

# Lezione 11 - Generative adversarial networks - 06/12/2024

## Generative adversarial networks

Questo è un paradigma diverso rispetto a quello che abbiamo visto fin ora, non vogliamo classificare, ma vogliamo generare dati nuovi che si assomigli ai nostri dati.

**Generative adversarial networks** è stata la tecnica più usata, fino a pochi anni fa (finché gli approcci di stable diffusion sono arrivati).

L'esempio del 2017 usava un dataset di cani, ci sono delle gambe, delle forme... ma non c'è la generazione di un cane. Solo un anno dopo, la tecnica è migliorata molto, però ci sono delle ripetizioni di pattern (nella seconda per esempio), o la farfalla non è formata molto bene, però la qualità è buona.

[EBGAN](#) (2017)



[BigGAN](#) (2018)



# Generative tasks

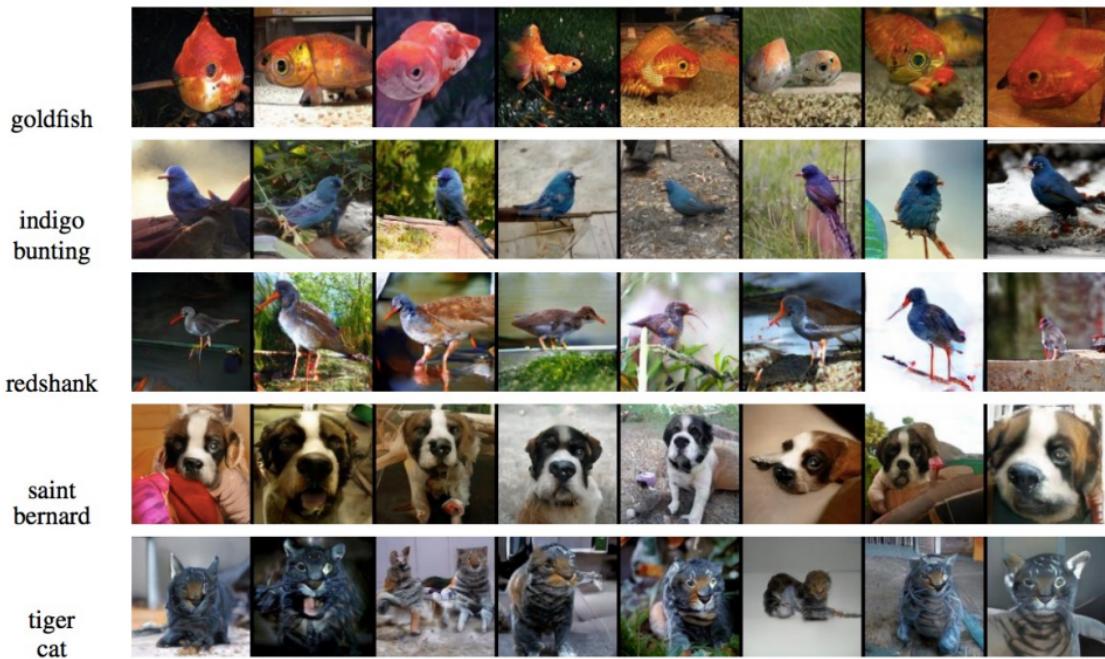
Vogliamo che la task di **generazione**, generi qualcosa from-scratch, cioè dal nulla.

In particolare, vogliamo imparare a **campionare dalla distribuzione rappresentata dal training set**.

Se il training set è un'insieme di immagini di camere da letto, allora questo è quello che voglio generare, questa è la distribuzione che voglio rappresentare.

In particolare questa è una task **unsupervised**, non utilizziamo labels.

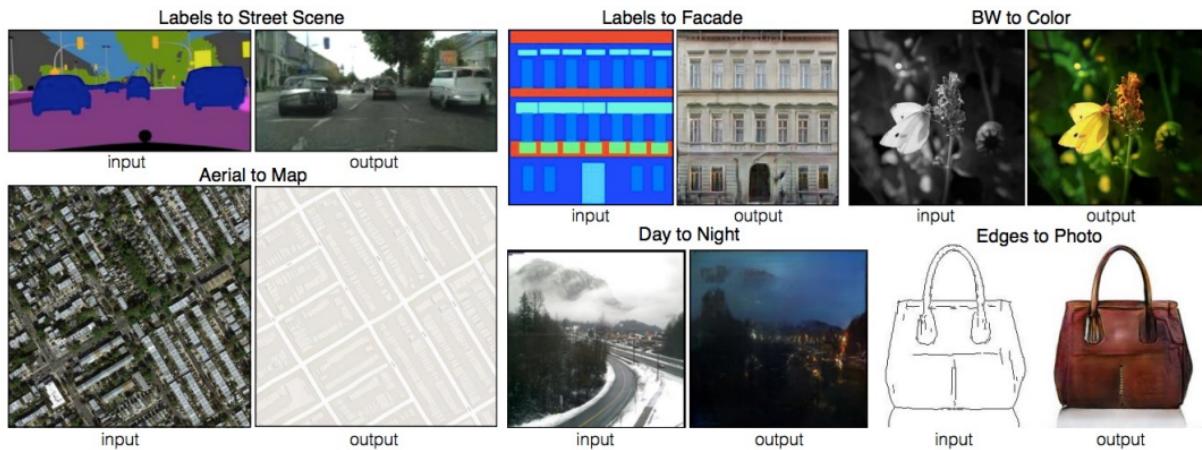
A volte vorremmo controllare il nostro output, si chiama **conditional generation** perchè vorrei per esempio creare una nuova istanza di un cane specifico. Questa è ancora una task unsupervised, perchè non abbiamo la ground truth, aggiungiamo solo una proprietà, è una conditional generation.



L'**image-to-image translation** invece è quando abbiamo un'immagine con delle caratteristiche, e vogliamo generare un'altra immagine con lo stesso contenuto ma caratteristiche diverse.

Per esempio ho un'immagine in bianco e nero, e voglio generare la versione colorata.

Oppure ho un'immagine fatta durante il giorno, e voglio trasformarla come se fosse stata scattata in notturna.



L'esempio in alto a sinistra più essere usato per **data augmentation**. Possiamo generare dati, specificando dove si trovano gli oggetti di interesse.

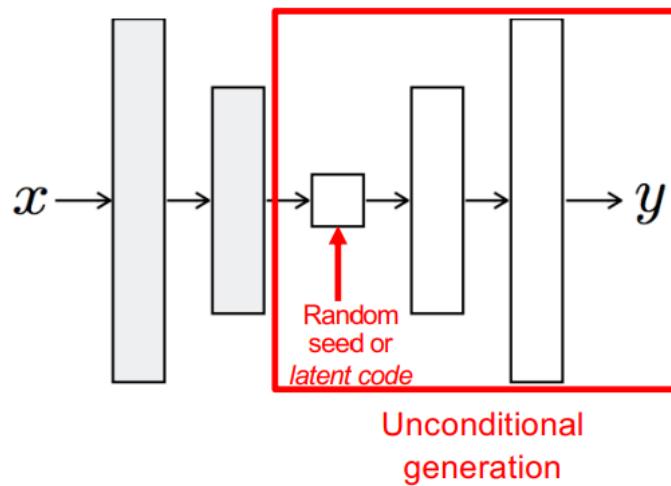
Nell'esempio day to night, si può usare questa generazione per generare lo stesso contenuto ma per esempio sotto condizioni di meteo diverse, per avere una vista chiara di cosa sta succedendo.

## Designing a network for generative tasks

Quindi come possiamo creare una rete che performa una task generativa?

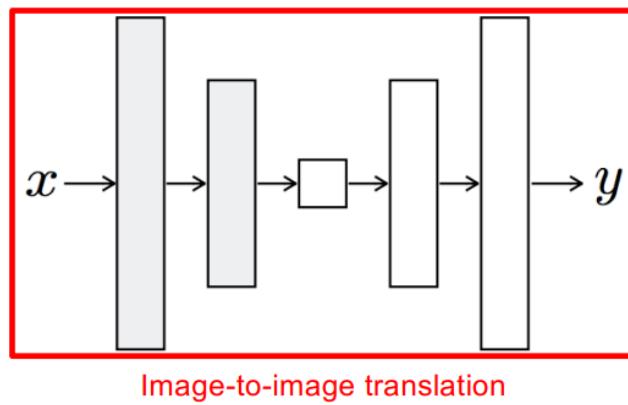
Fin ora abbiamo visto architetture che processano le immagini per ridurre la dimensione spaziale, per arrivare ad una feature che può essere usata per classificare. Ma ora invece vogliamo partire da una cosa piccola, e ingrandirla finché arriviamo all'immagine finale.

Abbiamo già visto qualcosa di simile quando abbiamo lavorato sugli autoencoders. La prima parte ha l'input, otteniamo una rappresentazione sempre più piccola, e poi da quella più piccola abbiamo la parte del **decoder**, che partendo da un'informazione piccola, la aumenta fino a generare l'immagine finale, cercando di ricreare l'input.



In particolare, vorremmo non avere questa prima parte (encoder) perché non abbiamo un'immagine da cui partire. Vogliamo generare qualcosa, partendo da noise random.

Questo tipo di architettura è usata per **image-to-image translation**, e si chiama U-NET (penso?), ha anche delle skip connections tra l'encoder e il decoder.



Quindi però non possiamo usare le **loss** che abbiamo visto fin ora, perchè hanno l'assunzione che abbiamo i dati di input (e/o la label), di modo da confrontare la label reale con quella predetta. In questo caso invece abbiamo random noise in input, quindi non abbiamo nulla con cui fare una comparazione.

Negli autoencoders era tutto semplice, avevamo fully connected layers, la strozzatura con la latent representation...

Se usiamo immagini, o comunque dati che sono su un grid, la convoluzione funziona meglio. Nella convoluzione però abbiamo visto che riduciamo la dimensione spaziale, non la aumentiamo mai. Quindi questa può essere usata per la parte di encoder, ma non per il decoder, dobbiamo trovare un modo.

Ora andiamo a rivisitare la convoluzione in un modo diverso, per introdurre un nuovo tipo di convoluzione che può aumentare gli spatial data.

10:29 Ven 6 Dic 2024

Lezione 12

FILTER  $\downarrow$  STRIDE=1

$X_{11} \quad X_{12} \quad X_{13} \quad | \quad X_{14}$

$X_{21} \quad X_{22} \quad X_{23} \quad | \quad X_{24}$

$X_{31} \quad X_{32} \quad X_{33} \quad | \quad X_{34}$

$X_{41} \quad X_{42} \quad X_{43} \quad X_{44}$

$W_{11} \quad W_{12} \quad W_{13}$

$* \quad W_{21} \quad W_{22} \quad W_{23}$

$W_{31} \quad W_{32} \quad W_{33}$

$= \quad Z_{11} \quad Z_{12}$

$Z_{11} \quad Z_{21}$

OUTPUT  $2 \times 2$

FILTER  $3 \times 3 (x1)$

INPUT  $4 \times 4 (x1)$

$Z_{11} = X_{11} \cdot w_{11} + X_{12} \cdot w_{12} + X_{13} \cdot w_{13} +$

$X_{21} \cdot w_{21} + X_{22} \cdot w_{22} + X_{23} \cdot w_{23} +$

$X_{31} \cdot w_{31} + X_{32} \cdot w_{32} + X_{33} \cdot w_{33}$

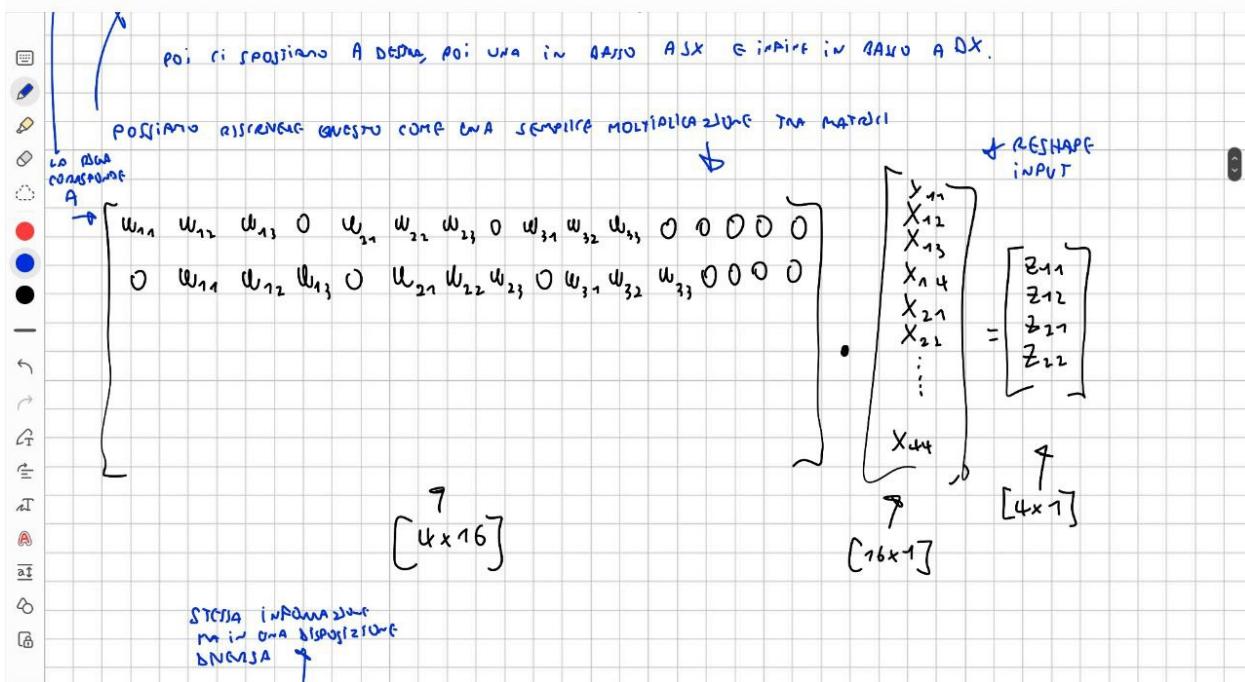
poi si spostano a destra, poi una in alto ASX è spostato in alto a dx.

POSSIAMO RISCRIVERE QUESTO COME UNA SEMPRE MOLTIPLICAZIONE TRA MATRICI

RESHAPE INPUT

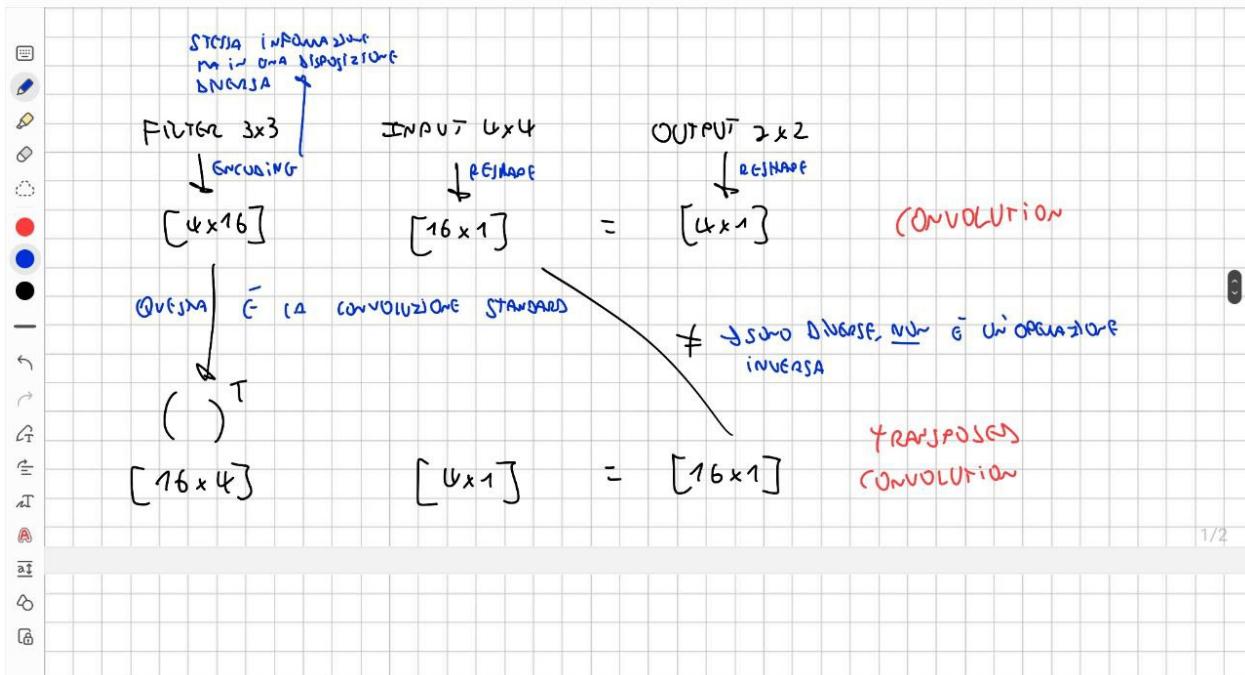
## Lezione 12

◀ □ ⌂ ⌃ :



## Lezione 12

◀ □ ⌂ ⌃ ⌄ :



Stesse cose sulle slides:

- Regular convolution (stride 1, pad 0)

$$\begin{array}{|c|c|c|c|} \hline x_{11} & x_{12} & x_{13} & x_{14} \\ \hline x_{21} & x_{22} & x_{23} & x_{24} \\ \hline x_{31} & x_{32} & x_{33} & x_{34} \\ \hline x_{41} & x_{42} & x_{43} & x_{44} \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline w_{11} & w_{12} & w_{13} \\ \hline w_{21} & w_{22} & w_{23} \\ \hline w_{31} & w_{32} & w_{33} \\ \hline w_{41} & w_{42} & w_{43} \\ \hline \end{array} = \begin{array}{|c|c|} \hline z_{11} & z_{12} \\ \hline z_{21} & z_{22} \\ \hline \end{array}$$

- Matrix-vector form:

$$\begin{pmatrix} w_{11} & w_{12} & w_{13} & 0 & w_{21} & w_{22} & w_{23} & 0 & w_{31} & w_{32} & w_{33} & 0 & 0 & 0 & 0 \\ 0 & w_{11} & w_{12} & w_{13} & 0 & w_{21} & w_{22} & w_{23} & 0 & w_{31} & w_{32} & w_{33} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{11} & w_{12} & w_{13} & 0 & w_{21} & w_{22} & w_{23} & 0 & w_{31} & w_{32} & w_{33} \\ 0 & 0 & 0 & 0 & 0 & w_{11} & w_{12} & w_{13} & 0 & w_{21} & w_{22} & w_{23} & 0 & w_{31} & w_{32} & w_{33} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ \vdots \\ x_{44} \end{pmatrix} = \begin{pmatrix} z_{11} \\ z_{12} \\ z_{21} \\ z_{22} \end{pmatrix}$$

4x4 input, 2x2 output

- Transposed convolution

$$\begin{array}{|c|c|} \hline z_{11} & z_{12} \\ \hline z_{21} & z_{22} \\ \hline \end{array} *^T \begin{array}{|c|c|c|} \hline w_{11} & w_{12} & w_{13} \\ \hline w_{21} & w_{22} & w_{23} \\ \hline w_{31} & w_{32} & w_{33} \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline x_{11} & x_{12} & x_{13} & x_{14} \\ \hline x_{21} & x_{22} & x_{23} & x_{24} \\ \hline x_{31} & x_{32} & x_{33} & x_{34} \\ \hline x_{41} & x_{42} & x_{43} & x_{44} \\ \hline \end{array}$$

$$\begin{pmatrix} w_{11} & 0 & 0 & 0 \\ w_{12} & w_{11} & 0 & 0 \\ w_{13} & w_{12} & 0 & 0 \\ 0 & w_{13} & 0 & 0 \\ w_{21} & 0 & w_{11} & 0 \\ w_{22} & w_{21} & w_{12} & w_{11} \\ w_{23} & w_{22} & w_{13} & w_{12} \\ 0 & w_{23} & 0 & w_{13} \\ w_{31} & 0 & w_{21} & 0 \\ w_{32} & w_{31} & w_{22} & w_{21} \\ w_{33} & w_{32} & w_{23} & w_{22} \\ 0 & w_{33} & 0 & w_{23} \\ 0 & 0 & w_{31} & 0 \\ 0 & 0 & w_{32} & w_{31} \\ 0 & 0 & w_{33} & w_{32} \\ 0 & 0 & 0 & w_{33} \end{pmatrix} = \begin{pmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \\ x_{31} \\ x_{32} \\ x_{33} \\ x_{34} \\ x_{41} \\ x_{42} \\ x_{43} \\ x_{44} \end{pmatrix}$$

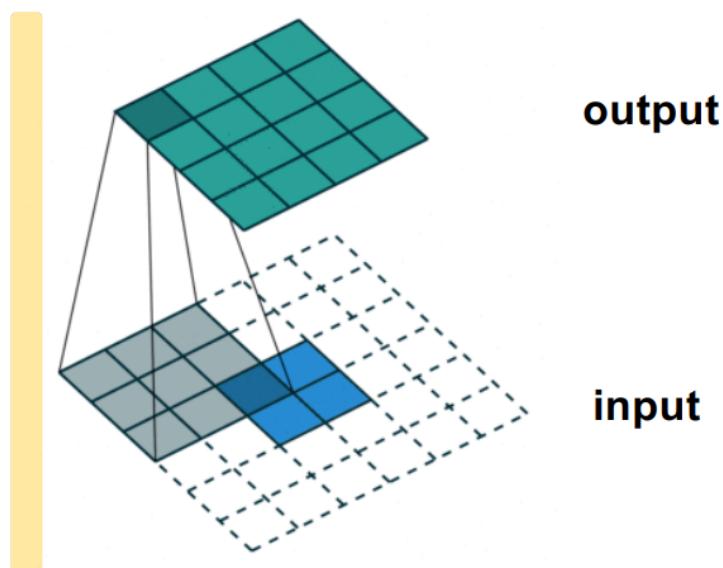
2x2 input, 4x4 output

*Not an inverse of the original convolution operation, simply reverses dimension change!*

Questo è il layer più importante. Perchè con la transposed convolution otteniamo un output più grande rispetto all'input. Si chiama trasposta perchè facciamo la trasposta della matrice delle moltiplicazioni.

Se applichiamo la convoluzione normale, e poi sull'output applichiamo la convoluzione trasposta, NON otteniamo l'immagine originale, non è l'operazione inversa.

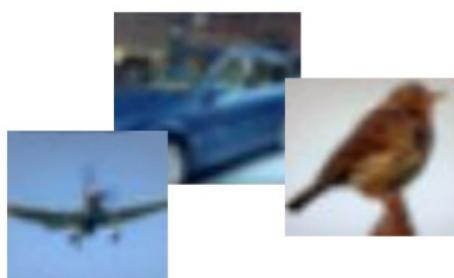
Questo è quello che succede nella **transposed convolution**:



Il  $3 \times 3$  grigio è il filtro.

## Learning to sample

Partiamo dal concetto del **learning to sample**. Vogliamo generare dei nuovi samples, in base alla probabilità del modello, e vogliamo che la probabilità del modello matchi quella dei dati.



Training data  $x \sim p_{\text{data}}$



Generated samples  $x \sim p_{\text{model}}$

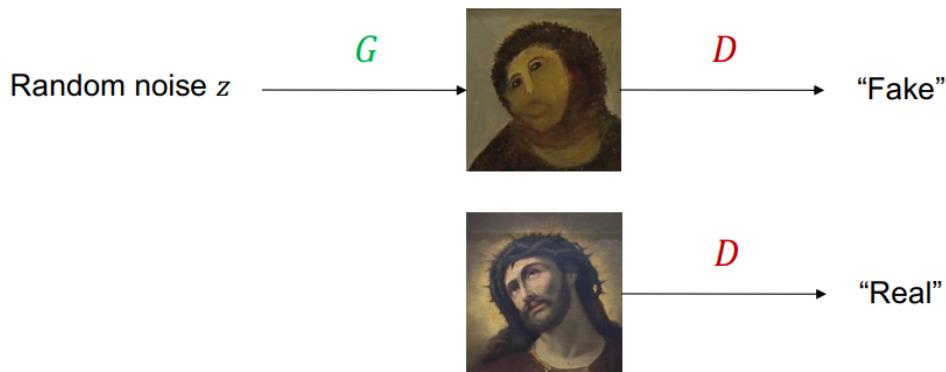
We want to learn  $p_{\text{model}}$  that matches  $p_{\text{data}}$

## Generative adversarial networks (GAN)

In particolare, possiamo vedere che nel titolo abbiamo il plurale su networks.  
Questo è necessario perché per questa task non abbiamo un singolo modello, ma abbiamo **2 reti diverse con 2 task diverse**, che competono tra di loro,

Il **generatore** è la rete che vuole imparare a generare i samples.

Il **discriminatore** vuole imparare a distinguere tra i samples reali e quelli generati.



Il discriminatore farà fatica.

Il discriminatore  $D(x)$  deve avere come output la probabilità che il sample  $x$  sia reale. In particolare, vogliamo che l'output sia vicino a 1 per dati reali, e vicino a 0 per dati generati.

Possiamo calcolare la **expected conditional log likelihood** per dati reali e generati:  
 (la prima parte è per i dati reali, e la seconda per quelli generati)

$$\mathbb{E}_{x \sim p_{\text{data}}} \log D(x) + \mathbb{E}_{x \sim p_{\text{gen}}} \log(1 - D(x)) \\ = \mathbb{E}_{x \sim p_{\text{data}}} \log D(x) + \mathbb{E}_{z \sim p} \log(1 - D(G(z)))$$

We seed the generator with noise  $z$   
 drawn from a simple distribution  $p$   
 (Gaussian or uniform)

Ma visto che la  $x$  dei dati generati è data dalla generazione da random noise, la riscriviamo nel secondo modo.

Quindi questo è il nostro obiettivo:

$$V(G, D) = \mathbb{E}_{x \sim p_{\text{data}}} \log D(x) + \mathbb{E}_{z \sim p} \log(1 - D(G(z)))$$

Abbiamo il discriminatore, che vuole distinguere tra real e fake, vuole massimizzare l'objective.-

$$D^* = \arg \max_D V(G, D)$$

D'altro canto il generatore vuole "fregare" il discriminatore, minimizzando l'objective.

$$G^* = \arg \min_G V(G, D)$$

Quindi li alleniamo insieme, in quello che si chiama **minmax game**.

Informazioni extra sul funzionamento (alternato):

- Assuming unlimited capacity for generator and discriminator and unlimited training data:
  - The objective  $\min_G \max_D V(G, D)$  is equivalent to *Jensen-Shannon divergence* between  $p_{\text{data}}$  and  $p_{\text{gen}}$  and global optimum (*Nash equilibrium*) is given by  $p_{\text{data}} = p_{\text{gen}}$
  - If at each step,  $D$  is allowed to reach its optimum given  $G$ , and  $G$  is updated to decrease  $V(G, D)$ , then  $p_{\text{gen}}$  will eventually converge to  $p_{\text{data}}$

Al posto di alternare, potremmo provare a riscrivere questa loss, di modo che stiamo sempre minimizzando o stiamo sempre massimizzando.

Al posto di minimizzare quello che abbiamo per il generatore, possiamo riscriverlo cambiando il segno dentro al log, di modo che ora siamo interessati a massimizzare.

$$\begin{aligned} G^* &= \arg \min_G V(G, D) \\ &= \arg \min_G \mathbb{E}_{z \sim p} \log(1 - D(G(z))) \end{aligned}$$

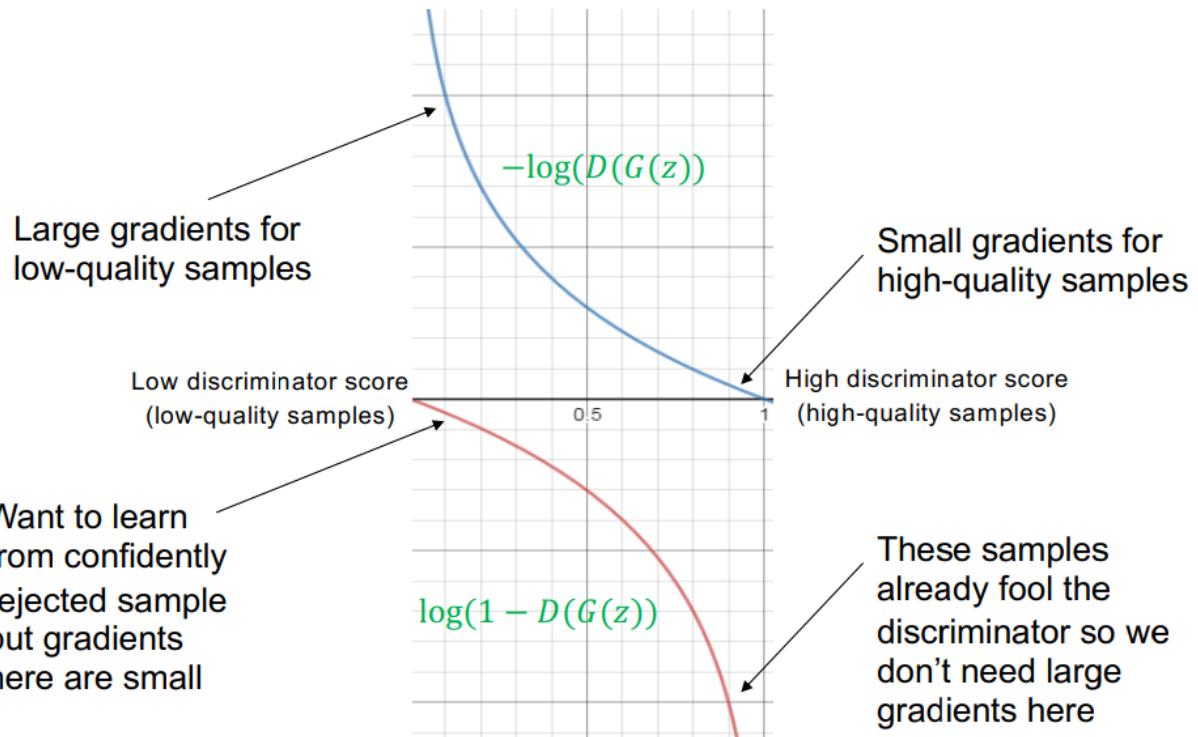
In questo modo, durante l'ottimizzazione, stiamo sempre andando nella stessa direzione.

$$G^* = \arg \max_G \mathbb{E}_{z \sim p} \log(D(G(z)))$$

Questo ha anche un impatto su cosa sta imparando la rete.

Penso che il grafico sopra è quello nuovo mentre quello sotto è quello vecchio.

$$\min_{w_G} \mathbb{E}_{z \sim p} \log(1 - D(G(z))) \text{ vs. } \max_{w_G} \mathbb{E}_{z \sim p} \log(D(G(z)))$$



Quando il discriminatore da uno score molto vicino a 0, sappiamo che praticamente sicuramente non è un sample reale. Mentre invece se siamo vicini all'1, abbiamo samples ad alta qualità e il discriminatore pensa che siano reali.

Vorremmo poter imparare da questi esempi che sono brutti, perché possono avere tanta informazione per migliorare velocemente, specialmente all'inizio del training.

Però qui i gradienti non sono molto grandi (lo slope è basso).

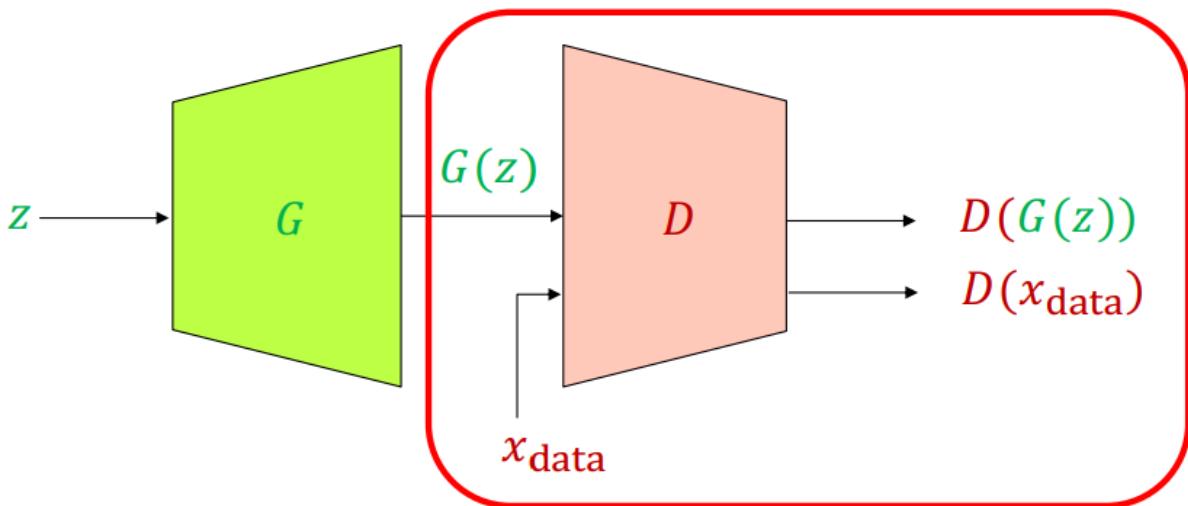
Però lo slope sopra è molto più grande, quindi abbiamo gradienti molto più grandi, che permettono di imparare molto più velocemente all'inizio del training.  
più altre informazioni sull'immagine.

---

Training:

- Update discriminator
  - Repeat for  $k$  steps:
    - Sample mini-batch of noise samples  $z_1, \dots, z_m$  and mini-batch of real samples  $x_1, \dots, x_m$
    - Update parameters of  $D$  by stochastic gradient ascent on
 
$$\frac{1}{m} \sum_m [\log D(x_m) + \log(1 - D(G(z_m)))]$$
  - Update generator
    - Sample mini-batch of noise samples  $z_1, \dots, z_m$
    - Update parameters of  $G$  by stochastic gradient ascent on
 
$$\frac{1}{m} \sum_m \log D(G(z_m))$$
- Repeat until happy with results

Quindi andiamo a fare il processo inverso di una convolutional NN, partiamo da qualcosa a dimensione spaziale bassa, e andiamo ad aumentarla. Poi l'output del generatore va nel discriminatore per calcolare lo score, ovvero la probabilità che sia un sample reale o fake.



Quindi diamo al generatore sia il dato generato, che un dato reale.

- Update discriminator: push  $D(x_{\text{data}})$  close to 1 and  $D(G(z))$  close to 0
  - The generator is a “black box” to the discriminator

*G E N E R A T O R E*

Il discriminatore è un **black box** rispetto al discriminatore perché i gradienti del generatore non danno informazioni al discriminatore perché vengono aggiornati dopo.

Per il generatore, vogliamo aumentare lo score che il generatore dia, quindi vogliamo fregarlo, di modo che il discriminatore dia score alti alle immagini generate.

Quindi vogliamo aumentare  $D(G(z))$  aggiornando G.

*C G E N E R A T O R E*

Quindi il discriminatore non è un black box rispetto al generatore, perchè riceve i gradienti che passano dal discriminatore, prima di updateare il generatore.

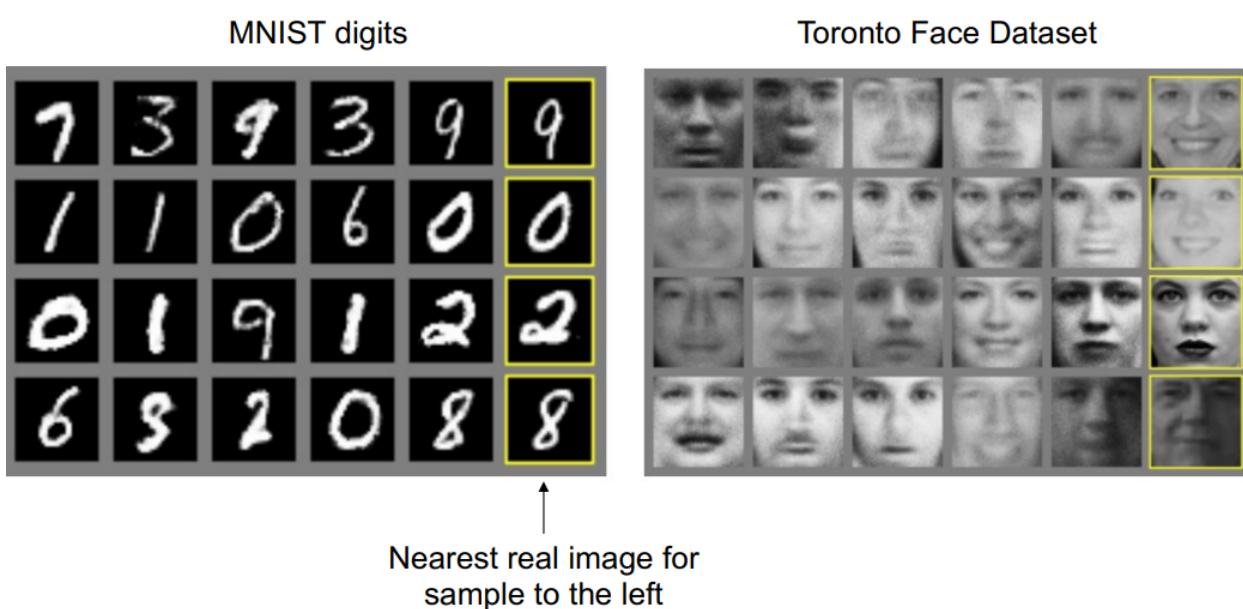
Quindi uno è black box per l’altro, ma non il contrario.

---

A test time, rimuoviamo il discriminatore, che non è più usato, e usiamo solamente il generatore, a cui diamo samples random. Vedremo poi che anche il discriminatore può essere usato per qualcosa, ma per ora usiamo solo il generatore.

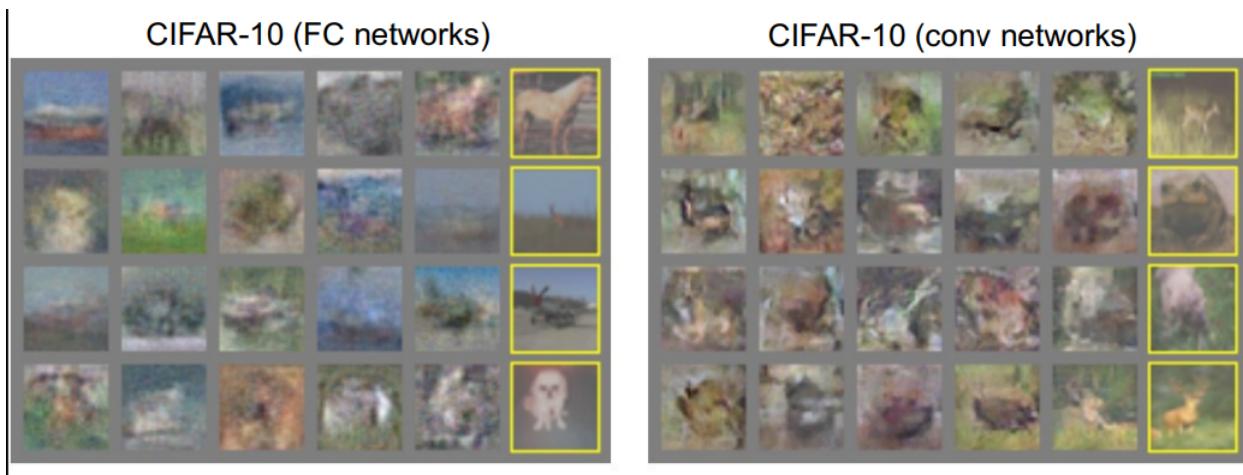
Questi sono i risultati del paper originale di questo argomento (2014), a cui p stato applicato il MNIST dataset.

Qui sta, appunto, generando samples simili a quelli nel training set. Sono simili, ma non identici.



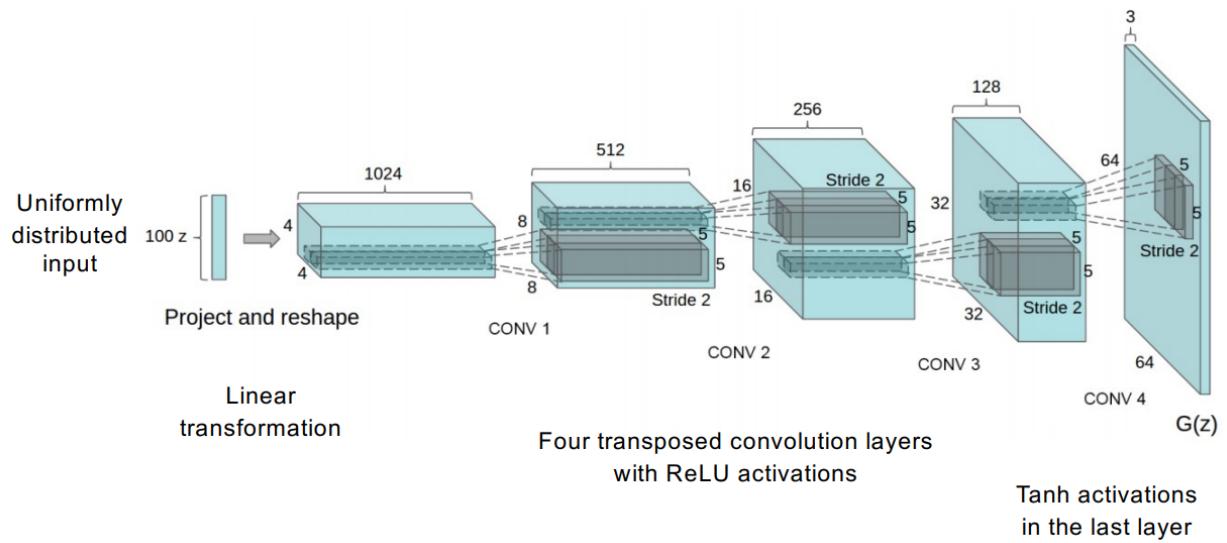
Nell'esempio a destra vediamo che le immagini generate sono molto noisy (quelle non gialle).

Hanno anche provato con dataset colorati:



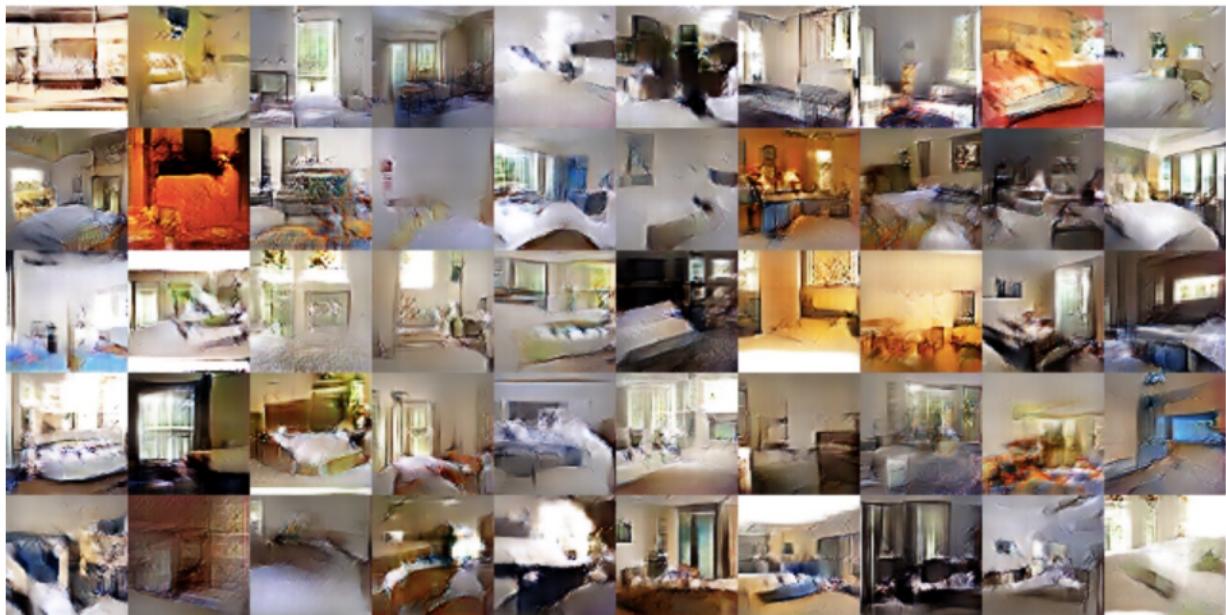
## DCGAN

**DCGAN (Deep Convolutional Generative Adversarial Network).** In questo paper del 2016 sono stati proposti dei principi per usare un'architettura convoluzionale per le GAN.



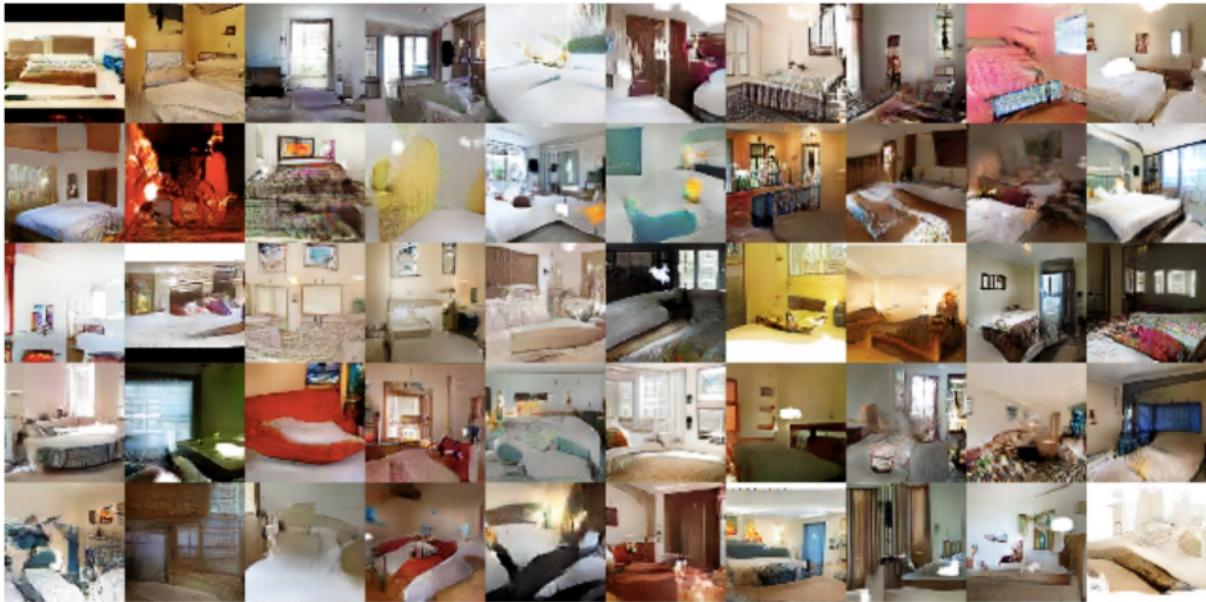
Si arriva nell'ultimo layer a 3 color channels,  $64 \times 64$ . Se la vediamo al contrario, è una convolutional neural network classica.

Generated bedrooms after one epoch



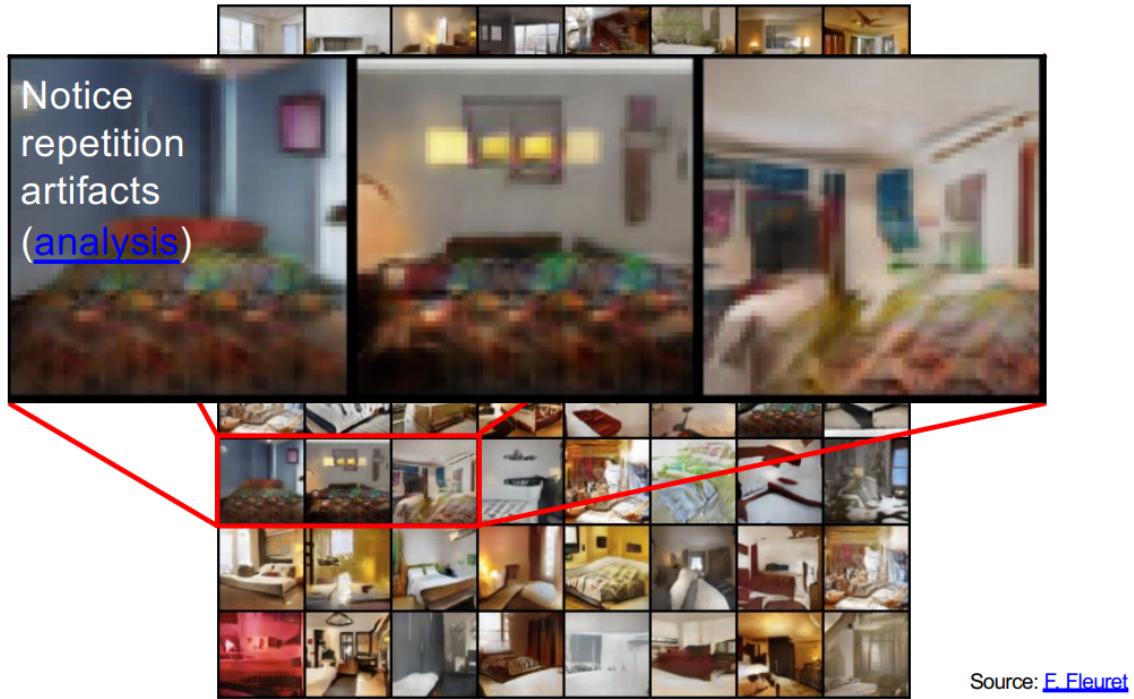
Mancano molto i dettagli qui sopra.

## Generated bedrooms after five epochs



Dopo 5 epoche i risultati migliorano.

## Generated bedrooms from reference implementation



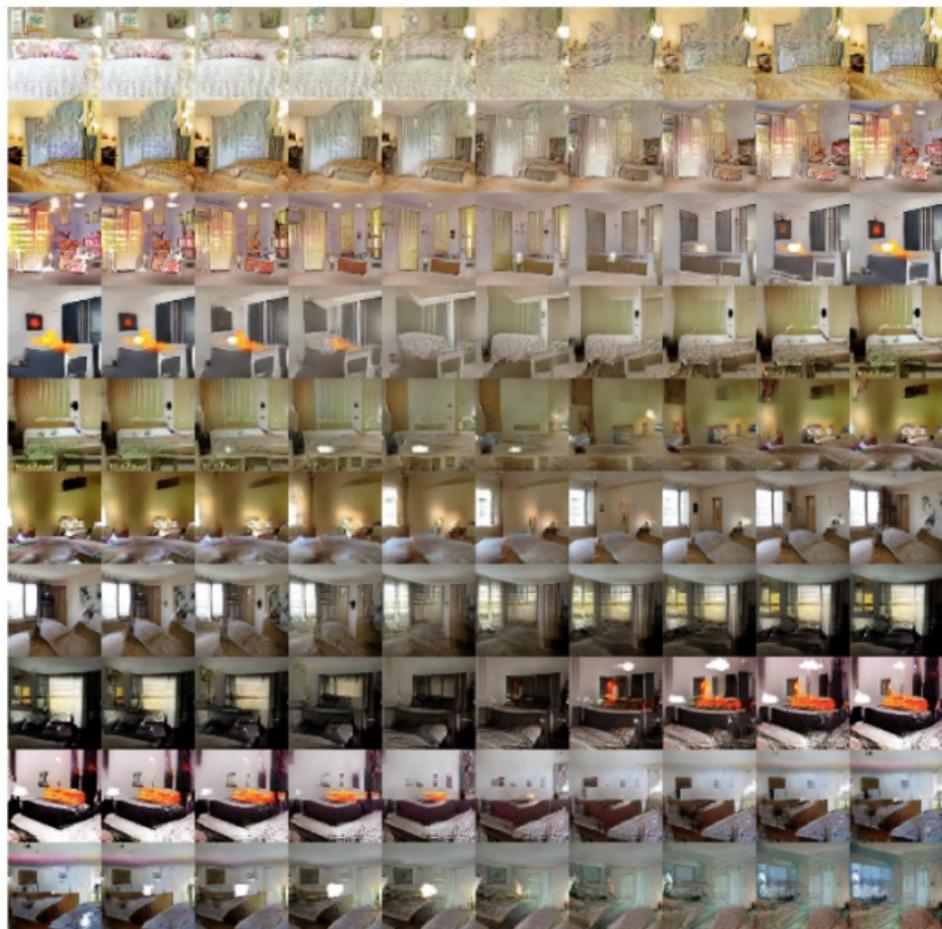
Source: [F. Fleuret](#)

Nei risultati finali vediamo dei problemi, come una ripetizione di patterns, e una mancanza di risoluzione, non si capisce bene cosa c'è nell'immagine.

## Proprietà

Una cosa interessante sono le **proprietà**. Se prendo un vector random, genero un'immagine, e poi faccio la stessa cosa con un altro vettore, se faccio una linear interpolation (nell'immagine, a destra un'immagine generata e a sinistra un'altra, andando verso il centro c'è l'interpolazione. Raggiungiamo un morphing tra un'immagine e l'altra.

Interpolation between different points in the z space

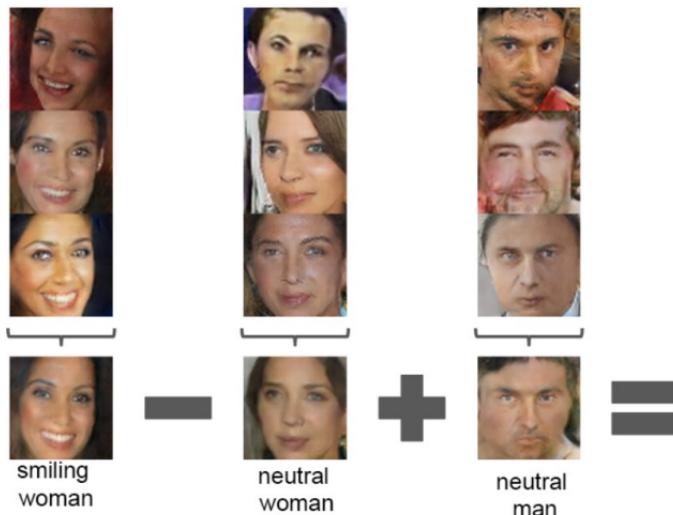


Quindi il contenuto viene mischiato ma mantenendo il contenuto, non è solo un mix.

Abbiamo anche delle aritmetiche di vettori che funzionano. Se prendiamo il random vector che genera una donna che sorride, e sottraiamo quello di una

donna che non sorride, e sommiamo il concetto di un uomo che non sorride, otteniamo un uomo che sorride.

- Vector arithmetic in the z space

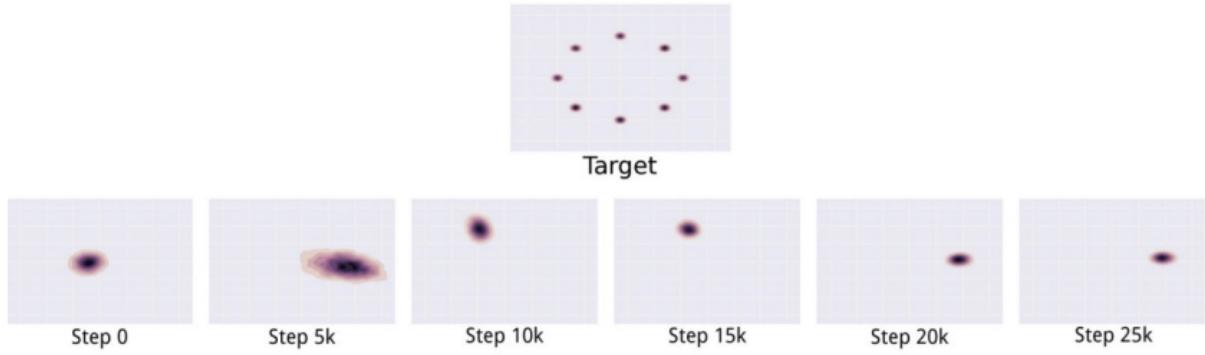


## Problemi delle GANs

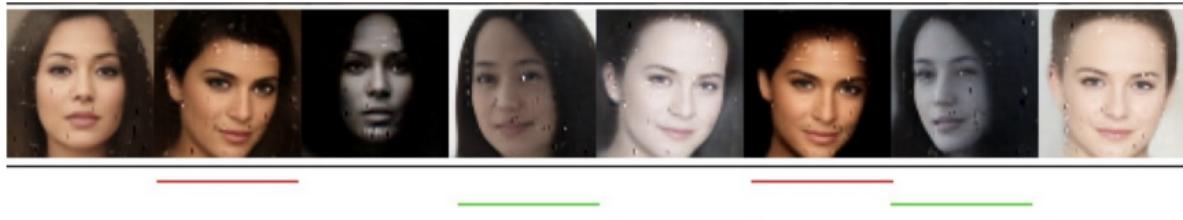
**Stabilità:** i parametri del modello possono facilmente oscillare o anche divergere, in particolare sono molto sensibili a **hyperparameter selection**. Nelle CNNs, anche se gli iperparametri non sono settati, di solito si riesce comunque a convergere ad una soluzione, magari non quella ottima. Qui invece la soluzione è molto sensibile alla selezione di iperparametri, che va fatta in modo attento.

Un altro problema è che **la generator loss non si correla alla qualità dei samples**. Se la loss aumenta (visto che stiamo massimizzando) non possiamo dire che i risultati stiano migliorando visualmente.

Un altro problema è il **mode collapse**. Supponiamo di essere in questa configurazione (Target), con le diverse classi che compongono il training set. Quello che può succedere, è che ad una certa epoca sto generando samples solo per una categoria, quindi a ciascuna epoca sto generando una classe diversa, ma solo quella. Quindi il modello non sta facendo sampling dalla distribuzione, ma ogni volta si concentra solo su una classe.



Questo può succedere in scenari più complicati, dove genera diverse facce ma della stessa identità, sta facendo solamente diverse versioni, ma della stessa persona



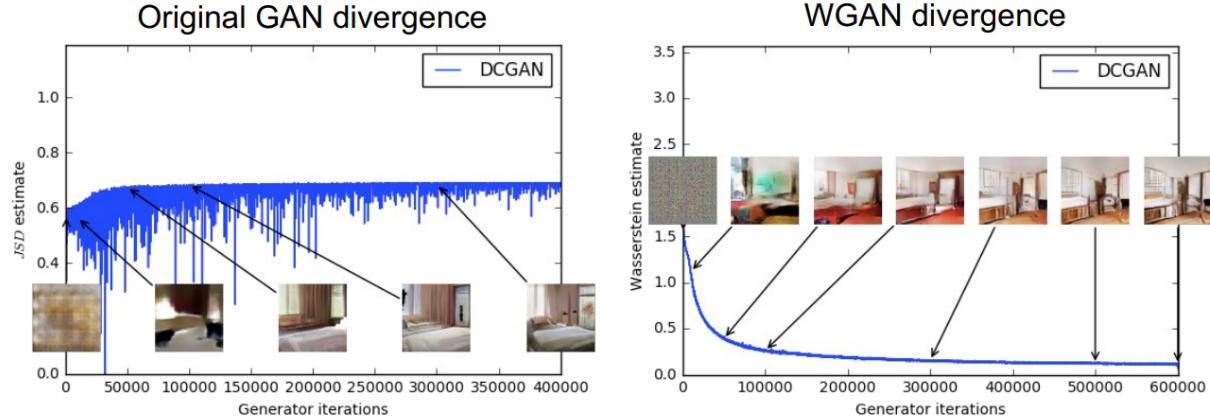

---

Diverse formalizzazioni di GANs sono state fatte per risolvere questo problema.

### **Wasserstein GAN**

Durante il training, nel nostro metodo, ci sono tanti drops, e anche se i risultati sono molto diversi, la loss è simile (non è correlata alla visual quality).

Invece con la **Wasserstein GAN** è stata introdotta una loss che è molto più smooth, e vediamo con ridurre la loss ha un impatto sulla qualità e sulla quantità di dettagli.



## How to evaluate GANs?

Vedere dei samples non è sufficiente, perchè stiamo prendendo samples da una distribuzione infinita.

Il modo più semplice, è quello di fare una sorta di **turing test**, chiedendo ad utenti, di selezionare quali esempi sono reali e quali fake.

Altri approcci sono basati sulle **distanze**. Da un lato, possiamo dire che se siamo bravi a generare animali, allora potremmo dire che il modello che li genera bene, dovrebbe anche essere in grado di riconoscerli bene. Possiamo anche vedere la distanza tra le immagini generate e le immagini del training set.

Quindi possiamo usare delle features estratte da un pre-trained classifier, per capire quanto siamo bravi.

- Key idea: generators should produce images with a variety of recognizable object classes
  - Defined as  $IS(G) = \exp[\mathbb{E}_{x \sim G} KL(P(y|x) \parallel P(y))]$  where  $P(y|x)$  is the posterior label distribution returned by an image classifier (e.g., InceptionNet) for sample  $x$ 
    - If  $x$  contains a recognizable object, entropy of  $P(y|x)$  should be low
    - If generator generates images of diverse objects, the marginal distribution  $P(y)$  should have high entropy
  - Disadvantage: a GAN that simply memorizes the training data (overfitting) or outputs a single image per class (mode dropping) could still score well

Un'altra possibilità, visto che siamo imparando a fare samples della distribuzione originale, potremmo fare molti samples e poi misuriamo come la distribuzione di questi samples è diversa dalla distribuzione delle immagini originali.