).



# Three method about using residual block in SR

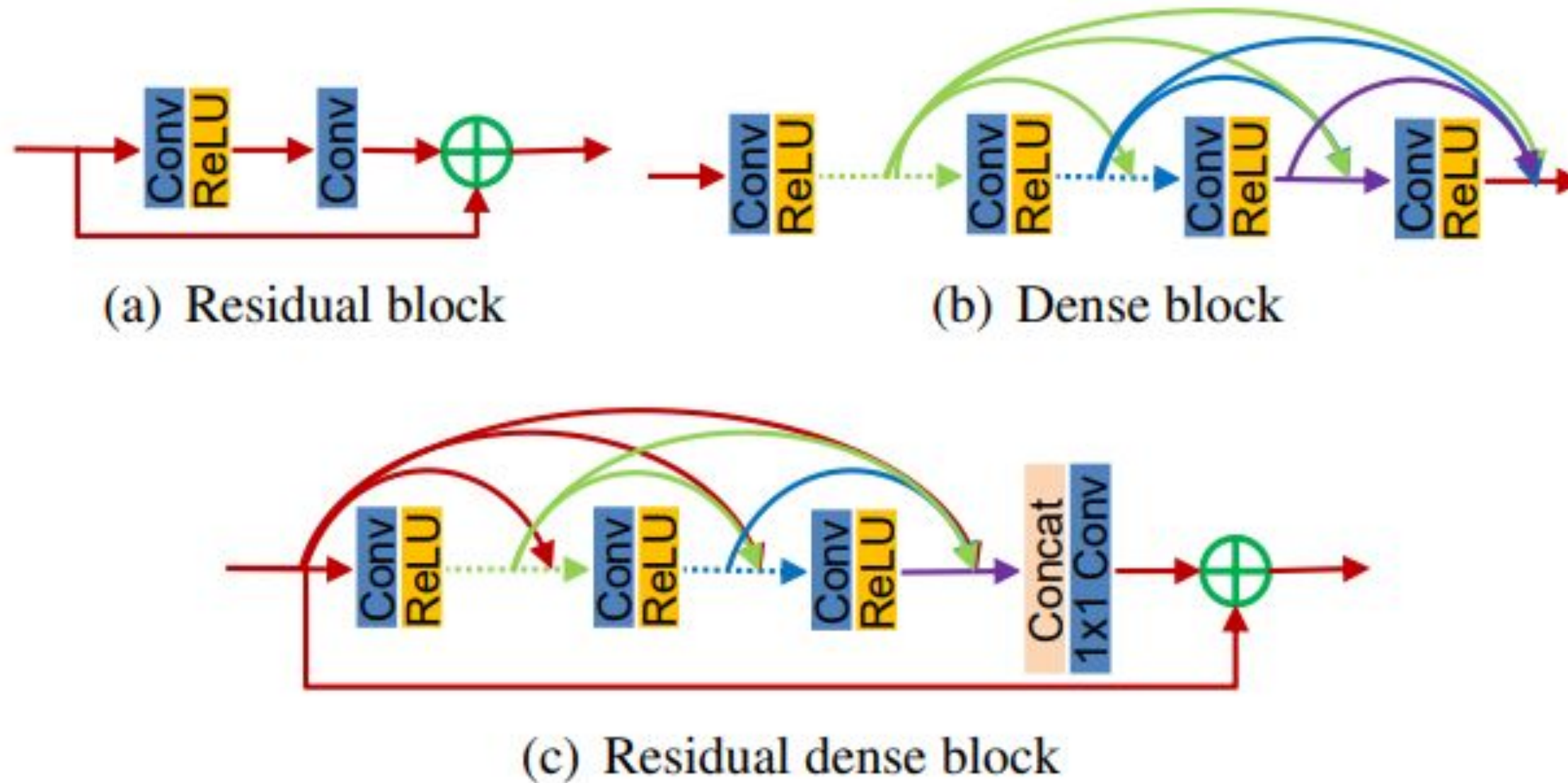
EDSR



Batch Normalization을 제거 -> 사용 메모리를 줄임 -> 효율적인 Resource 활용

# Residual Dense Network - RDB

| Residual Dense Block



Residual Block (a) 에서 사용한 skip connection과 Dense block에서 사용한 각각의 convolution layer 간의 skip Connection (b) 을 섞어서 만든 Block.

(논문에서는 (c)와 달리 convolutional layer의 개수를 6개로 설정)

# Residual Dense Network - RDB

| Residual Dense Block

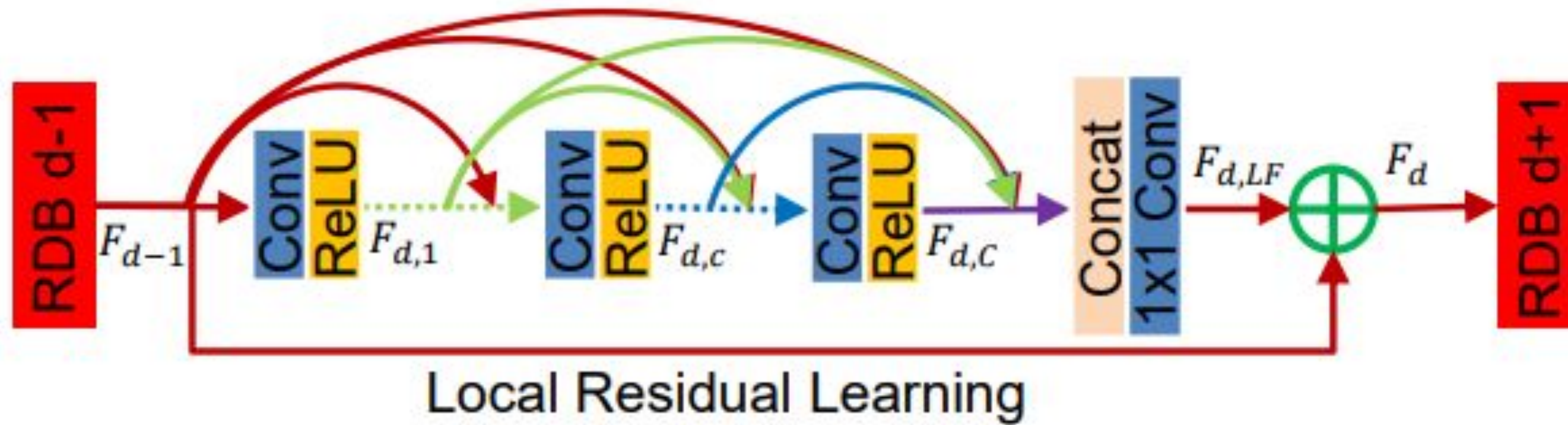


Figure 3. Residual dense block (RDB) architecture.

RDN에서 각각의 RDB는  $F_{d-1}$  (이전 항) 만이  $F_d$  (현재 항)에 영향을 미치는 형태의 구조를 가짐



# Residual Dense Network

| Total Structure

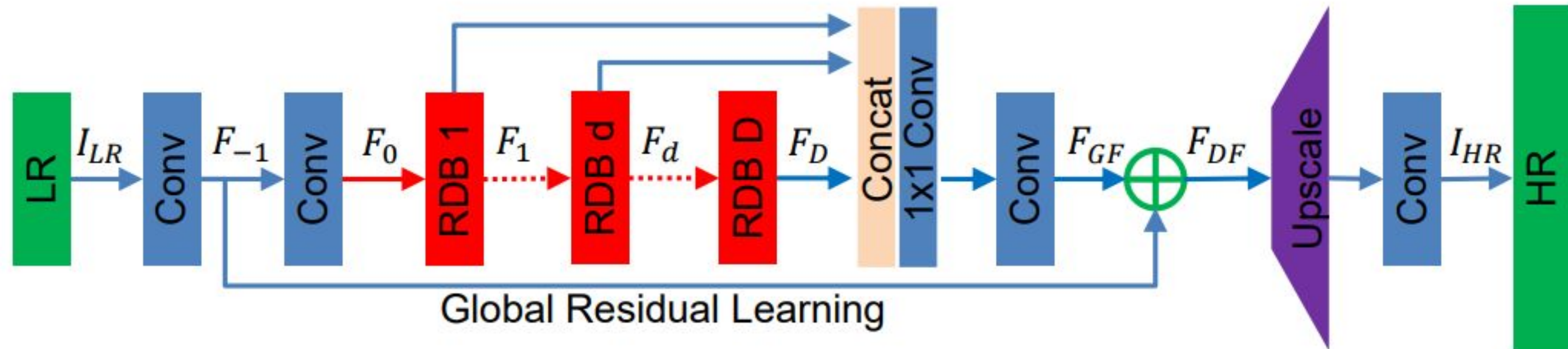


Figure 2. The architecture of our proposed residual dense network (RDN).

1. 각각의 CL(Convolutional Layer)에 L1 regulation을 부여 (L2 regulation보다 가중치에 대하여 더 robust한 효과를 가짐)
2. Global Residual Learning을 통하여 RDB에서 소실 될 수 있는 Global Information을 보존하여 학습을 진행
3. Upscale Layer에서는 Subpixel Layer를 사용하여 SR을 진행

# Our Problems & Solutions

프로젝트 진행이 안되요...

```
ResourceExhaustedError: 2 root error(s) found.  
  (0) Resource exhausted: OOM when allocating tensor with shape[420128,512] and type float32  
    [[{{node training_1/Adam/mul_23}}]]  
Hint: If you want to see a list of allocated tensors when OOM happens, add report_tensor_allocations_upon_oom  
  (1) Resource exhausted: OOM when allocating tensor with shape[420128,512] and type float32  
    [[{{node training_1/Adam/mul_23}}]]  
Hint: If you want to see a list of allocated tensors when OOM happens, add report_tensor_allocations_upon_oom  
  
0 successful operations.  
0 derived errors ignored.
```

```
~\anaconda3\envs\tf2.4\lib\site-packages\six.py in rais  
ResourceExhaustedError: OOM when allocating tensor with
```





# Our Problems & Solutions

프로젝트 진행이 안되요... => Batch\_size 줄여보기=>1달 걸림

**When you decrease the batch size to handle the CUDA OOM error.**

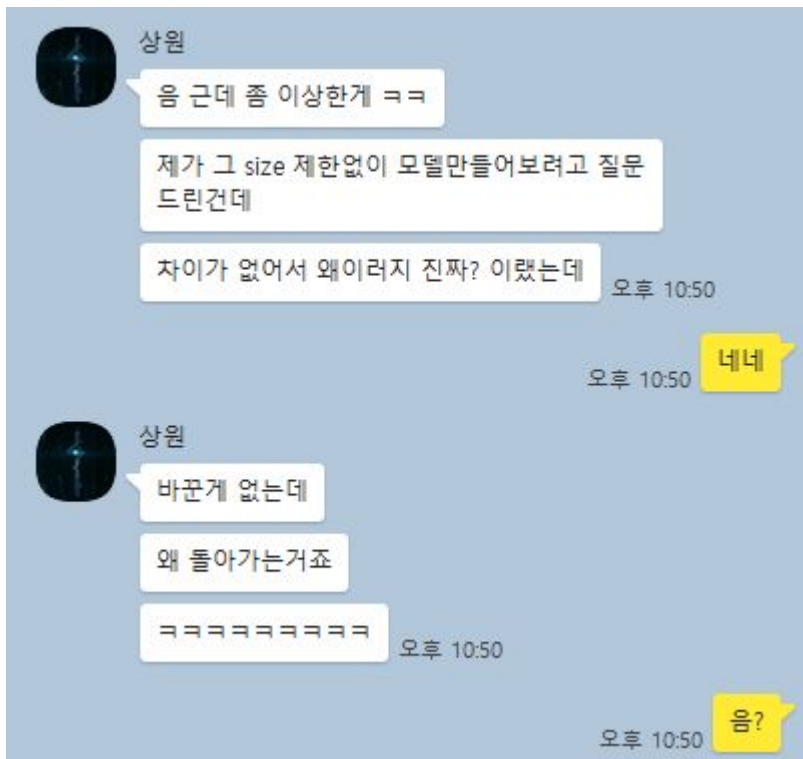


VS

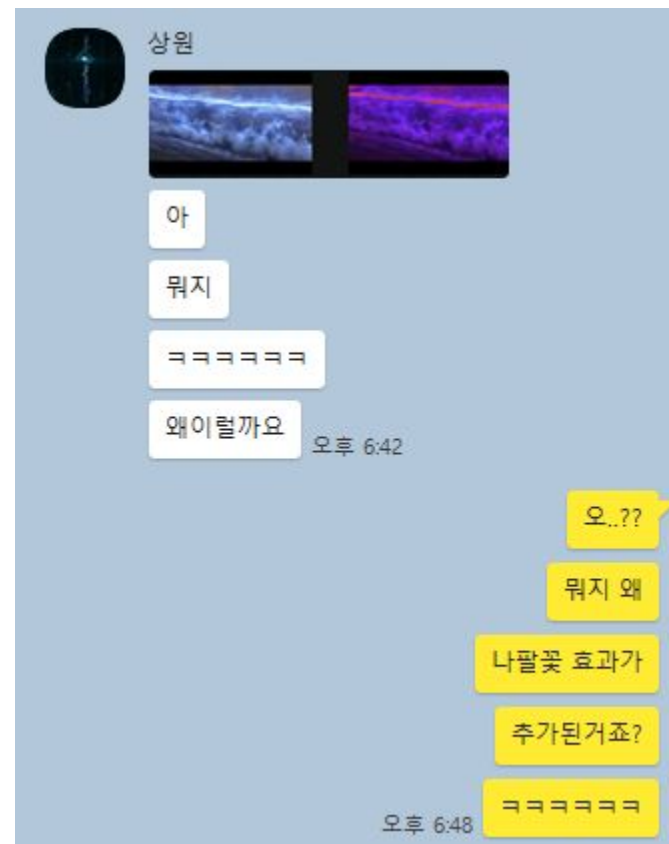


# Our Problems & Solutions

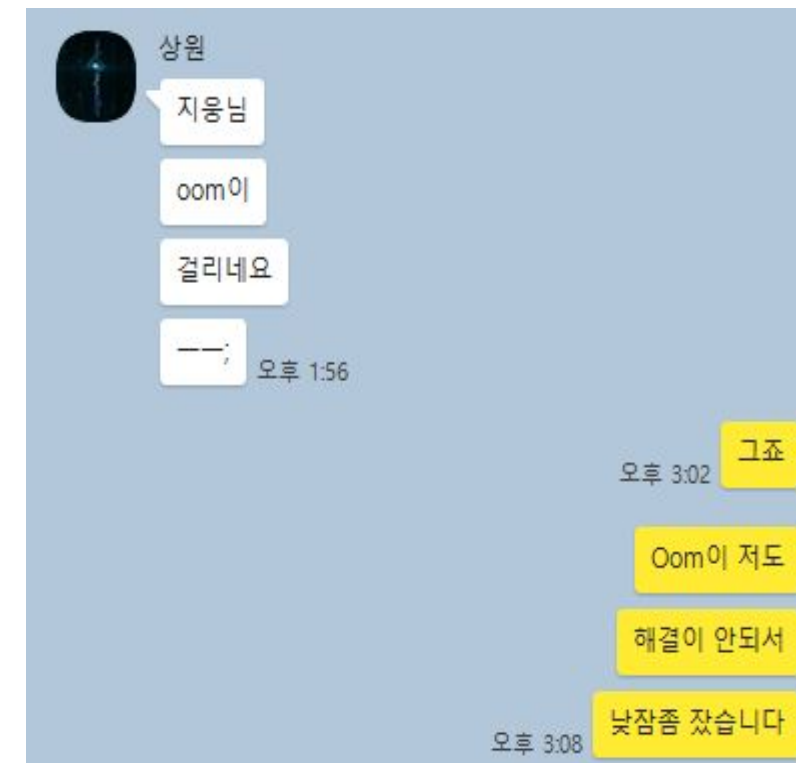
고생의 흔적들...(살려주세요)



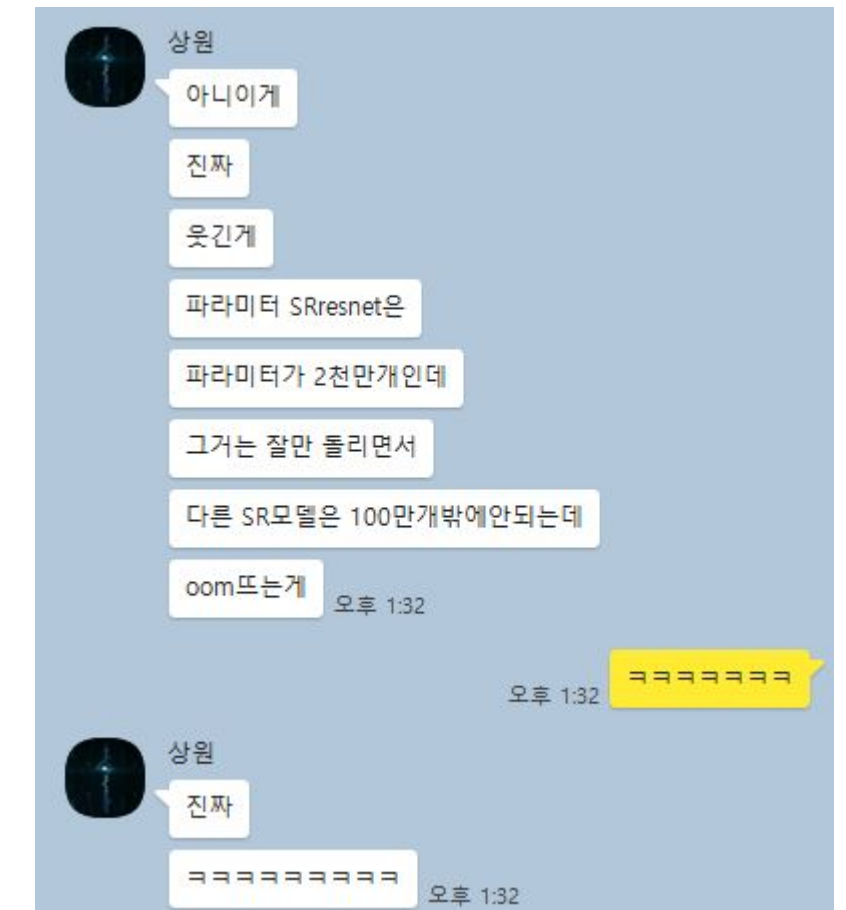
바꾼게 없는데 코드가  
돌아감



레이저 영상인데 나팔꽃  
나옴  
(뭐임 진짜?)



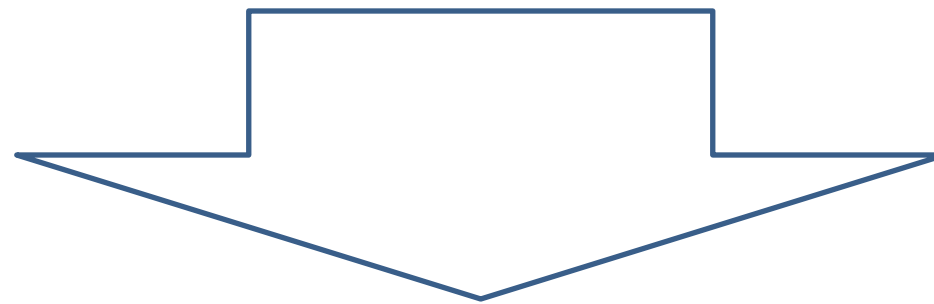
OOM은 모르겠고 낮잠이나  
자야겠다



웃는게 웃는게 아님  
(개발 2주차)

# Our Problems & Solutions

1. 열악한 작업 환경 (Colab pro이라는 가장 좋은 개발환경에서도 버틸 수 가 없다)
2. 그로 인한 무거운 모델을 사용하지 못함 (기존 RDN은 약 4천만개의 파라미터를 사용 but 우리는 100만개만 되도 OOM)
3. 게다가 OOM 때문에 batch\_size를 적게 부여 -> training time이 과하게 길어짐(영원히 학습됨 영원히)



Total params: 889,923  
Trainable params: 889,923  
Non-trainable params: 0

이게 터지는게 말이 됩니까?



## Solution

1. 기존의 RDN을 대폭 축소시킨 Light Model을 사용. (RDB 내에 Convolutional layer 수: 6 -> 4, RDB의 숫자: 20 -> 5)
2. 트레이일러 내의 전체 이미지가 아닌 1초마다 찍힌 image를 Train data로 사용 □ 효율적인 학습 & training time 감소
3. 모델 성능 향상을 위해 기존 논문에서는 Subpixel을 사용했지만, 대신에 Transpose layer를 사용



# Trials – Custom Loss Function

## | MSE & PSNR Loss

기존의 MSE Loss 를 기본으로 하되 ,추가적으로 psnr loss를 활용하여 Model Training이 적절히 이루어지고 있는지 확인

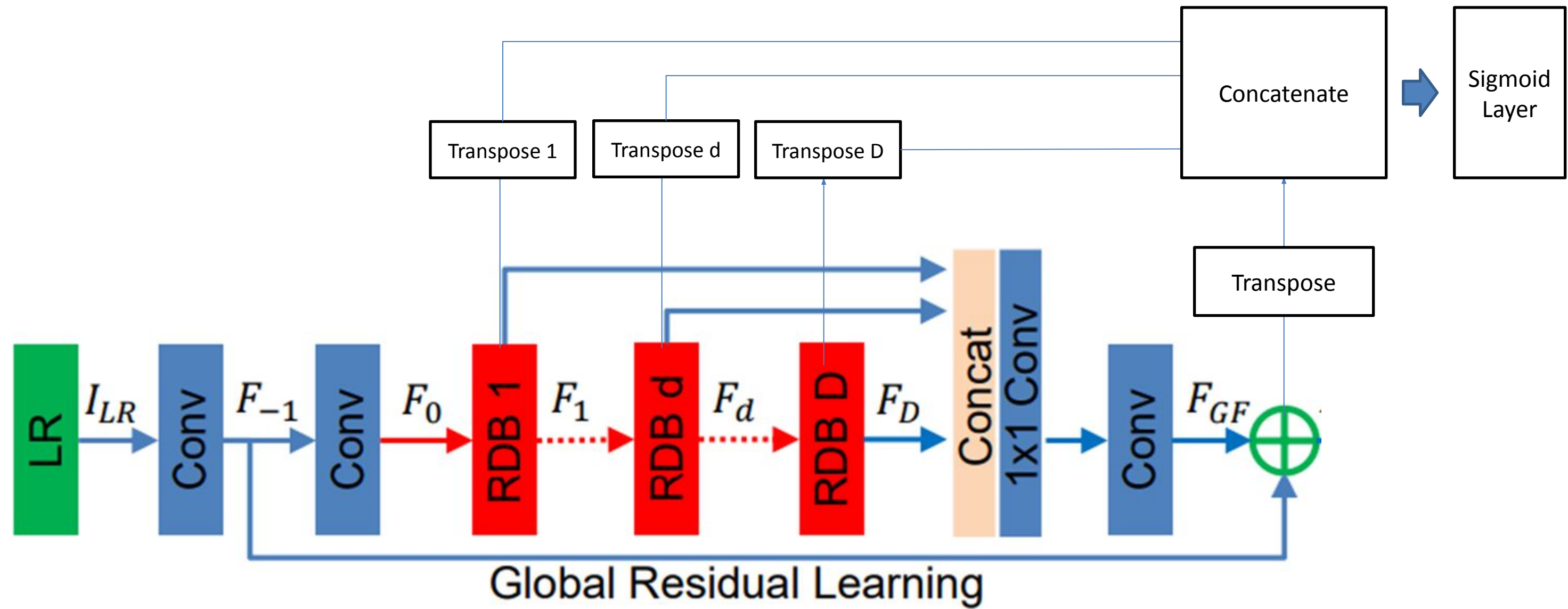
```
def psnr_loss(a,b):  
    x = tf.image.psnr(a, b, 1, name=None)  
    return x
```

```
def ssim(y_true, y_pred):  
    return K.expand_dims(tf.image.ssim(y_true, y_pred, 255.), 0)
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=25e-5, decay = 1e-5)  
model.compile(optimizer=optimizer, loss = tf.keras.losses.MeanSquaredError(), metrics=[psnr_loss])
```



# Trials – Residual Dense Transpose Network

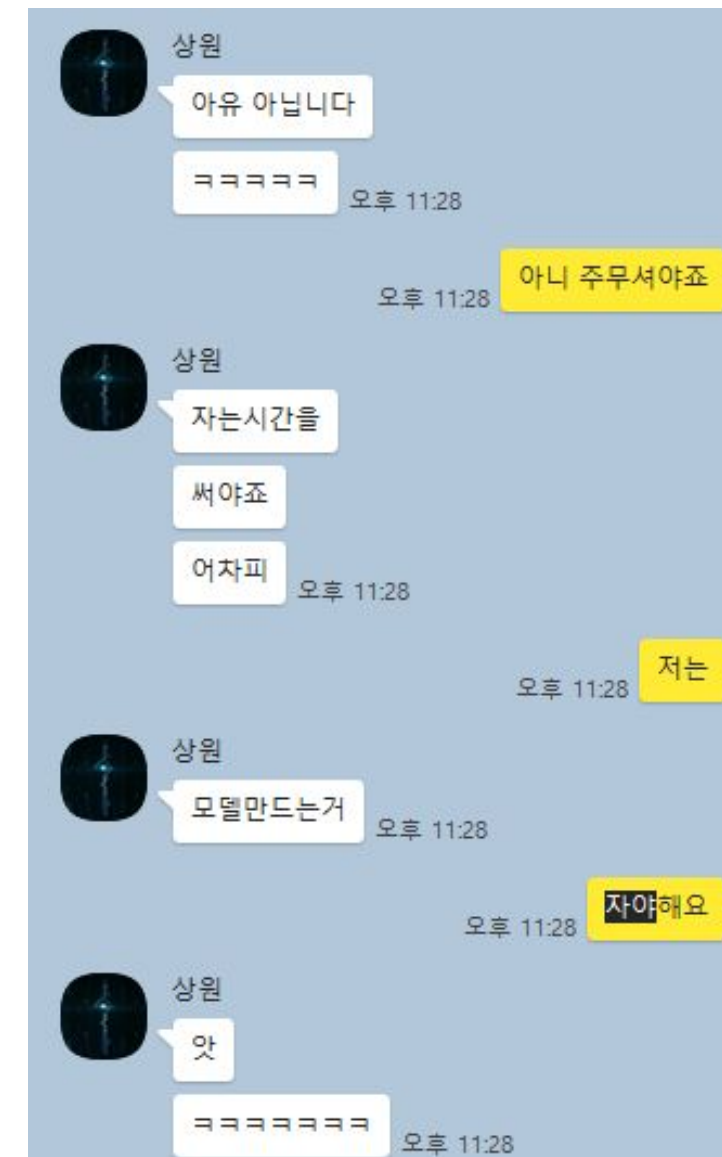


# PSNR Test

(모델이 예측한 이미지의 PSNR) -  
(기존의 보간법으로 만든 이미지의 PSNR)

0 번째 : 4.586807084096762  
1 번째 : 6.294196355844861  
2 번째 : 4.510441592167339  
3 번째 : 4.785234142400306  
4 번째 : 3.0774229860777  
5 번째 : 3.6585866069876616  
6 번째 : 3.725929319671323  
7 번째 : 2.9912164809460933  
8 번째 : 2.930274062653851  
9 번째 : 1.7375296980210067  
10 번째 : 1.6996239356845493  
11 번째 : 2.9424685431181317  
12 번째 : 2.436337043867951  
13 번째 : 3.7491327968721606  
14 번째 : 2.339336056309733  
15 번째 : 4.083245743290377  
16 번째 : 8.100278511604337  
17 번째 : 3.4829206755317728  
18 번째 : 2.9422120572422017  
19 번째 : 3.239090562131288  
20 번째 : 2.2325513450562653  
21 번째 : 2.6860712688378996  
22 번째 : 7.7124932045190775  
23 번째 : 2.86741115256401  
24 번째 : 2.7986627426629056  
25 번째 : 4.2010696589127505  
26 번째 : 3.5196261477283173  
27 번째 : 2.591439328145846  
28 번째 : 4.247993546016964  
29 번째 : 14.880969447490713  
original mean psnr : 37.265637711370445  
predict mean psnr : 41.30065678125225

기존의 보간법(bicubic) 대비 새로 적용한 RDTN이  
평균적으로 약 4 정도의 향상을 보였고, 모든 image에서  
기존의 보간법 보다 더 좋은 성능을 보여줌



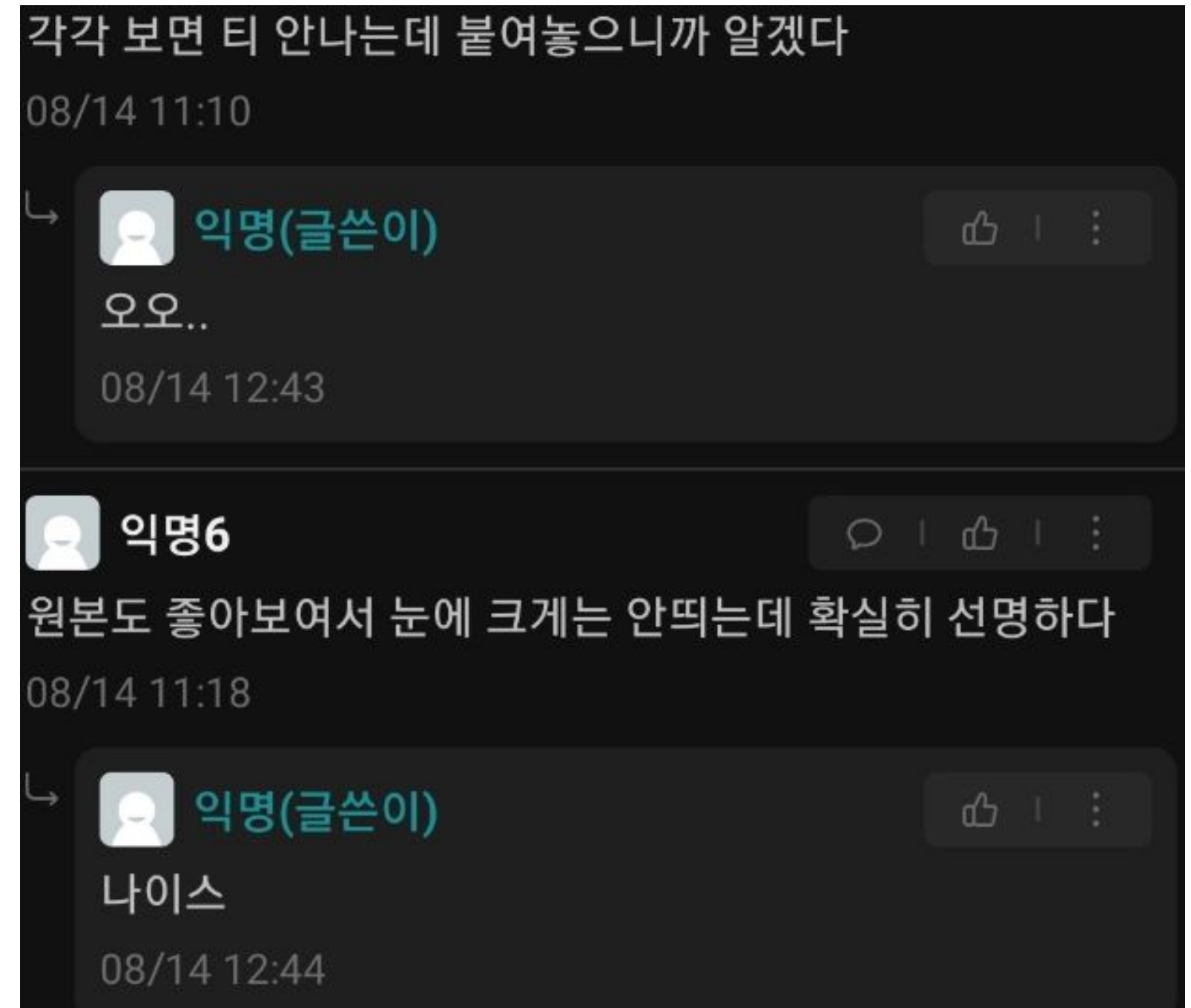
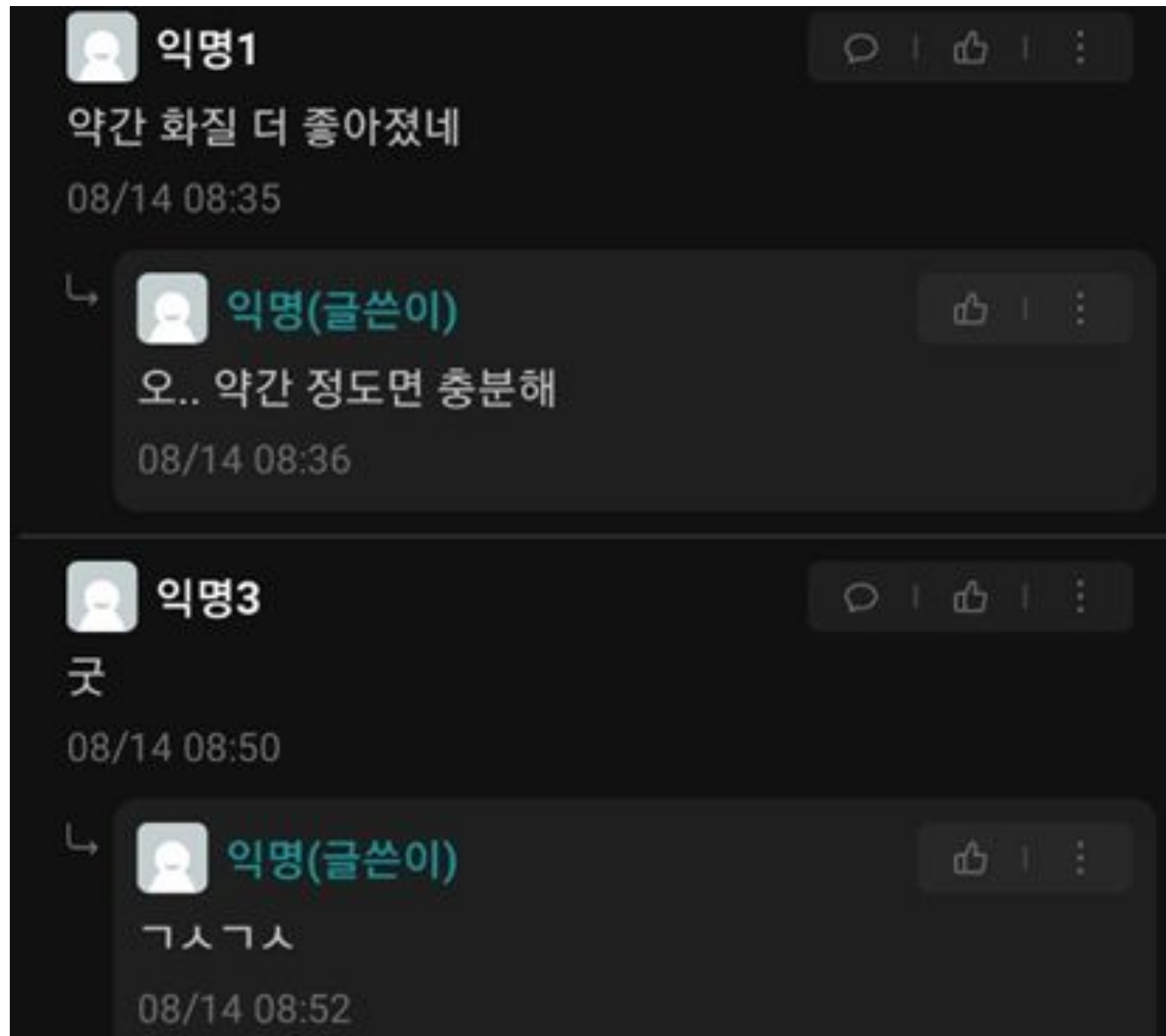
(덕분에 드디어 잘 수 있었습니다)

# Bicubic vs RDTN

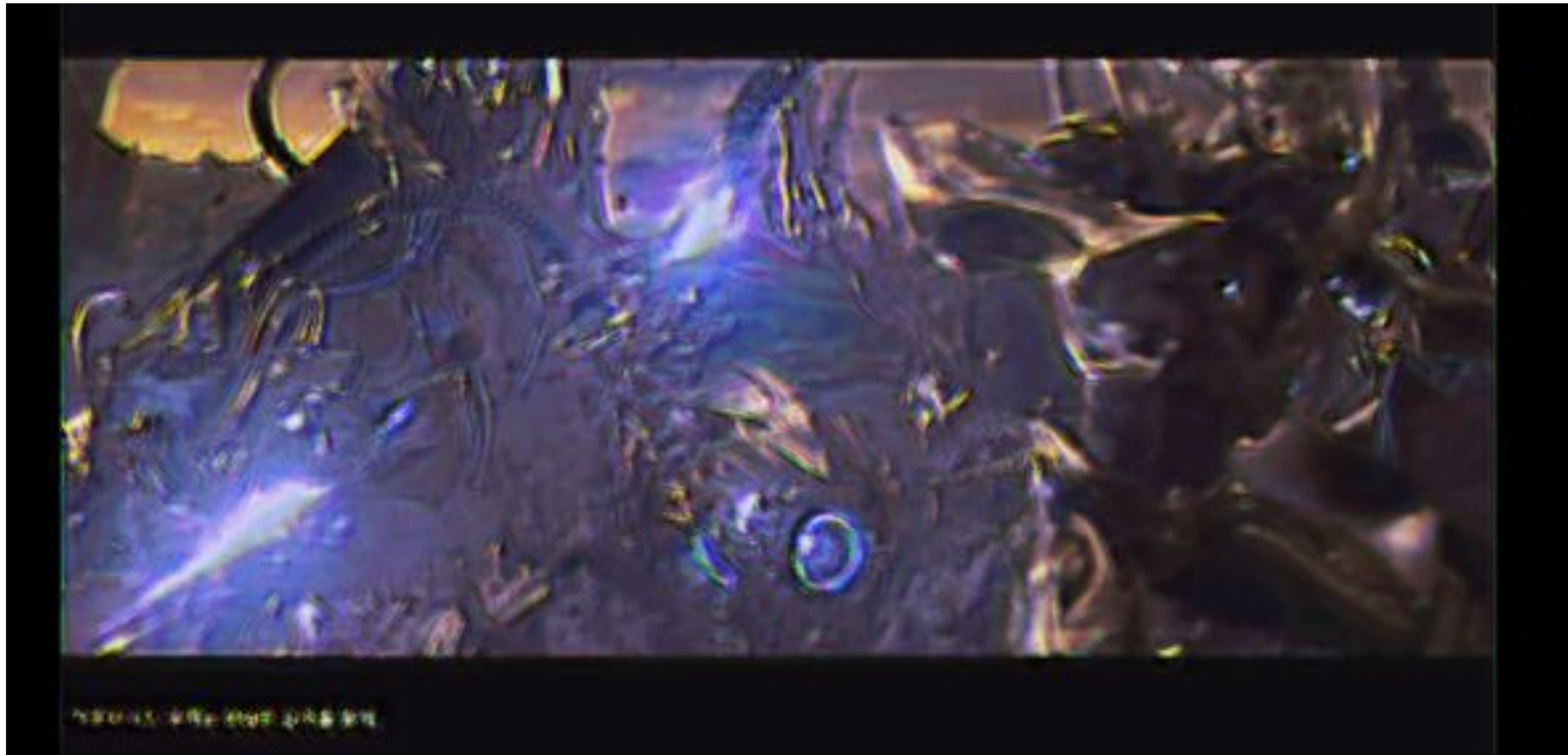




# Bicubic vs RDTN



# 실패작 – mse를 무시하게 된다면?



# 영상 시연





2021 – 1 학기 SAI A – CV 팀 프로젝트 발표

# 감사합니다.

