# Chap 17.9 생성적 적대 신경망

목표: 판별자를 속이기

> GAN은 다른 목표를 가진 두 네트워크로 구성되므로, 훈련 반복이 두 단계로 이뤄짐

## 1단계: 판별자 훈련

- 훈련 세트에서 실제 이미지 배치를 샘플링
- 생성자에서 생성한 동일한 수의 가짜 이미지를 합침
- 가짜 이미지의 레이블은 0, 진짜 이미지는 1로 세팅
- 판별자는 이진 크로스 엔트로피를 사용해 한 스텝 동안 위처럼 레이블된 배치로 훈련
- 역전파는 판별자의 가중치만 최적화

## 2단계: 생성자 훈련

- 생성자를 사용해 다른 가짜 이미지 배치를 만듦
- 생성자가 만든 가짜 데이터를 판별자가 실제 데이터라고 추정할 확률을 최대화하도록 학습
- 생성자는 진짜 이미지를 보지 않고, 판별자의 반응만 보고 학습

## 패션 MNIST 데이터셋으로 GAN 학습하기

: 생성자와 판별자 생성

```
codings_size = 30

generator = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape=[codings_size]),
    keras.layers.Dense(150, activation="selu"),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
discriminator = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(150, activation="selu"),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(1, activation="sigmoid")
])
gan = keras.models.Sequential([generator, discriminator])
```

- 생성자는 오토인코더의 디코더와 비슷한 모습
- 판별자는 일반적인 이진 분류기
- 훈련 반복의 두 번째 단계에서 생성자와 판별자가 연결된 전체 GAN 모델 필요

## 패션 MNIST 데이터셋으로 GAN 학습하기

: GAN 모델 컴파일

```
discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")
discriminator.trainable = False
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")
```

- 판별자는 이진 분류기 -> 이진 크로스 엔트로피 손실 사용
- 생성자는 GAN 모델을 통해서만 훈련됨 -> 따로 컴파일 할 필요X
- GAN 모델도 이진 분류기 -> 이진 크로스 엔트로피 손실 사용

: 이미지를 순회하는 Dataset 생성

```
batch_size = 32
dataset = tf.data.Dataset.from_tensor_slices(X_train).shuffle(1000)
dataset = dataset.batch(batch_size, drop_remainder=True).prefetch(1)
```

## 패션 MNIST 데이터셋으로 GAN 학습하기

: 훈련 반복 코드 구현

```
def train_gan(gan, dataset, batch_size, codings_size, n_epochs=50):
    generator, discriminator = gan.layers
    for epoch in range(n_epochs):
        print("Epoch {}/{}".format(epoch + 1, n_epochs))              # not sho
        for X_batch in dataset:
            # phase 1 - training the discriminator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            generated_images = generator(noise)
            X_fake_and_real = tf.concat([generated_images, X_batch], axis=0)
            y1 = tf.constant([[0.]] * batch_size + [[1.]] * batch_size)
            discriminator.trainable = True
            discriminator.train_on_batch(X_fake_and_real, y1)
            # phase 2 - training the generator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            y2 = tf.constant([[1.]] * batch_size)
            discriminator.trainable = False
            gan.train_on_batch(noise, y2)
        plot_multiple_images(generated_images, 8)                     # not sho
        plt.show()                                                    # not sho
```

1단계

- 가우시안 잡음을 생성자에 주입해 가짜 이미지 생성
- 생성한 이미지와 동일한 개수의 진짜 이미지를 합쳐 배치 구성
- 타깃 y1은 가짜 이미지일 경우 0, 진짜 이미지일 경우 1로 설정
- 이 배치에서 판별자를 훈련

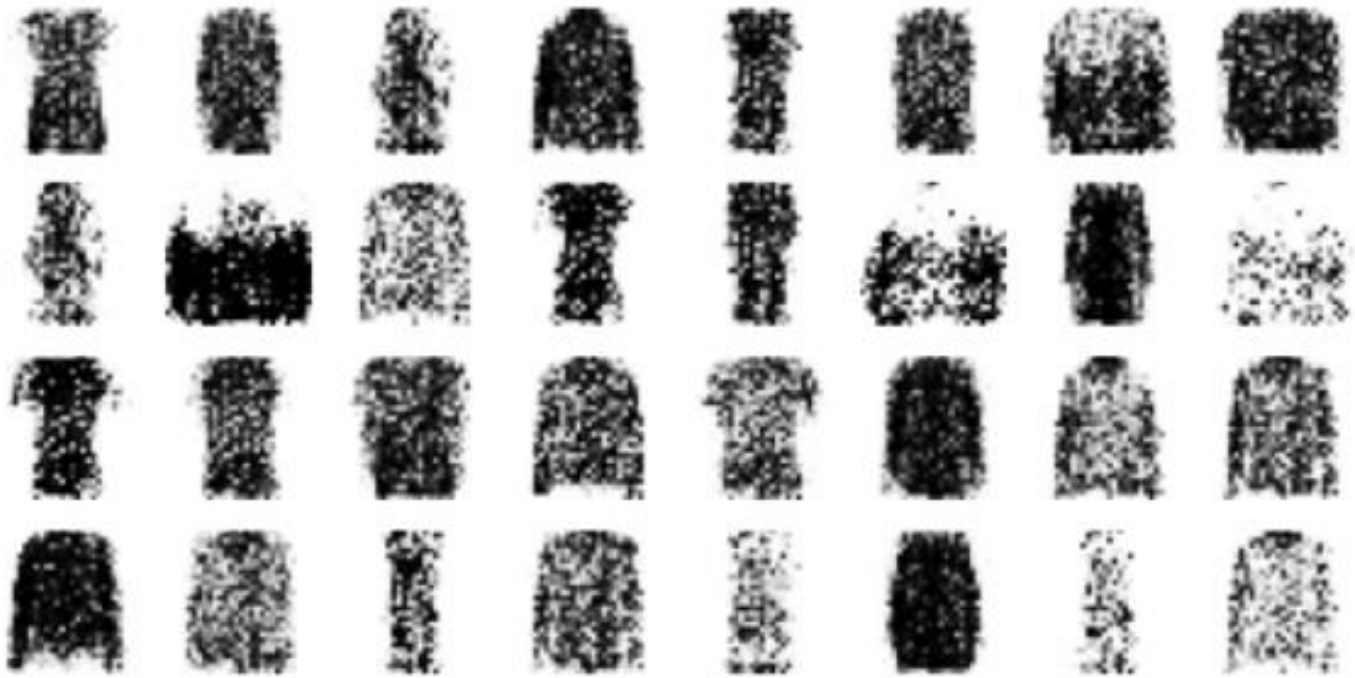
2단계

- GAN에 가우시안 잡음 주입
- 생성자가 먼저 가짜 이미지 생성, 판별자가 추측
- 판별자가 가짜 이미지를 진짜로 믿게 하고 싶으므로 타깃 y2를 1로 지정

## 패션 MNIST 데이터셋으로 GAN 학습하기

: 생성된 이미지 출력

```
train_gan(gan, dataset, batch_size, codings_size, n_epochs=1)
```



# GAN 훈련의 어려움

훈련 과정에서 생성자와 판별자는 서로 앞서려고 끊임없이 노력

제로섬 게임

여러 사람이 서로 영향을 받는 상황에서 모든 이득의 총합이 항상 제로 또는 그 상태

훈련이 내시 균형에 도달 가능

# GAN 훈련의 어려움

## 내시 균형

- **다른 플레이어가 전략을 수정하지 않을 것이므로, 어떤 플레이어도 자신의 전략을 수정하지 않는 상태**

  Ex) 모든 사람이 도로 왼쪽으로 운전할 때, 어떤 운전자도 반대 방향으로 운전하는 것이 도움되지 않는 상태

- **내시 균형 하나는 경쟁 전략도 포괄 가능**

  Ex) 포식자는 먹이를 쫓고, 먹이감은 도망침

# GAN 훈련의 어려움

✓ **GAN은 하나의 내시 균형에만 도달 가능**

WHEN?

- 생성자가 완벽하게 실제와 같은 이미지를 만들어 내, 판별자가 추측밖에 할 수 없을 때

  ➡ 판별자가 50% 확률로 정답을 맞추는 상태를 유지하게 될 때

- GAN을 충분히 훈련하면 완벽한 생성자를 만들어 균형에 도달 가능하지만, 보장X

  ➡ 가장 큰 어려움: 모드 붕괴

# GAN 훈련의 어려움

## 어려움 1. 모드 붕괴

: 생성자의 출력의 다양성이 줄어들 때 발생

➡ GAN이 몇 개의 클래스를 오가다가, 어떤 클래스에서도 좋은 결과를 못 만들 수 있음

## 어려움 2. 파라미터의 변동

: 생성자와 판별자가 지속적으로 서로에게 영향을 줌 -> 파라미터 변동이 크고, 불안정해질 수 있음

(훈련이 안정적으로 시작돼도 이유 없이 갑자기 발산 가능

➡ GAN의 하이퍼파라미터는 매우 민감, 튜닝에 많은 노력 필요

## 안정적인 훈련을 위해서는?

### ✓ 경험 재생

: 매 반복에서 생성자가 만든 이미지를 재생 버퍼에 저장하고, 실제 이미지와 이 버퍼에서 뽑은 가짜 이미지를 더해 판별자를 훈련

➡ 판별자가 생성자의 가장 최근 출력에 과대적합 될 가능성을 줄임

### ✓ 미니배치 판별

: 배치 간에 얼마나 비슷한 이미지가 있는 지 측정해, 이 통계를 판별자에게 제공

: 판별자는 다양성이 부족한 가짜 이미지 배치 전체를 쉽게 거부 가능

➡ 생성자가 다양한 이미지를 생성하도록 유도해 모드 붕괴의 위험 줄임

# 심층 합성곱 GAN

2014년 원본 GAN 논문에서는 합성곱 층을 통해 작은 이미지만 생성

이후, 큰 이미지를 위해 깊은 합성곱 층 기반의 GAN을 만들기 위해 노력

→ 심층 합성곱 GAN(DCGAN) 제안

**안정적인 합성곱 GAN을 구축하기 위한 가이드라인**

* 항상 맞는 것은 X -> 여러 하이퍼파라미터로 실험 필요

- 판별자의 풀링 층을 스트라이드 합성곱으로 변경
- 생성자의 풀링 층은 전치 합성곱으로 변경
- 생성자와 판별자에 배치 정규화를 사용
- 깊은 층을 위해 완전 연결 은닉층을 제거
- 생성자의 모든 층은 ReLU 활성화 함수 사용
- 판별자의 모든 층은 LeakyReLU 활성화 함수 사용

## 패션 MNIST DCGAN 모델

```
codings_size = 100

generator = keras.models.Sequential([
    keras.layers.Dense(7 * 7 * 128, input_shape=[codings_size]),
    keras.layers.Reshape([7, 7, 128]),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(64, kernel_size=5, strides=2, padding="SAME",
                                 activation="selu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(1, kernel_size=5, strides=2, padding="SAME",
                                 activation="tanh"),
])
discriminator = keras.models.Sequential([
    keras.layers.Conv2D(64, kernel_size=5, strides=2, padding="SAME",
                        activation=keras.layers.LeakyReLU(0.2),
                        input_shape=[28, 28, 1]),
    keras.layers.Dropout(0.4),
    keras.layers.Conv2D(128, kernel_size=5, strides=2, padding="SAME",
                        activation=keras.layers.LeakyReLU(0.2)),
    keras.layers.Dropout(0.4),
    keras.layers.Flatten(),
    keras.layers.Dense(1, activation="sigmoid")
])
gan = keras.models.Sequential([generator, discriminator])
```

### 생성자

- 크기는 100의 코딩을 받아 6272차원으로 투영하고, 이 결과를 7X7X128 크기의 텐서로 변형
- 이 텐서는 배치 정규화를 거쳐 스트라이드가 2인 전치 합성곱 층에 주입
- 다시 배치 정규화를 거쳐 스트라이드가 2인 전치 합성곱 층에 주입

# 심층 합성곱 GAN

## 패션 MNIST DCGAN 모델

```
X_train_dcgan = X_train.reshape(-1, 28, 28, 1) * 2. - 1. # reshape and rescale
```

### 판별자

- 이진 분류를 위한 일반적인 CNN과 비슷한 구조,
- BUT 이미지를 다운 샘플링할 때 MaxPooling이 아닌 스트라이드 합성곱 사용

### DCGAN 가이드라인과의 차이점

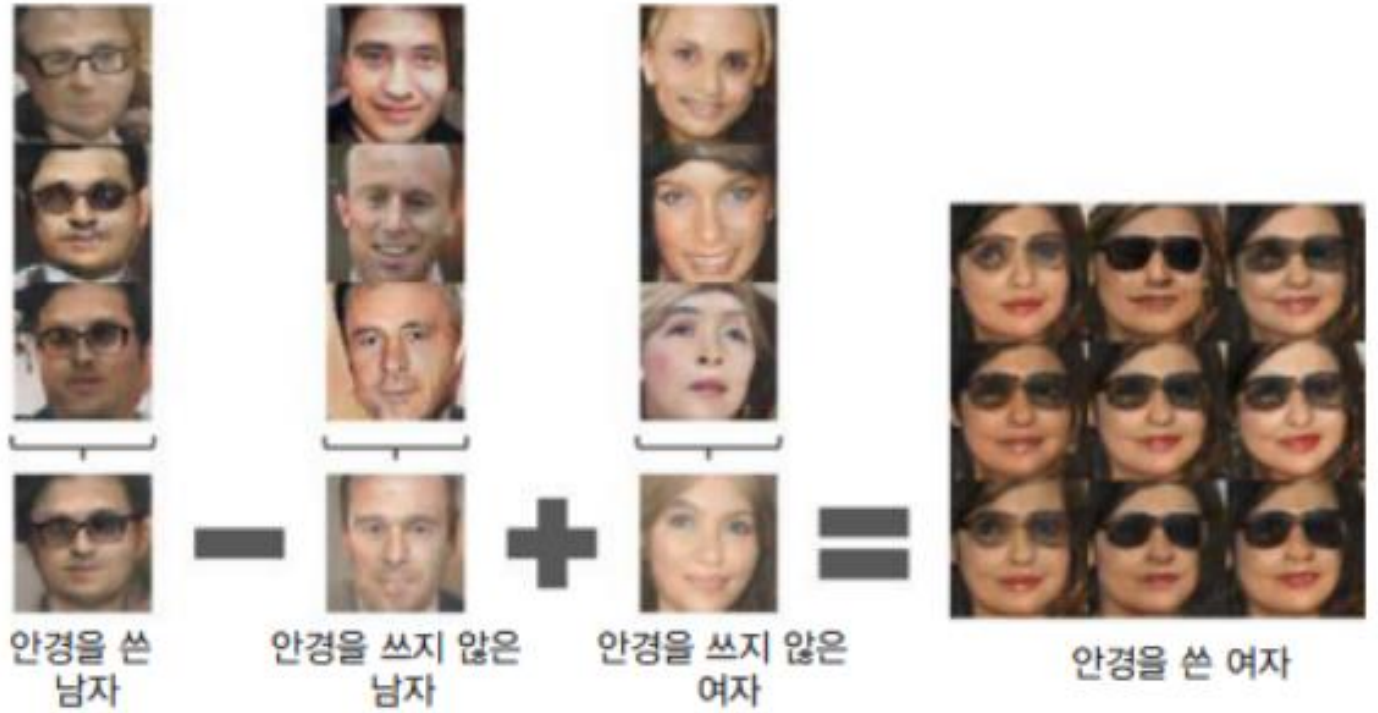
- 훈련의 안정성을 위해 판별자의 BatchNormalization 층 -> Dropout 층으로 변경
- 생성자에서 ReLU가 아닌 SELU 사용

* 자유롭게 구조를 바꿔보면서 하이퍼파라미터에 대한 민감성 확인 가능

## 훈련 결과

# 심층 합성곱 GAN

## DCGAN의 잠재 표현 학습




* DCGAN은 완벽하지 않기 때문에, 이를 가지고 매우 큰 이미지를 생성하면 일관성이 없는 이미지를 얻을 가능성이 높음

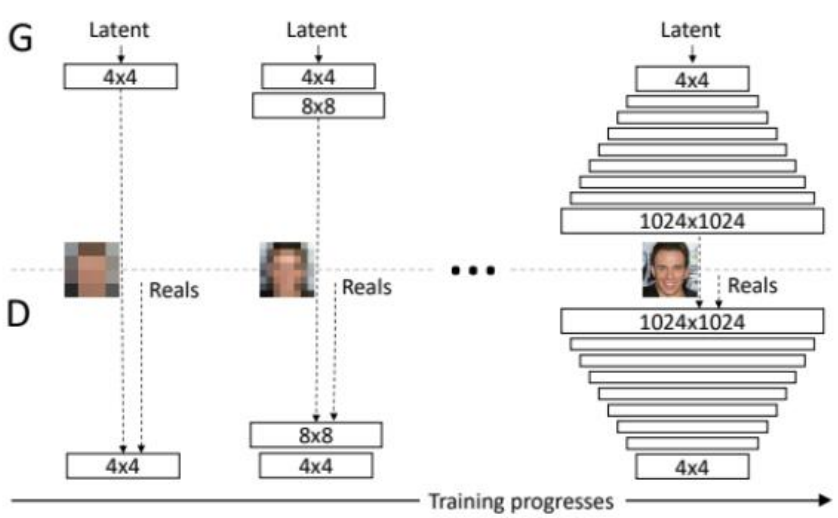
# ProGAN

훈련 초기에 작은 이미지를 생성하고, 점진적으로 생성자와 판별자에 합성곱을 추가해, 갈 수록 큰 이미지를 만드는 방법

-> 적층 오토인코더를 층별로 훈련하는 것과 유사

-> 이전에 훈련된 층은 그대로 두고, 생성자의 끝과 판별자의 시작 부분에 층을 추가하며 쌓아감

## ProGAN의 구조

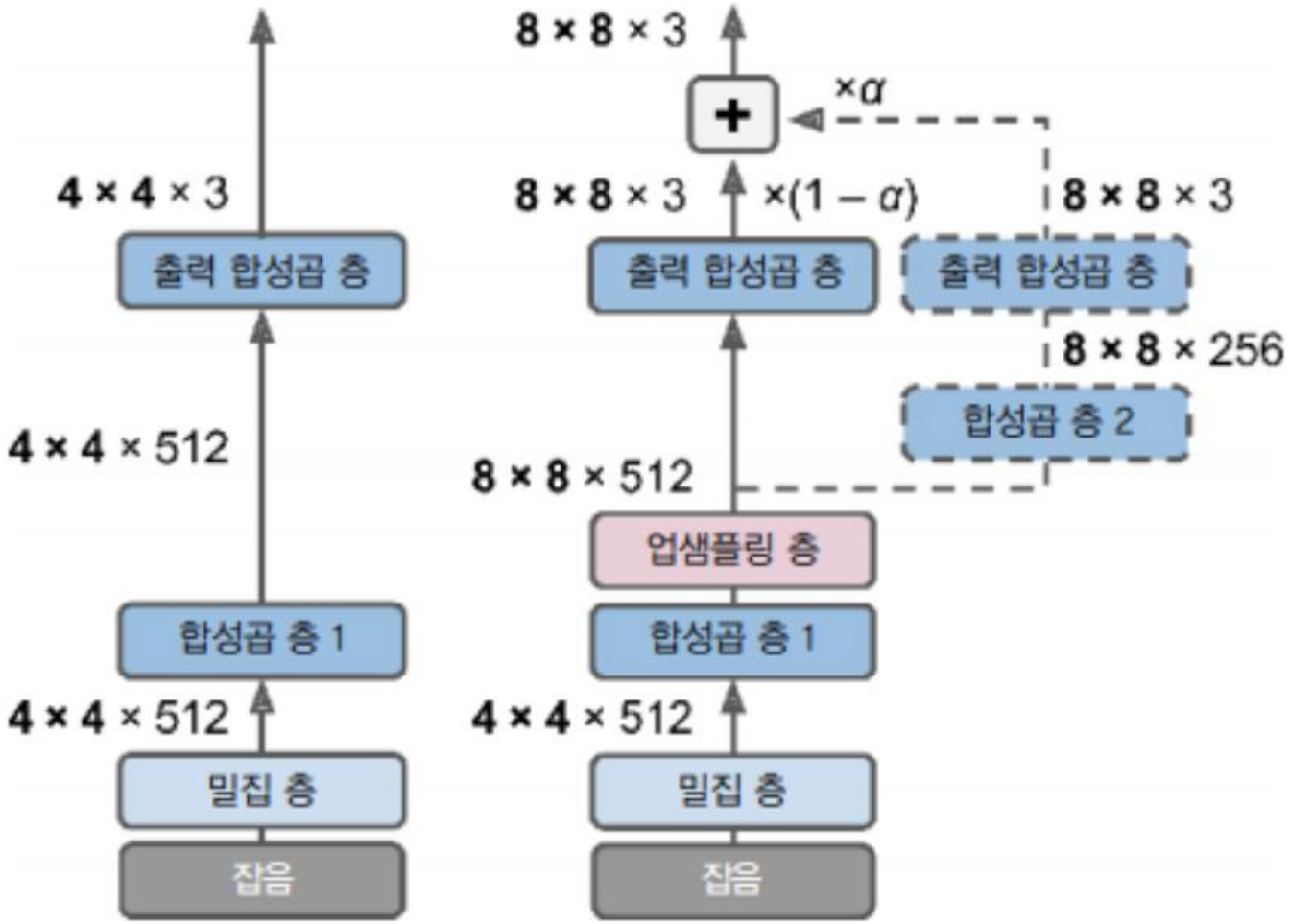



한번에 전체 크기의 이미지 특성을 학습시키기보다,

4X4 저해상도로 large-scale structur를 찾아내도록,

점차 finer-scale detail을 찾을 수 있도록

1024x1024 고해상도로 높아지는 것이 나음



## 미니배치 표준편차 층

- 판별자의 마지막 층 근처에 추가
- 입력에 있는 모든 위치에 대해 모든 채널과 배치의 모든 샘플에 걸쳐 표준편차를 계산함

```
S = tf.math.reduce_std(inputs, axis=[0, -1])
이 표준편차는 모든 픽셀에 대해 평균해 하나의 값을 얻음
v = tf.reduce_mean(S)
```

- 추가적인 특성 맵이 배치의 모든 샘플에 추가되고, 계산된 이 값으로 채워짐

```
tf.concat([inputs, tf.fill([batch_size, height, width, 1], v)], axis=-1)
```

생성자가 만든 이미지에 다양성이 부족하면, 판별자의 특성 맵 간의 표준편차가 작을 것

판별자는 이 통계를 통해 다양성이 적은 이미지를 만드는 생성자에게 속을 가능성이 줄어듦

➡ 생성자가 다양한 출력을 만들도록 유도해 모드 붕괴의 위험성을 줄임

## 동일한 학습 속도

- He 초기화 대신, 평균이 0이고 표준편차가 1인 가우시안 분포를 사용해 모든 가중치를 초기화
- But, 런타임에 He 초기화를 통해 가중치 스케일을 낮춤

  가중치를 √(2/ninputs)로 나눔

- RMSProp, Adam 같은 적응적 그래디언트 옵티마이저 사용 -> 이 기법이 GAN의 성능을 더 높임

  각자 추정한 표준편차로 그래디언트 업데이트를 정규화함

  - 다이나믹 레인지가 큰 파라미터는 훈련 시간이 오래 걸림
  - 다이나믹 레인지가 작은 파라미터는 너무 빠르게 업데이트 되어 불안정해질 수 있음

- 가중치 초기화에서 스케일을 맞추지 않고, 모델의 한 부분으로 가중치를 조절해 훈련 내내 모든 파라미터의 다이나믹 레인지를 동일하게 만듦

  ➡ 모든 가중치가 동일한 속도로 학습되어, 훈련 속도와 안정성이 높아짐

### 픽셀별 정규화 층

- 생성자의 합성곱 층 뒤에 추가
- 픽셀 단위 별로 정규화해주는 것으로, 동일한 이미지의 동일 위치에 있는 모든 활성화를 채널에 대해 정규화함

> 활성화의 제곱 평균을 제곱근으로 나눔
>
> inputs/tf.sqrt(tf.reduce_mean(tf.square(X), axis=-1, keepdims=True)+1e-8

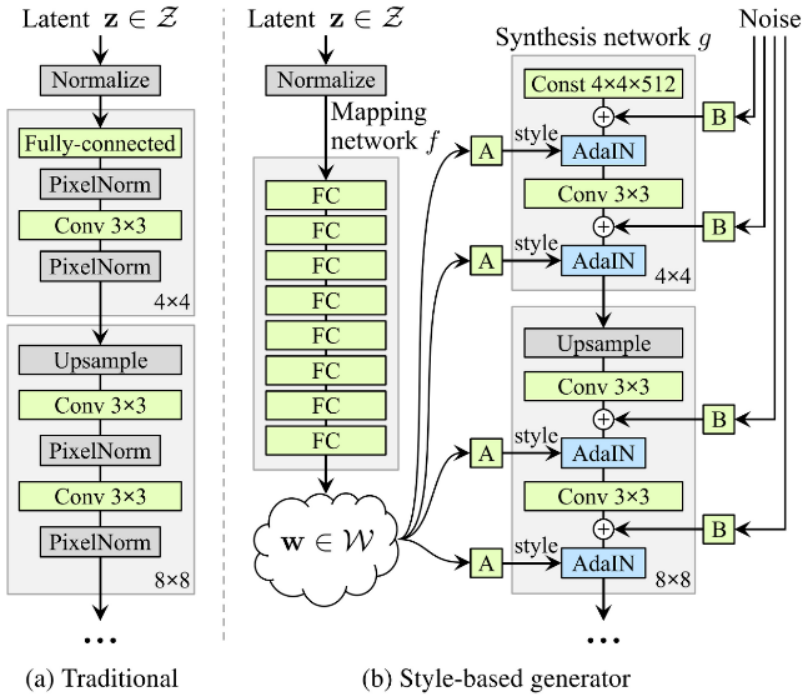
생성자와 판별자 사이의 과도한 경쟁으로 활성화 값이 폭주되는 것을 막음

* 일반적으로 GAN은 생성자와 판별자의 불필요한 경쟁으로 활성화 값이 폭주되는 것을 억제하기 위해 배치 정규화 사용
  BUT, PGGAN은 이 방법이 효과 X -> 픽셀 정규화 방식 사용

생성자에 Style Transfer 기법을 사용해, 생성된 이미지가 훈련된 이미지와 같은 다양한 크기의 국부적인 구조를 갖도록 만듦

-> 생성된 이미지의 품질을 크게 높여줌

## StyleGAN의 구조




(a) Traditional (b) Style-based generator

- 판별자와 손실한 수는 그대로, 생성자만 변경됨
- 두 가지 네트워크로 구성됨: 매핑 네트워크, 합성 네트워크

## 매핑 네트워크

- StyleGAN은 잠재 표현 z(코딩)으로부터 직접 이미지를 생성하지 않고, 매핑 네트워크를 거침
- 8개의 MLP가 z를 벡터 w로 매핑

  여러 아핀 변환으로 전달되어 벡터 여러 개를 생성

  미세한 텍스처부터 고수준 특성까지 각기 다른 수준에서 생성된 이미지의 스타일을 제어

➡ 매핑 네트워크는 코딩을 여러 스타일 벡터로 매핑하는 것

## 합성 네트워크

- 이미지의 생성을 책임짐
- 이 네트워크는 일정하게 학습된 입력을 받음 -> 입력을 합성곱 여러 개와 업샘플링 층에 통과시킴
- 차이점

  1) 입력과 모든 합성곱 층의 출력에 잡음이 섞임

  2) 잡음이 섞인 다음에 적응적 인스턴스 정규화(AdaIN) 층이 뒤 따름

## 코딩에 독립적으로 잡음을 추가해야하는 이유

- 이미지의 랜덤한 어떤 부분, 무작위성이 초기 GAN에서는 코딩이나 생성자 자체에서 만든 랜덤한 잡음에서 옴

-> 생성자가 코딩의 표현 능력을 잡음을 저장하는데 할애한다는 의미

-> 이 잡음이 네트워크를 흘러 생성자의 마지막 층에 도달해야 함

훈련 속도를 느리게하는 제약 사항이 될 수 있음

-> 각기 다른 수준에서 동일한 잡음이 사용 됨 -> 일부 인공적인 요소가 나타날 수 있음

=> 별도의 잡음 추가시 이런 이슈 해결 가능

=> 추가된 잡음을 사용해 이미지의 각 부분에 정확한 양의 무작위성 추가 가능

## 믹싱 규제(스타일 믹싱)

: 일정 비율의 이미지를 두 개의 다른 코딩으로 생성하는 StyleGAN의 기법

- 코딩 c1과 c2가 매핑 네트워크를 통과해 두 스타일 벡터 w1, w2 생성
- 이후, 합성 네트워크가 첫 번째 단계에서 스타일 w1으로 나머지 단계에서는 스타일 w2를 바탕으로 이미지 생성

  * 변경되는 단계는 랜덤하게 선택 -> 네트워크가 연속된 두 layer 간의 스타일이 상관관계를 가진다고 가정하지 못하게 함

=> 각 스타일 벡터가 생성된 이미지에 있는 제한된 개수의 속성에만 영향을 미치는 StyleGAN의 국지성을 촉진시킴