Pro GAN

KNOU GAN STUDY (2021)
JH.KIM

Pro GAN Resources

- 논문 원본
  ([1710.10196] Progressive Growing of GANs for Improved Quality, Stability, and Variation)
- 한글 요약
  (PGGAN(PGAN, ProGAN)논문 Full Reading - Progressive Growing of GANs For Imporved Quality, Stability and Variation)
- 논문 깃헙
  tkarras/progressive_growing_of_gans: Progressive Growing of GANs for Improved Quality, Stability, and Variation
- 논문 trained networks in Google Drive
  ProgressiveGAN
- 논문 보충 비디오 (Youtube)
  Progressive Growing of GANs for Improved Quality, Stability, and Variation
- GAN in Action 교재 6장 깃헙
  https://colab.research.google.com/github/rickiepark/gans-in-action/blob/master/chapter-6/Chapter_6_ProGAN.ipynb

베이스라인

- Baseline 논문
  [R][1704.00028] Improved Training of Wasserstein GANs
- Baseline github
  https://github.com/igul222/improved_wgan_training

기존 GAN 모델들의 단점

- autoregressive models(van den Oord et al., 2016b;c)
  slow to evaluate and do not have a latent representation as they directly model the conditional distribution over pixels, potentially limiting their applicability.
- variational autoencoders (VAE) (Kingma & Welling, 2014)
  tend to produce blurry results due to restrictions in the model
- generative adversarial networks (GAN) (Goodfellow et al., 2014)
  only in fairly small resolutions and with somewhat limited variation, and the training continues to be unstable

고해상도 이미지 생성이 어려운 이유

- higher resolution makes it easier to tell the generated images apart from training images (Odena et al., 2017), thus drastically amplifying the gradient problem
- using smaller mini batches due to memory constraints, further compromising training stability.

⇨ 미니배치 사이즈를 메모리 사용 효율화를 위해 다이내믹하게 조정
(4,4) ~ (128,128) : 16
(256,256) : 14
(512,512) : 6
(1024,1024) : 3

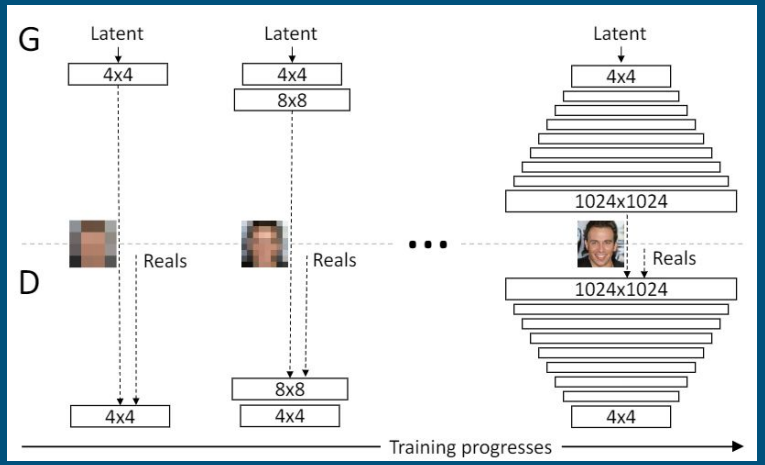
Distance metric

- Jensen-Shannon divergence (Goodfellow et al., 2014)
- least squares (Mao et al., 2016b)
- absolute deviation with margin (Zhao et al., 2017)
- Wasserstein distance (Arjovsky et al., 2017)
  [딥러닝]Wasserstein distance 에 관하여 Magritte Magritte
- 이 논문에서 사용한 loss
  WGAN-GP loss (Gulrajani et al., 2017)
  LSGAN loss (Mao et al., 2016b)

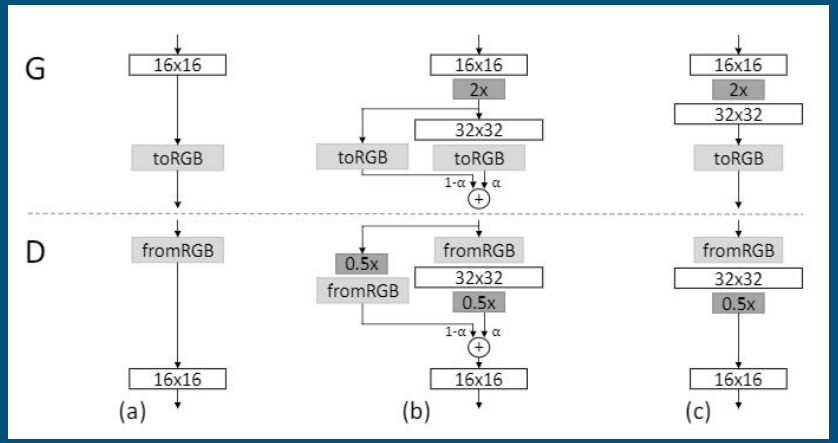
4가지 혁신

- 고해상도 층으로 점진적 증대와 단계적 도입
  해상도를 점진적으로 늘린다. 새 층을 부드럽게 주입한다.
- 미니배치 표준편차로 변동성 증가
  이미지 해상도와 변동성간의 트레이드오프 해소, 변동성 증가 방안
- 균등 학습률 (EQUALIZED LEARNING RATE)
  딥러닝 네트워크 초기화의 수정으로, 층간의 학습속도차이에 균형을 달성
- 생성자의 픽셀별 특성 정규화
  모드붕괴(Mode collapse) 를 방지하기 위한 방법

혁신1. 고해상도 층으로 점진적 증대와 단계적 도입



As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process



During the transition (b) we treat the layers that operate on the higher resolution like a residual block, whose weight α increases linearly from 0 to 1

⇨ stabilizes & reduced time

모델의 실제 구조

Generator	Act.	Output shape	Params
Latent vector	–	512 × 1 × 1	–
Conv 4 × 4	LReLU	512 × 4 × 4	4.2M
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Upsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Upsample	–	512 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Upsample	–	512 × 32 × 32	–
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Upsample	–	512 × 64 × 64	–
Conv 3 × 3	LReLU	256 × 64 × 64	1.2M
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Upsample	–	256 × 128 × 128	–
Conv 3 × 3	LReLU	128 × 128 × 128	295k
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Upsample	–	128 × 256 × 256	–
Conv 3 × 3	LReLU	64 × 256 × 256	74k
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Upsample	–	64 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	18k
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Upsample	–	32 × 1024 × 1024	–
Conv 3 × 3	LReLU	16 × 1024 × 1024	4.6k
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 1 × 1	linear	3 × 1024 × 1024	51
Total trainable parameters			**23.1M**

Discriminator	Act.	Output shape	Params
Input image	–	3 × 1024 × 1024	–
Conv 1 × 1	LReLU	16 × 1024 × 1024	64
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 3 × 3	LReLU	32 × 1024 × 1024	4.6k
Downsample	–	32 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Conv 3 × 3	LReLU	64 × 512 × 512	18k
Downsample	–	64 × 256 × 256	–
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Conv 3 × 3	LReLU	128 × 256 × 256	74k
Downsample	–	128 × 128 × 128	–
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Conv 3 × 3	LReLU	256 × 128 × 128	295k
Downsample	–	256 × 64 × 64	–
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Conv 3 × 3	LReLU	512 × 64 × 64	1.2M
Downsample	–	512 × 32 × 32	–
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Downsample	–	512 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Downsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Downsample	–	512 × 4 × 4	–
Minibatch stddev	–	513 × 4 × 4	–
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Conv 4 × 4	LReLU	512 × 1 × 1	4.2M
Fully-connected	linear	1 × 1 × 1	513
Total trainable parameters			**23.1M**

- CELEBA-HQ 훈련에 사용
- replicated 3-layer blocks
- Conv1×1 : toRGB, fromRGB block
- Fade In : discriminator 가 800k real image를 처리할 때마다 3-layer block을 추가하고, 안정화과정을 거친다.
- 잠재벡터 : 512의 길이
- 이미지표현 : [-1,1] 의 실수공간
- leaky ReLU with leakiness 0.2
- batch normalization, layer normalization, weight normalization 안한다.
- pixelwise normal-ization of the feature vectors : 생성자의 Conv 3x3 층 다음에 매번 수행
- minibatch standard

```python
def upscale_layer(layer, upscale_factor):
    '''
    upscale_factor(int)만큼 층(텐서)을 업스케일합니다.
    텐서 크기는 [group, height, width, channels]입니다.
    '''
    height, width = layer.get_shape()[1:3]
    size = (upscale_factor * height, upscale_factor * width)
    upscaled_layer = tf.image.resize_nearest_neighbor(layer, size)
    return upscaled_layer
```

Nearest-neighbor interpolation

The nearest neighbor algorithm selects the value of the nearest point and does not consider the values of neighboring points at all, yielding a piecewise-constant interpolation.

```python
def smoothly_merge_last_layer(list_of_layers, alpha):
    '''
    임곗값 알파를 기반으로 층을 부드럽게 합칩니다.
    이 함수는 모든 층이 이미 RGB로 바뀌었다고 가정합니다.
    생성자를 위한 함수입니다.
    :list_of_layers    :   해상도(크기) 순서대로 정렬된 텐서 리스트
    :alpha             :   (0,1) 사이의 실수
    '''
    # 업스케일링을 위해 끝에서 두 번째 층을 선택합니다.
    last_fully_trained_layer = list_of_layers[-2]
    # 마지막으로 훈련된 층을 업스케일링합니다.
    last_layer_upscaled = upscale_layer(last_fully_trained_layer, 2)

    # 새로 추가된 층은 아직 완전히 훈련되지 않았습니다.
    larger_native_layer = list_of_layers[-1]

    # 합치기 전에 층 크기가 같은지 확인합니다.
    assert larger_native_layer.get_shape() == last_layer_upscaled.get_shape()

    # 곱셈은 브로드캐스팅되어 수행됩니다.
    new_layer = (1-alpha) * last_layer_upscaled + larger_native_layer * alpha

    return new_layer
```

혁신2. 미니배치 표준편차로 변동성 증가

- GANs have a tendency to capture only a subset of the variation found in training data, and Salimanset al. (2016) suggest “minibatch discrimination” as a solution.

1. first compute the standard deviation for each feature in each spatial location over the minibatch
2. average these estimates over all features and spatial locations to arrive at a single value
3. replicate the value and concatenate it to all spatial locations and over the minibatch, yielding one additional (constant) feature map

```python
def minibatch_std_layer(layer, group_size=4):
    '''
    층의 미니배치 표준편차를 계산합니다.
    층의 데이터 타입은 float32로 가정합니다. 그렇지 않으면 타입 변환이 필요합니다.
    '''
    # 미니배치는 group_size로 나눌 수 있거나 group_size 보다 같거나 작아야 합니다.
    group_size = K.backend.minimum(group_size, tf.shape(layer)[0])

    # 간단하게 쓰기 위해 크기 정보를 따로 저장합니다.
    # 그래프 실행 전에는 일반적으로 배치 차원이 None이기 때문에 tf.shape에서 이 크기를 얻습니다.
    shape = list(K.int_shape(input))
    shape[0] = tf.shape(input)[0]

    # 미니배치 수준에서 연산하기 위해 크기를 바꿉니다.
    # 이 코드는 층이 [그룹(G), 미니배치(M), 너비(W), 높이(H), 채널(C)]라고 가정합니다.
    # 하지만 씨아노(Theano) 방식의 순서를 사용하는 구현도 있습니다.
    minibatch = K.backend.reshape(layer, (group_size, -1, shape[1], shape[2], shape[3]))

    # [M, W, H, C] 그룹의 평균을 계산합니다.
    minibatch -= tf.reduce_mean(minibatch, axis=0, keepdims=True)
    # [M, W, H, C] 그룹의 분산을 계산합니다.
    minibatch = tf.reduce_mean(K.backend.square(minibatch), axis = 0)
    # [M,W,H,C] 그룹의 표준편차를 계산합니다.
    minibatch = K.backend.square(minibatch + 1e-8)
    # 특성 맵을 평균하여 [M,1,1,1] 픽셀을 얻습니다.
    minibatch = tf.reduce_mean(minibatch, axis=[1,2,3], keepdims=True)
    # 스칼라 값을 그룹과 픽셀에 맞게 변환합니다.
    minibatch = K.backend.tile(minibatch, [group_size, 1, shape[2], shape[3]])
    # 새로운 특성 맵을 추가합니다.
    return K.backend.concatenate([layer, minibatch], axis=1)
```

혁신3. 균등 학습률 (EQUALIZED LEARNING RATE)

- wi=wi/c
  where wi are the weights and c is the per-layer normalization constant from He's initializer (Heet al., 2015)
- relates to the scale-invariance in commonly used adaptive stochastic gradient descent methods such as RMSProp (Tieleman & Hinton, 2012) and Adam (Kingma & Ba, 2015), These methods normalize a gradient update by its estimated standard deviation, it is possible that a learning rate is both too large and too small at the same time
- Our approach ensures that the dynamic range, and thus the learning speed, is the same for all weight
- Scale the weights with a layer-specific constant

```
def equalize_learning_rate(shape, gain, fan_in=None):
    '''
    He 초기화의 상수로 모든 층의 가중치를 조정하여
    특성마다 각기 다른 다이나믹 레인지를 가지도록 분산을 맞춥니다.
    shape   :   텐서(층)의 크기: 각 층의 차원입니다.
        예를 들어, [4,4,48,3]. 이 경우 [커널크기, 커널크기, 필터개수, 특성맵]입니다.
        하지만 구현에 따라 조금 다를 수 있습니다.
    gain    :   일반적으로 sqrt(2)
    fan_in  :   세이비어/He 초기화에서 입력 연결 개수
    '''
    # 기본값은 특성 맵 차원을 제외하고 shape의 모든 차원을 곱합니다. 이를 통해 뉴런마다 입력 연결 개수를 얻습니다.
    if fan_in is None: fan_in = np.prod(shape[:-1])
    # He 초기화 상수 (He et al, 2015)
    std = gain / K.sqrt(fan_in)
    # 조정을 위한 상수를 만듭니다.
    wscale = K.constant(std, name='wscale', dtype=np.float32)
    # 가중치 값을 얻어 브로드캐스팅으로 wscale을 적용합니다.
    adjusted_weights = K.get_value('layer', shape=shape,
            initializer=tf.initializers.random_normal()) * wscale
    return adjusted_weights
```

혁신4. 생성자의 픽셀별 특성 정규화

- 목적: To disallow the scenario where the magnitudes in the generator and discriminator spiral out of control as a result of competition
- normalize the feature vector in each pixel to unit length in the generator after each convolutional layer
- using a variant of “local response normalization” (Krizhevsky et al., 2012)
- with most datasets it does not change the results much, but it prevents the escalation of signal magnitudes very effectively when needed

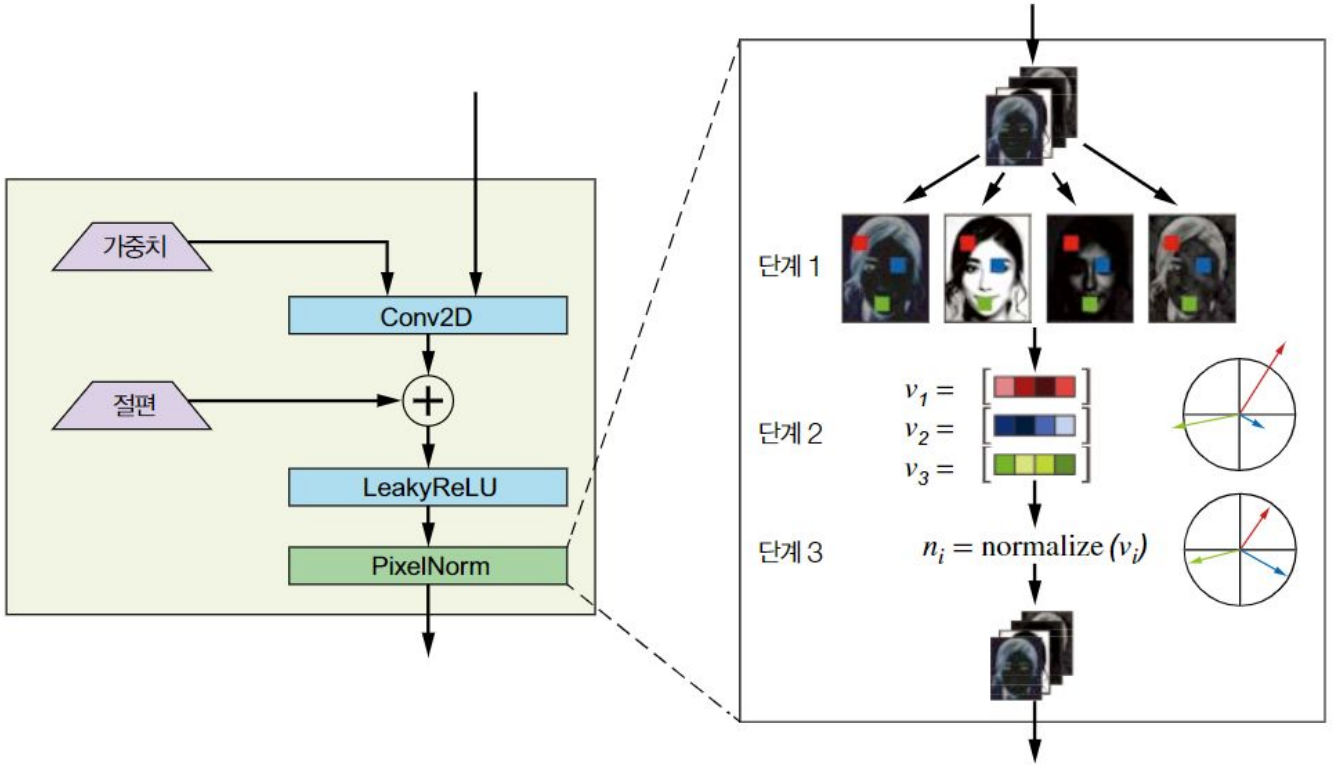


그림 6-4 이미지에 있는 모든 포인트(단계 1)를 일련의 벡터로 매핑합니다(단계 2). 그다음 이를 정규화하여 동일한 범위(일반적으로 고차원 공간에서 0과 1 사이)로 만듭니다(단계 3).

```
def pixelwise_feat_norm(inputs, **kwargs):
    '''
    Krizhevsky 등이 2012년 논문에 제안한 픽셀별 특성 정규화
    :inputs : 케라스 / TF 층
    '''
    normalization_constant = K.backend.sqrt(K.backend.mean(
                    inputs**2, axis=-1, keepdims=True) + 1.0e-8)
    return inputs / normalization_constant
```

평가지표

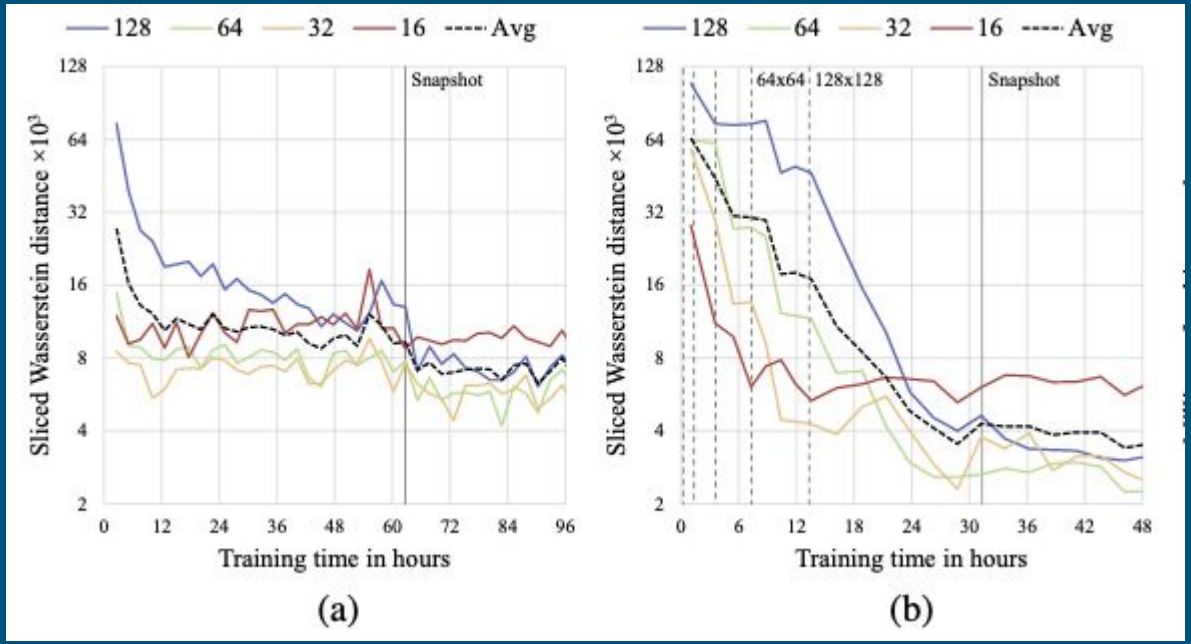
- 수작업 확인 : tedious, difficult, and subjective
- MS-SSIM (Odena et al., 2017) : fail to react to smaller effects such as loss of variation in colors or textures, do not directly assess image quality in terms of similarity to the training set
- the multi-scale statistical similarity between distributions of local image patches drawn from Laplacian pyramid (Burt & Adelson, 1987) representations of generated and target images ([OpenCV Practice 14] 이미지 피라미드 (Image Pyramid))
- Sliced Wasserstein distance (SWD)
  The distance between the patch sets extracted from the lowest-resolution 16×16 images indicate similarity in large-scale image structures, while the finest-level patches encode information about pixel-level attributes such as sharpness of edges and noise
- Fŕechet Inception Distance (FID)
  (https://cyc1am3n.github.io/2020/03/01/is_fid.html)

혁신의 효과

Training configuration	CELEBA						LSUN BEDROOM					
	Sliced Wasserstein distance $\times 10^3$					MS-SSIM	Sliced Wasserstein distance $\times 10^3$					MS-SSIM
	128	64	32	16	Avg		128	64	32	16	Avg	
(a) Gulrajani et al. (2017)	12.99	7.79	7.62	8.73	9.28	0.2854	11.97	10.51	8.03	14.48	11.25	**0.0587**
(b) + Progressive growing	4.62	**2.64**	3.78	6.06	4.28	**0.2838**	7.09	6.27	7.40	9.64	7.60	0.0615
(c) + Small minibatch	75.42	41.33	41.62	26.57	46.23	0.4065	72.73	40.16	42.75	42.46	49.52	0.1061
(d) + Revised training parameters	9.20	6.53	4.71	11.84	8.07	0.3027	7.39	5.51	3.65	9.63	6.54	0.0662
(e*) + Minibatch discrimination	10.76	6.28	6.04	16.29	9.84	0.3057	10.29	6.22	5.32	11.88	8.43	0.0648
(e) Minibatch stddev	13.94	5.67	2.82	5.71	7.04	0.2950	7.77	5.23	3.27	9.64	6.48	0.0671
(f) + Equalized learning rate	4.42	3.28	2.32	7.52	4.39	0.2902	**3.61**	3.32	**2.71**	6.44	4.02	0.0668
(g) + Pixelwise normalization	**4.06**	3.04	**2.02**	**5.13**	**3.56**	0.2845	3.89	**3.05**	3.24	**5.87**	**4.01**	0.0640
(h) Converged	2.42	2.17	2.24	4.99	2.96	0.2828	3.47	2.60	2.30	4.87	3.31	0.0636

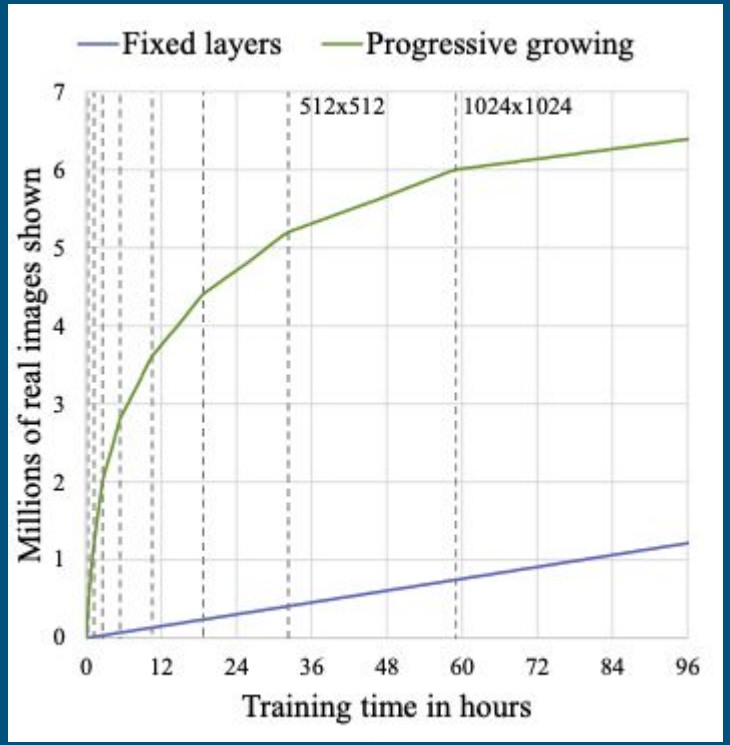
(a) (b) (c) (d) (e*) (e) (f) (g) (h) Converged

수렴



- 훈련 모델 : Gulrajani et al. (2017)
- (b) 는 progressive growing 적용
- 수렴과 훈련시간 단축 효과
- (a) 전체레이어가 동시에 학습됨
- (b) 라지스케일의 가중치가 가장 먼저 수렴되어 유지됨 (16)
- (b) 32, 64 순으로 순서대로 수렴됨

훈련 시간



- progressive growing 이 초반에 이미지 처리 속도가 빠르다. 왜냐하면, 초반에는 레이어가 간단하기 때문이다.
- 1024*1024의 레이어가 추가되면. 이미지 처리 속도가 고정레이어 방식과 동일해 진다.

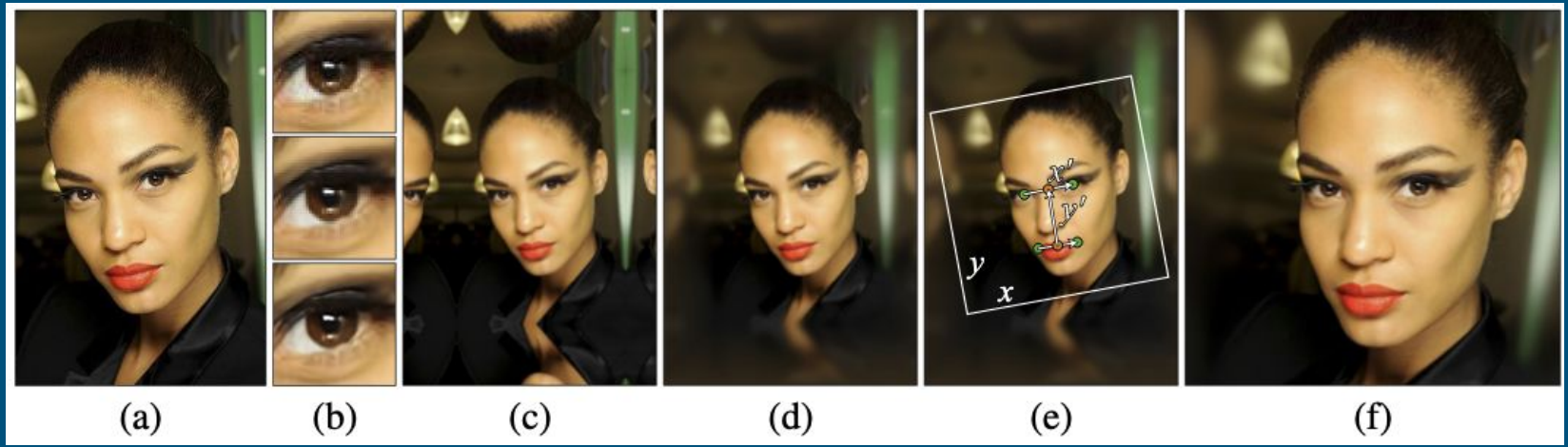
dataset

- CELEBA
  (https://www.tensorflow.org/datasets/catalog/celeb_a?hl=en)
- LSUN
  (https://www.tensorflow.org/datasets/catalog/lsun?hl=en)
- CIFAR10
  (https://www.tensorflow.org/datasets/catalog/cifar10?hl=en)

⇨ 고해상도 이미지 생성 훈련을 위해 이미지 전처리 작업 (CELEBA-HQ 생성) 필요

- CELEBA 해상도 : 43×55 ~ 6732×8984
- crowds of several people
- part of the face
- JPEG 포맷 : 이미지 압축 손실

CELEBA-HQ 생성 (1024x1024, 30000개)



(a) CELEBA DATASET 원본 이미지

(b) JPEG artifact removal (top -> middle)
convolutional autoencoder Mao et al.(2016a)
(bottom) 4x super-resolution (Korobchenko, 2017)

(c) 이미지 크기 증대 (Resizing) : mirror padding

(d) 화질 향상 : Gaussian filtering (피사계 심도 추가)

(e) 얼굴의 위치 자르기 : Facial Landmark Detection

(f) 1024x1024 로 이미지 화질 증대 : high-quality resampling
Image Resizing vs Resampling In Photoshop Explained

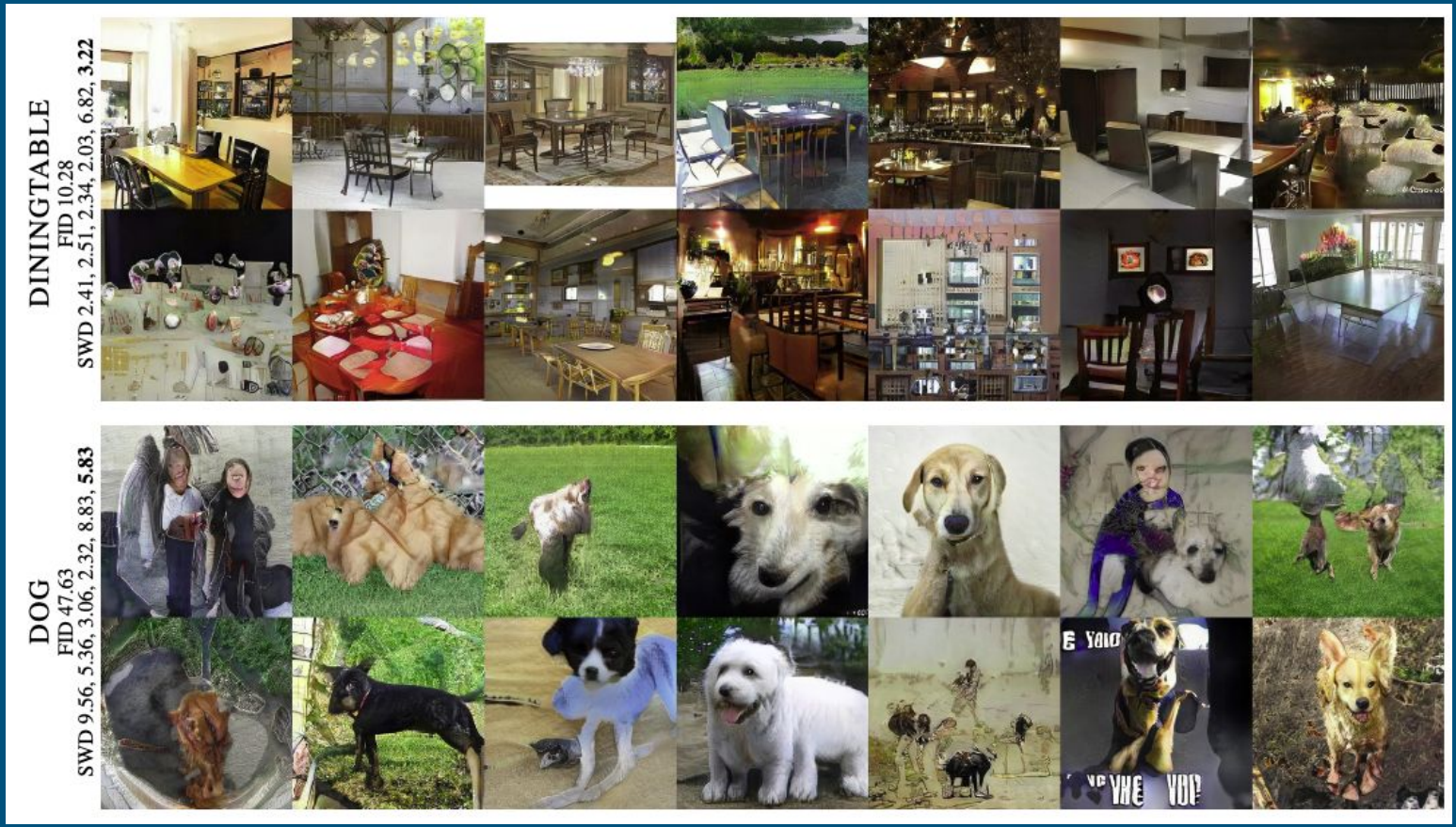
CELEBA-HQ 훈련 결과 (생성 이미지)

- Top : CELEBA-HQ 생성 이미지
- 아래행 : 기존 훈련셋에서 최근접이웃 검출 결과
- distance : feature-space distance (activations from five VGG layers) Chen & Koltun (2017)
- 훈련시간 : 8 Tesla V100 GPUs for 4 days
- 손실함수 : WGAN-GP

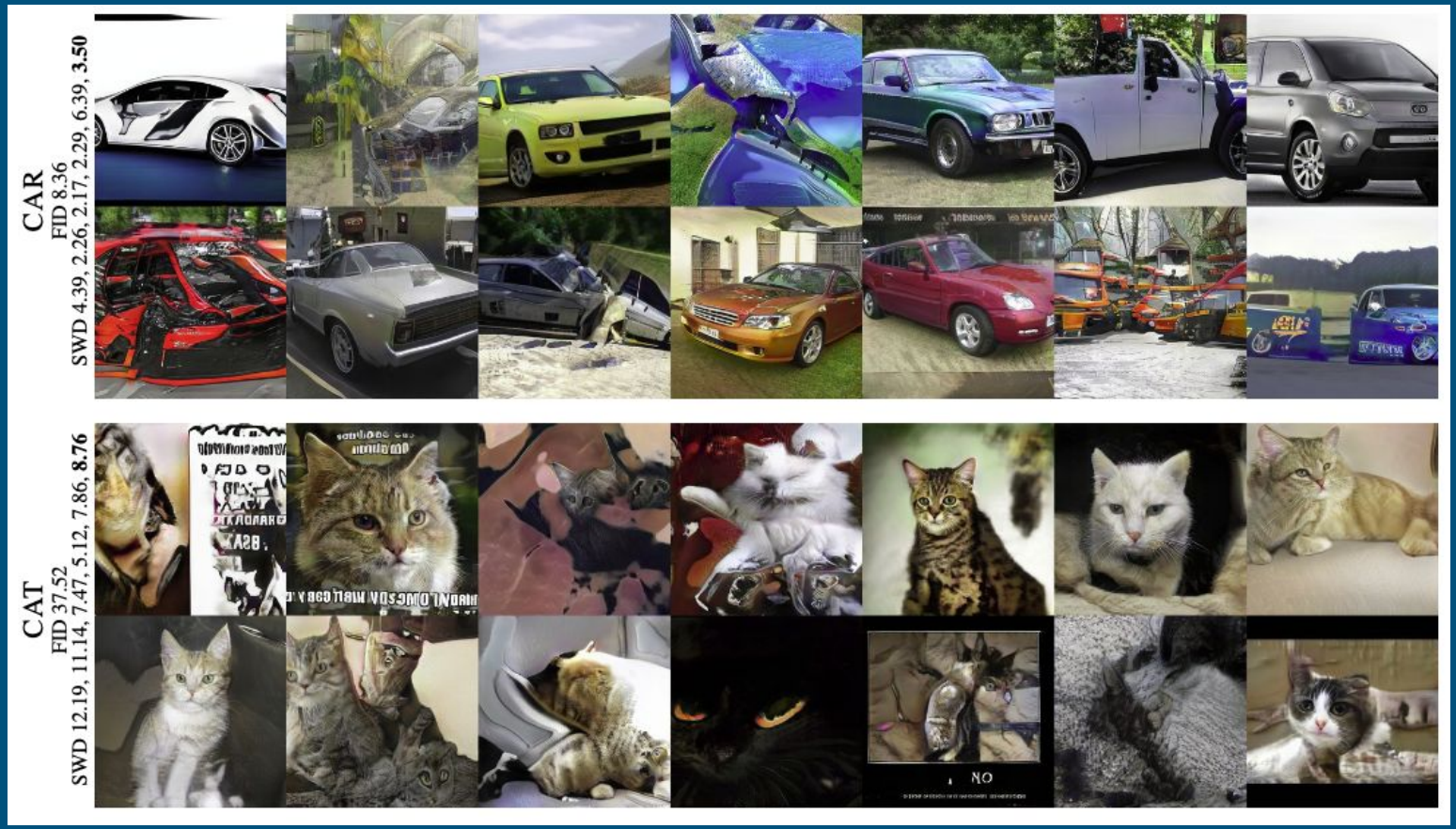
LSUN BEDROOMc

Mao et al. (2016b) (128 × 128) Gulrajani et al. (2017) (128 × 128) Our (256 × 256)

LSUN OTHERS


DINING TABLE
FID 10.28
SWD 2.41, 2.51, 2.34, 2.03, 6.82, 3.22
DOG
FID 47.63
SWD 9.56, 5.36, 3.06, 2.32, 8.83, 5.83

LSUN OTHERS


CAR
FID 8.36
SWD 4.39, 2.26, 2.17, 2.29, 6.39, 3.50
CAT
FID 37.52
SWD 12.19, 11.14, 7.47, 5.12, 7.86, 8.76

CIFAR10 인셉션 점수

Unsupervised		
Method		Inception score
ALI	(Dumoulin et al., 2016)	5.34 ± 0.05
GMAN	(Durugkar et al., 2016)	6.00 ± 0.19
Improved GAN	(Salimans et al., 2016)	6.86 ± 0.06
CEGAN-Ent-VI	(Dai et al., 2017)	7.07 ± 0.07
LR-AGN	(Yang et al., 2017)	7.17 ± 0.17
DFM	(Warde-Farley & Bengio, 2017)	7.72 ± 0.13
WGAN-GP	(Gulrajani et al., 2017)	7.86 ± 0.07
Splitting GAN	(Grinblat et al., 2017)	7.90 ± 0.09
Our (best run)		$\mathbf{8.80 \pm 0.05}$
Our (computed from 10 runs)		$\mathbf{8.56 \pm 0.06}$

Label conditioned		
Method		Inception score
DCGAN	(Radford et al., 2015)	6.58
Improved GAN	(Salimans et al., 2016)	8.09 ± 0.07
AC-GAN	(Odena et al., 2017)	8.25 ± 0.07
SGAN	(Huang et al., 2016)	8.59 ± 0.12
WGAN-GP	(Gulrajani et al., 2017)	8.67 ± 0.14
Splitting GAN	(Grinblat et al., 2017)	8.87 ± 0.09

1) 훈련하는 동안 최고 스코어 (here ± refers to the standard deviation returned by the inception score calculator)
2) 최고 스코어일 때의 평균과 표준편차 (starting from ten random initializations)

GAN 활용

- speech synthesis (van den Oord et al., 2016a)
- image-to-image translation (Zhu et al., 2017; Liu et al., 2017; Wang et al., 2017)
- image inpainting (Iizuka et al., 2017)
  (https://worthpreading.tistory.com/64)
- High-Resolution Mammogram Synthesis using Progressive Generative Adversarial Networks
  (https://arxiv.org/abs/1807.03401)

텐서플로 허브를 사용한 실습

```
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow_hub as hub

# TFHub에서 ProGAN을 임포트합니다.
module = hub.KerasLayer("https://tfhub.dev/google/progan-128/1")
# 생성할 샘플의 잠재 공간 차원
latent_dim = 512

# 시드를 바꾸면 다른 얼굴을 생성합니다.
latent_vector = tf.random.normal([1, latent_dim], seed=210810)

# 모듈을 사용해 잠재 공간에서 이미지를 생성합니다.
interpolated_images = module(latent_vector)

plt.imshow(interpolated_images.numpy().reshape(128,128,3))
plt.show()
```

- GAN in Action 교재 6장 깃헙 https://colab.research.google.com/github/rickiepark/gans-in-action/blob/master/chapter-6/Chapter_6_ProGAN.ipynb