

# Analisi di Immagini e Video (Computer Vision)

Giuseppe Manco

# Outline

- Generative Modeling
  - Motivazioni
  - task
- Approcci
  - Variational Autoencoders
  - Generative Adversarial Networks

# Crediti

- Alcune slides adattate da altri corsi:
  - Computer Vision (I. Gkioulekas) - CS CMU Edu

# Recap: ML estimation

- Dato un campione  $X = \{x_1, x_2, \dots, x_n\}$ 
  - Generato da una distribuzione (sconosciuta)  $\mathbb{P}_r$
- E data una distribuzione candidata  $\mathbb{P}_\theta$  parametrizzata da  $\theta$
- Troviamo il parametro  $\hat{\theta}$  che ottimizza la verosimiglianza:

$$\begin{aligned}\hat{\theta} &= \operatorname{argmax}_\theta P_\theta(X) \\ &= \operatorname{argmax}_\theta \prod_i P_\theta(x_i) \\ &= \operatorname{argmax}_\theta \sum_i \log P_\theta(x_i) \\ &= \operatorname{argmax}_\theta \mathbb{E}_{x \sim \mathbb{P}_r} \log P_\theta(x)\end{aligned}$$

# Problemi con ML

- ML è consistente: in linea di massima, può apprendere qualsiasi distribuzione, se osserva una quantità infinita di dati e lo spazio dei parametri è completo
  - Minimizzare la Likelihood è equivalente a minimizzare la Kullback-Leibler (KL) divergence tra la vera distribuzione  $\mathbb{P}_r$  e la distribuzione candidata  $\mathbb{P}_\theta$

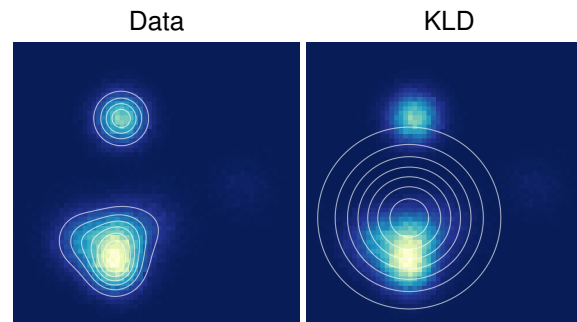
$$KL[\mathbb{P}_r|\mathbb{P}_\theta] = \int P_r(x) \log \frac{P_r(x)}{P_\theta(x)} dx$$

- Tuttavia, nella realtà (a causa della mis-specifica del modello e della quantità di dati finite), tende a produrre modelli **overgeneralized**

# Problemi con ML

$$KL[\mathbb{P}_r | \mathbb{P}_\theta] = \int P_r(x) \log \frac{P_r(x)}{P_\theta(x)} dx$$

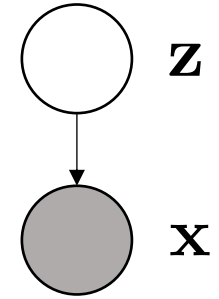
- Quando  $P_r(x) > P_\theta(x)$ , larghe regioni di  $\mathbb{P}_r$  assumono valori bassi in  $\mathbb{P}_\theta$ . Il loro contributo sulla  $KL[\mathbb{P}_r | \mathbb{P}_\theta]$  tende a infinito.
- Tuttavia, quando  $P_r(x) < P_\theta(x)$ ,  $x$  ha una bassa (true) probability, ma un'alta probabilità di essere generato dal modello. Il contributo a  $KL[\mathbb{P}_r | \mathbb{P}_\theta]$  tende a 0.



Source: [Theis et al 2016]

# Latent generative models

- Assumiamo un processo stocastico, governato da variabili latent  $\mathbf{z}$
- Deep latent generative models:
  - $P(\mathbf{x}|\mathbf{z})$  è definite da una rete neurale
  - Variational Autoencoders
  - **Generative Adversarial Networks**



$$\mathbf{z} \sim P_{\phi}(\cdot)$$
$$\mathbf{x} \sim P_{\theta}(\cdot|\mathbf{z})$$

$$P(\mathbf{x}) = \int P(\mathbf{x}|\mathbf{z})P(\mathbf{z})d\mathbf{z}$$

# Variational Autoencoders

- Definiamo una proposal distribution  $Q_\phi$  parametrizzata da  $\phi$
- Approssimiamo la log-likelihood con

$$\log P(x) \geq \underbrace{\mathbb{E}_{z \sim Q_\phi} [\log P_\theta(x|z)] - \text{KL}[Q_\phi(z)|P(z)]}_{\text{Evidence Lower Bound (ELBO)}}$$

Evidence Lower Bound (ELBO)

- Ottimizziamo ELBO su  $\phi$  e  $\theta$



# Variational Autoencoders

- Vantaggi:
  - Robusto all'overfitting (effetto regolarizzazione sulle variabili latenti)
  - Le variabili latent sono interpretabili (tramite  $P(z|x)$ )
- Svantaggi: esempi imprecisi (blurry)
  - ML-oriented training
  - ELBO è un'approssimazione della log-likelihood
  - Assunzioni sulla prior di  $z$

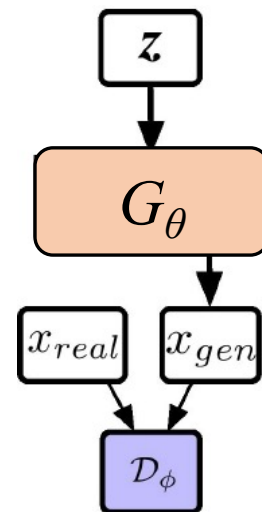
# Generative Adversarial Networks (GANs)

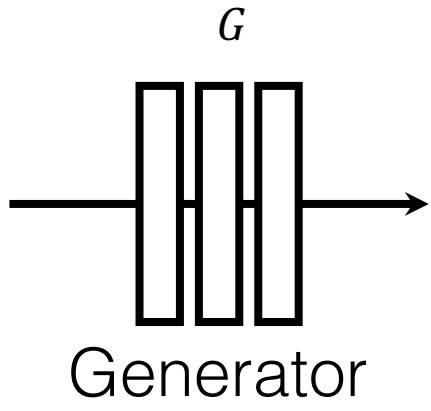
- **Modello**

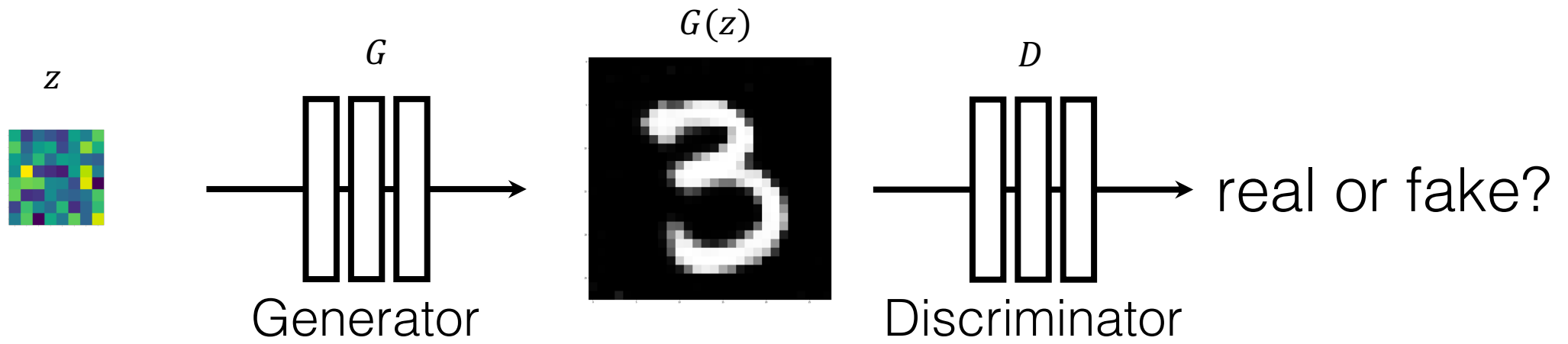
- Obiettivo: addestrare una funzione  $G_\theta$  che trasforma un valore random  $z$  in un dato
  - Nessuna assunzione sulle distribuzioni dei dati o sulle variabili latenti
  - Implementa la funzione di sampling da  $P_\theta(x)$  in maniera efficiente
  - Può riprodurre qualsiasi distribuzione  $P(x)$  se  $G_\theta$  è sufficientemente complessa

- **Learning as a two-player game**

- *Discriminator*  $D_\phi$ : Addestrato a discriminare tra dati reali e dati generati
- *Generator*  $G_\theta$ : Addestrato a generare esempi realistici per confondere il discriminatore

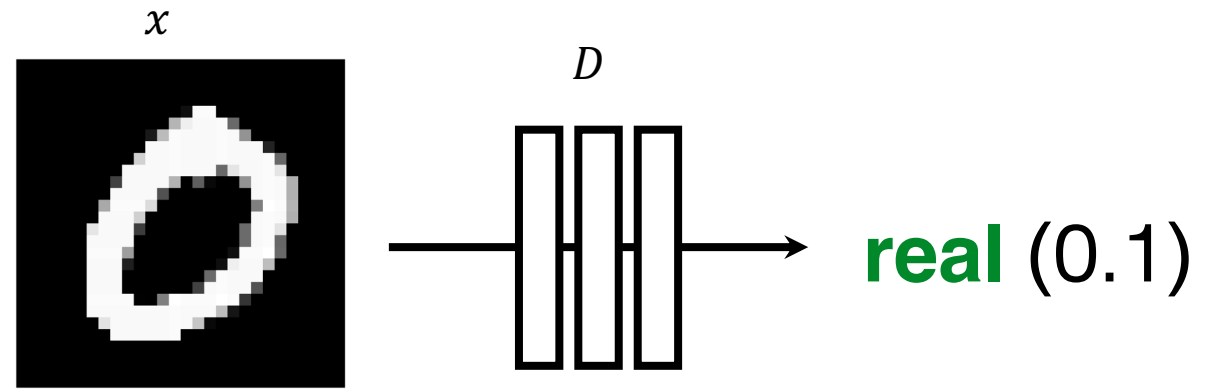
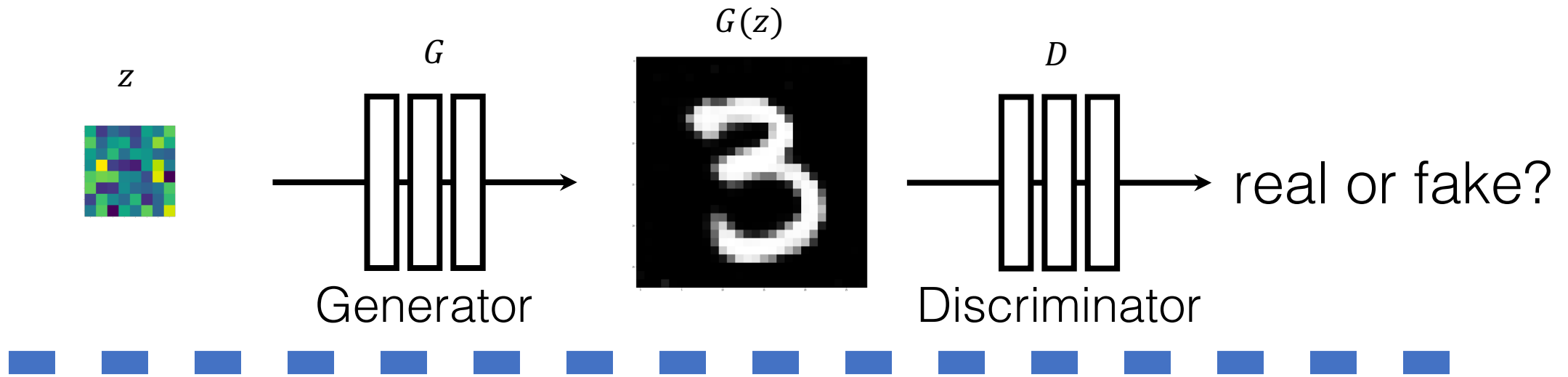






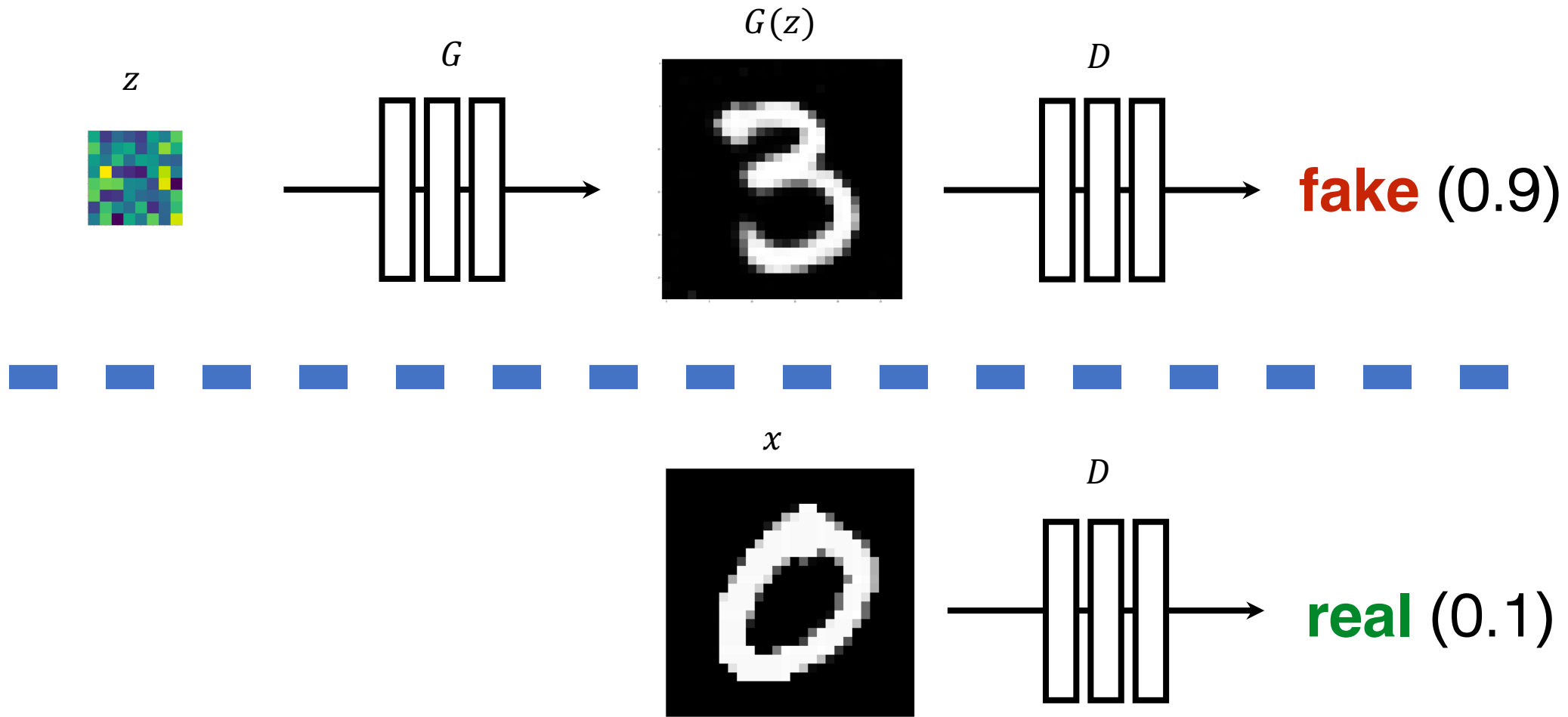
**G** sintetizza immagini che confondono **D**

**D** identifica le fakes

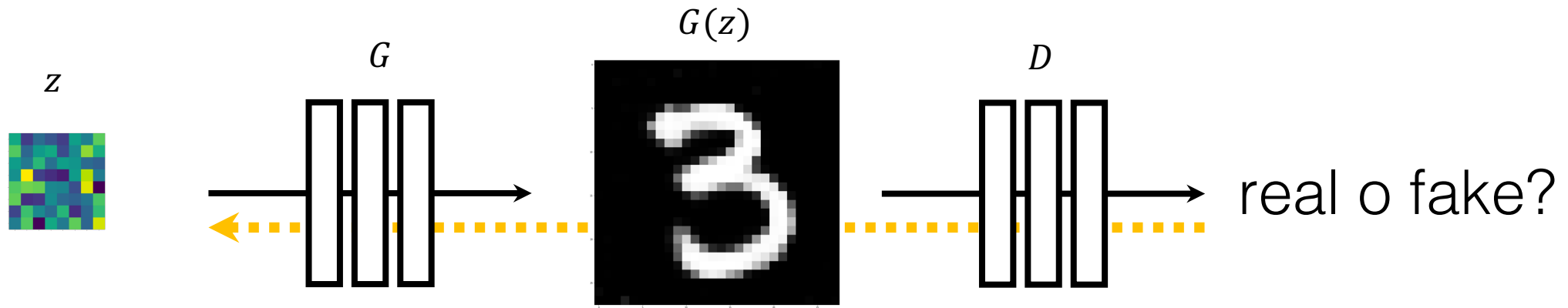


$argmax_D$

$$\mathbb{E}_x [\log D(x)]$$

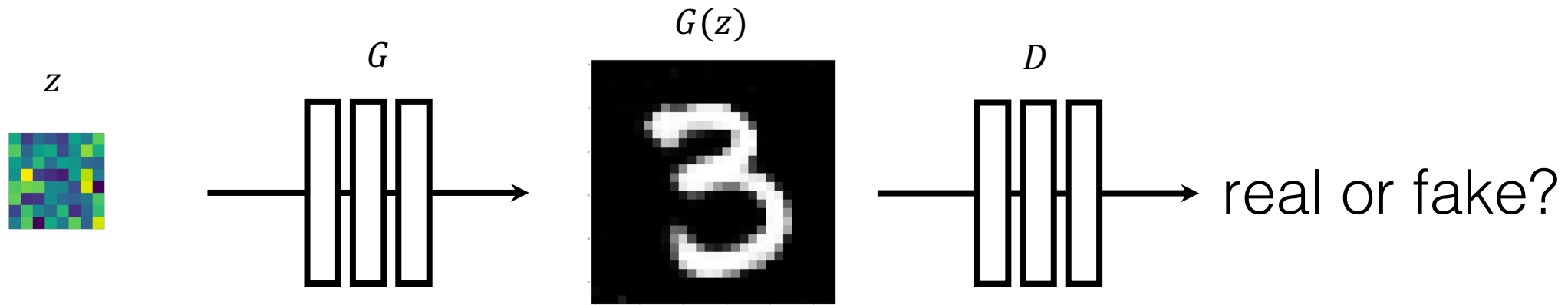


$$\operatorname{argmax}_D \operatorname{argmin}_G \mathbb{E}_x [\log D(x) + \log(1 - D(G(z)))]$$



**G** sintetizza le immagini che *confondono* **D**:

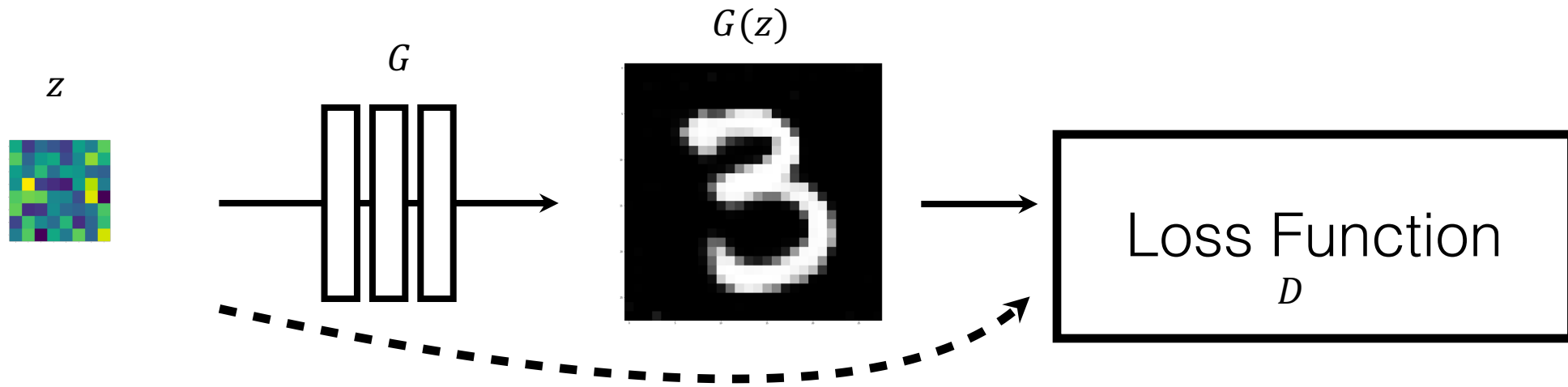
$$\operatorname{argmax}_D \operatorname{argmin}_G \mathbb{E}_x [\log D(x) + \log(1 - D(G(z)))]$$



**G** sintetizza le immagini che *confondono* il *miglior D*:

$$\mathop{\text{argmax}}_D \mathop{\text{argmin}}_G \mathbb{E}_x [\log D(x) + \log(1 - D(G(z)))]$$





Per  $\mathbf{G}$ ,  $\mathbf{D}$  è una funzione di loss.

Piuttosto che essere definita a mano, è *appresa*.

# Perché è meglio della ML?

- Discriminator Loss:

$$L_D(\phi, \theta) = \mathbb{E}_{x \sim \mathbb{P}_r} [\log D_\phi(x)] + \mathbb{E}_{x \sim \mathbb{P}_\theta} [\log(1 - D_\phi)]$$

- Generator loss

$$L_G(\phi, \theta) = -L_D(\phi, \theta)$$

- Adversarial Game

$$\max_{\phi} \min_{\theta} \mathbb{E}_{x \sim \mathbb{P}_r} [\log D_\phi(x)] + \mathbb{E}_{x \sim \mathbb{P}_\theta} [\log(1 - D_\phi)]$$

# Perché è meglio della ML?

- Optimal discriminator:

$$L_D(\phi, \theta) = \int p_r(x) \log D(x) + p_\theta(x) \log(1 - D(x)) dx$$

- Massimizzando il termine all'interno dell'integrale rispetto a  $D(x)$ :

$$D^*(x) = \frac{P_r(x)}{P_r(x) + P_\theta(x)}$$

Perché è meglio della ML?

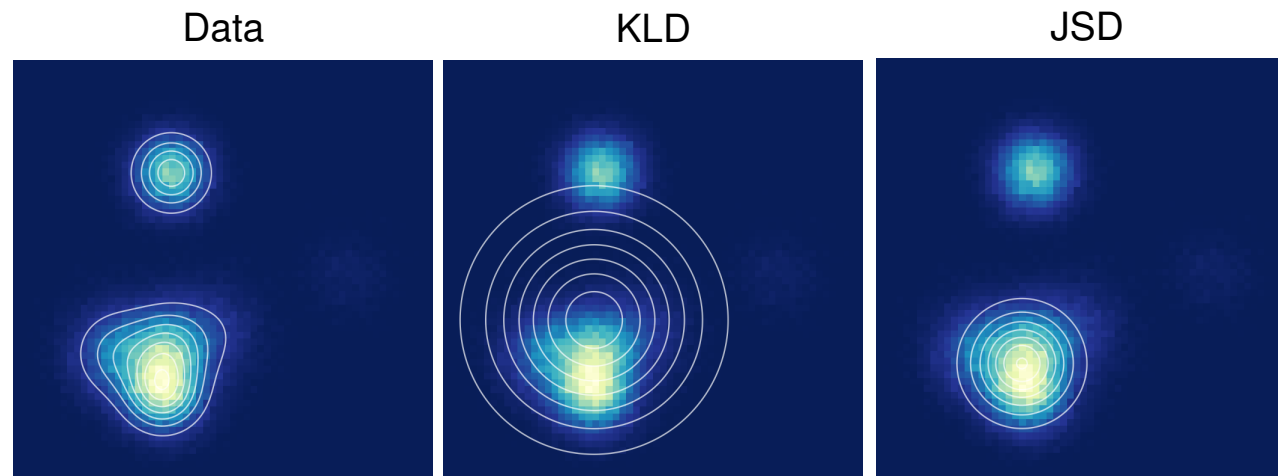
$$\begin{aligned}L_D(D^*, \theta) &= \int p_r(x) \log D(x) + p_\theta(x) \log(1 - D(x)) dx \\&= \int p_r(x) \log \frac{P_r(x)}{P_r(x) + P_\theta(x)} + p_\theta(x) \log \frac{P_\theta(x)}{P_r(x) + P_\theta(x)} dx \\&= KL \left[ \mathbb{P}_r \mid \frac{\mathbb{P}_\theta + \mathbb{P}_r}{2} \right] + KL \left[ \mathbb{P}_\theta \mid \frac{\mathbb{P}_\theta + \mathbb{P}_r}{2} \right] - 2 \log 2 \\&= 2JS[\mathbb{P}_r \mid \mathbb{P}_\theta] - 2 \log 2\end{aligned}$$

# Perché è meglio della ML?

$$L_G(D^*, \theta) \approx JS[\mathbb{P}_r | \mathbb{P}_\theta]$$

- Minimizzare su  $\theta$  equivale a minimizzare la Jensen-Shannon Divergence

$$JS[\mathbb{P}_r | \mathbb{P}_\theta] = \frac{1}{2} KL[\mathbb{P}_r | \mathbb{P}_\theta] + \frac{1}{2} KL[\mathbb{P}_\theta | \mathbb{P}_r]$$



Source: [Theis et al 2016]

# Schema GAN

---

**Algorithm 1** Inference algorithm.

---

- 1 Initialize  $\phi$  and  $\theta$
- 2 **for** number of epochs **do**
- 3     **for**  $k$  steps **do**
- 4         Sample  $\{\tilde{x}_\theta^{(1)}, \dots, \tilde{x}_\theta^{(m)}\}$  from  $\mathbb{P}_\theta$ ;
- 5         Sample  $\{x^{(1)}, \dots, x^{(1)}\}$  from  $\mathbb{P}_r$ ;
- 6         Update  $\phi$  by ascending its stochastic gradient:

$$\nabla_\phi \frac{1}{m} \sum_{i=1}^m \left[ \log \left( D_\phi \left( x^{(i)} \right) \right) + \log \left( 1 - D_\phi \left( \tilde{x}_\theta^{(1)} \right) \right) \right]$$

- 7     **end for**
- 8         Sample  $\{\tilde{x}_\theta^{(1)}, \dots, \tilde{x}_\theta^{(m)}\}$  from  $\mathbb{P}_\theta$ ;
- 9         Update  $\theta$  by descending its stochastic gradient:

$$\nabla_\theta \frac{1}{m} \sum_{i=1}^m \log \left( 1 - D_\phi \left( \tilde{x}_\theta^{(i)} \right) \right)$$

- 10    **end for**
  - 11    Return  $\phi$  and  $\theta$ .
-

# Schema GAN

- Critico: Backpropagation dagli esempi

---

**Algorithm 1** Inference algorithm.

---

- 1 Initialize  $\phi$  and  $\theta$
- 2 **for** number of epochs **do**
- 3     **for**  $k$  steps **do**
- 4         Sample  $\{\tilde{x}_\theta^{(1)}, \dots, \tilde{x}_\theta^{(m)}\}$  from  $\mathbb{P}_\theta$ ;
- 5         Sample  $\{x^{(1)}, \dots, x^{(1)}\}$  from  $\mathbb{P}_r$ ;
- 6         Update  $\phi$  by ascending its stochastic gradient:

$$\nabla_\phi \frac{1}{m} \sum_{i=1}^m \left[ \log \left( D_\phi \left( x^{(i)} \right) \right) + \log \left( 1 - D_\phi \left( \tilde{x}_\theta^{(1)} \right) \right) \right]$$

- 7     **end for**
- 8         Sample  $\{\tilde{x}_\theta^{(1)}, \dots, \tilde{x}_\theta^{(m)}\}$  from  $\mathbb{P}_\theta$ ;
- 9         Update  $\theta$  by descending its stochastic gradient:

$$\nabla_\theta \frac{1}{m} \sum_{i=1}^m \log \left( 1 - D_\phi \left( \tilde{x}_\theta^{(i)} \right) \right)$$

- 10    **end for**
  - 11    Return  $\phi$  and  $\theta$ .
-

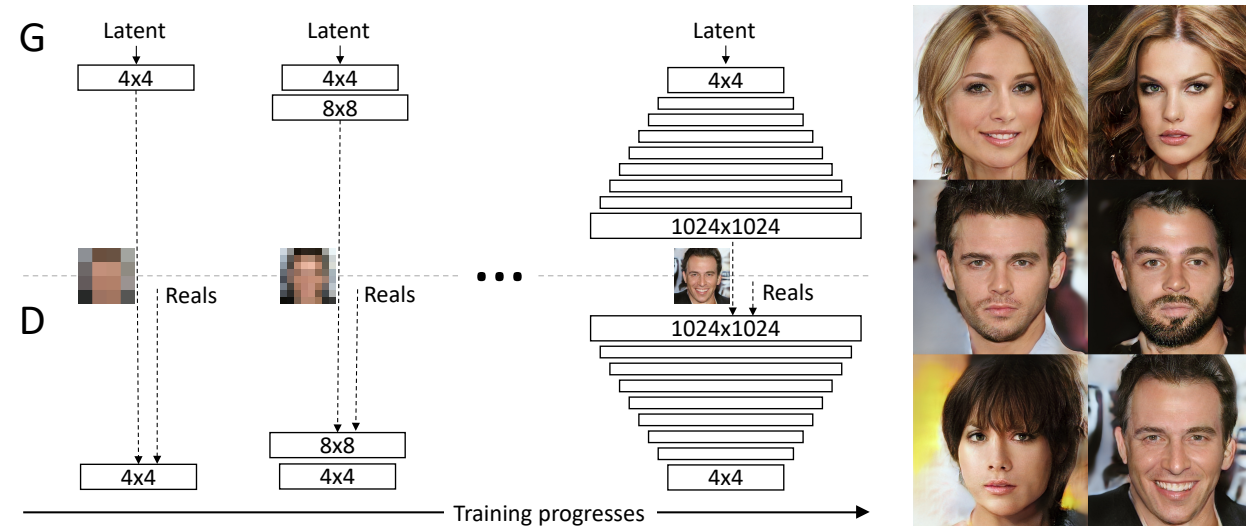
# Training Challenges

- Problemi
  - Mode Collapse
  - Slow Convergence
  - Overgeneralization
  - Instabilità
- Rimedi
  - Network Depth
  - Game setup, loss refinement
  - Hacks



# Network Depth

- Processo iterativo
  - Aumentiamo la complessità della rete progressivamente



# Game Setup

- Min-Max Gaming è instabile

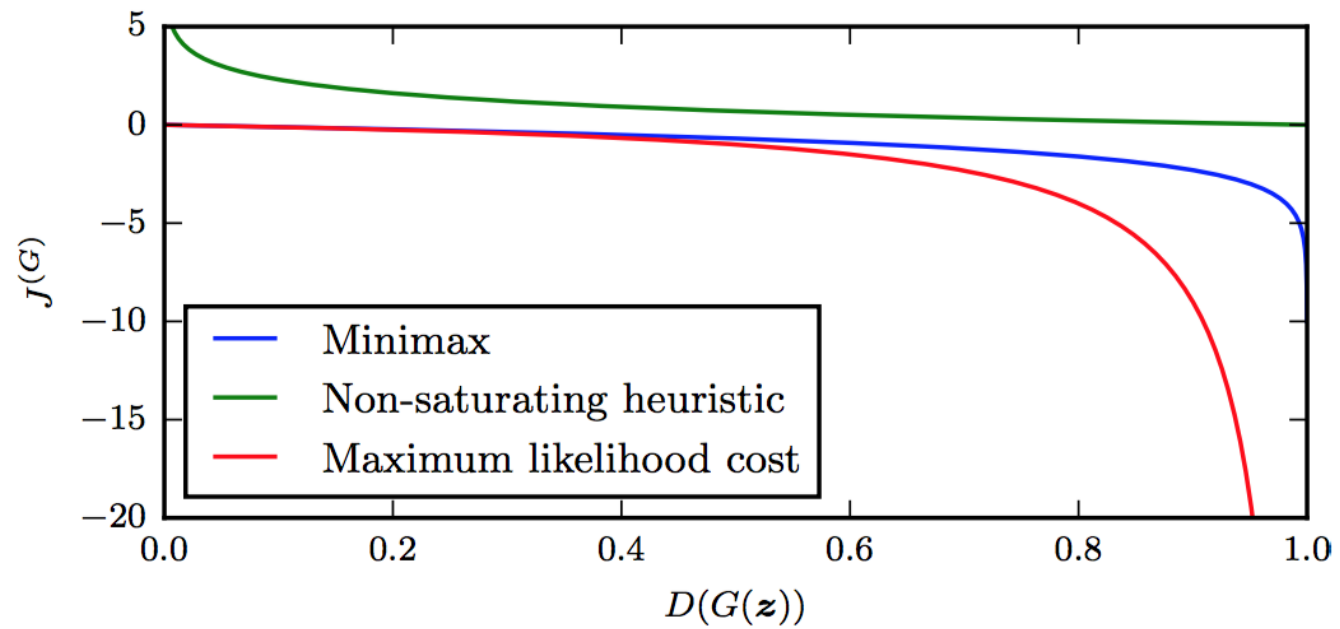
$$L_D(\phi, \theta) = \mathbb{E}_{x \sim \mathbb{P}_r}[\log D_\phi(x)] + \mathbb{E}_{x \sim \mathbb{P}_\theta}[\log(1 - D_\phi)]$$

$$L_G(\phi, \theta) = -L_D(\phi, \theta)$$

- Non-Saturating GAN

$$L_G(\phi, \theta) = \mathbb{E}_{x \sim \mathbb{P}_\theta}[\log(D_\phi)]$$

# Game setup



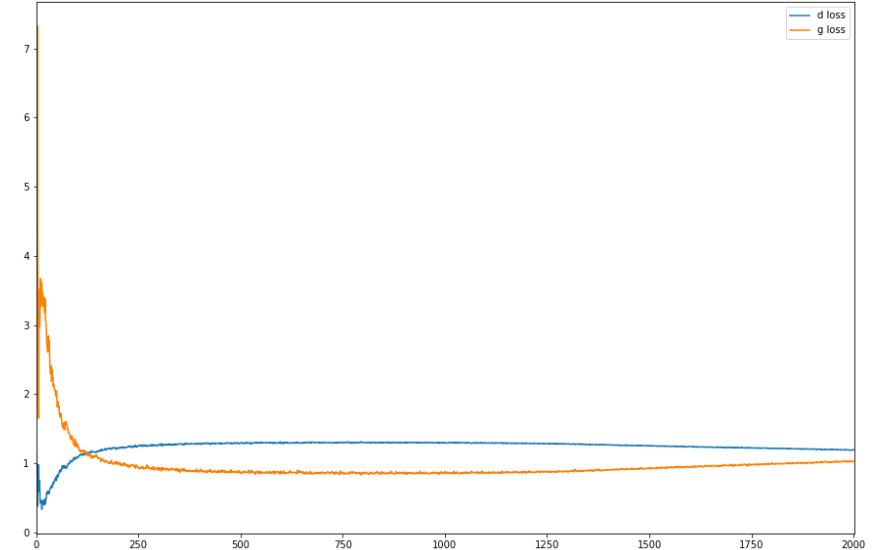
# Instabilità

- Quando fermare il training?
- Wassersteing GAN
  - Sostituisce la JS con EM distance

$$L_D(\phi, \theta) = \mathbb{E}_Z[f_\phi(G(z))] - \mathbb{E}_{x \sim \mathbb{P}_r}[f_\phi(x)]$$

$$L_G(\phi, \theta) = -L_D(\phi, \theta)$$

- $f_\phi$  1-Lipshitz:  $|f_\phi(x_1) - f_\phi(x_2)| \leq |x_1 - x_2|$ 
  - Gradient clipping
  - Gradient penalties



# Hacks

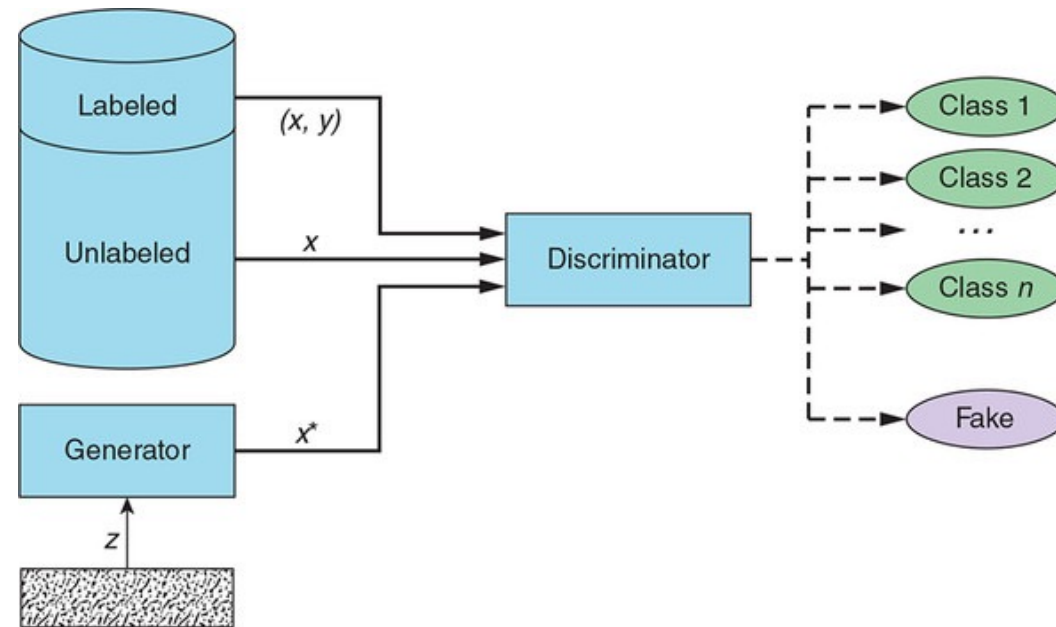
- Normalizzazione degli input
- Batch Normalization
- Gradient penalties
- Variare (aumentare) le iterazioni sul discriminatore
- Soft/noisy labels sul discriminator
- Evitare gradienti sparsi
  - No ReLU/MaxPool

# Sviluppi

- Semi-Supervised GAN
- Conditional GAN
- CycleGAN

# Semi-supervised GAN

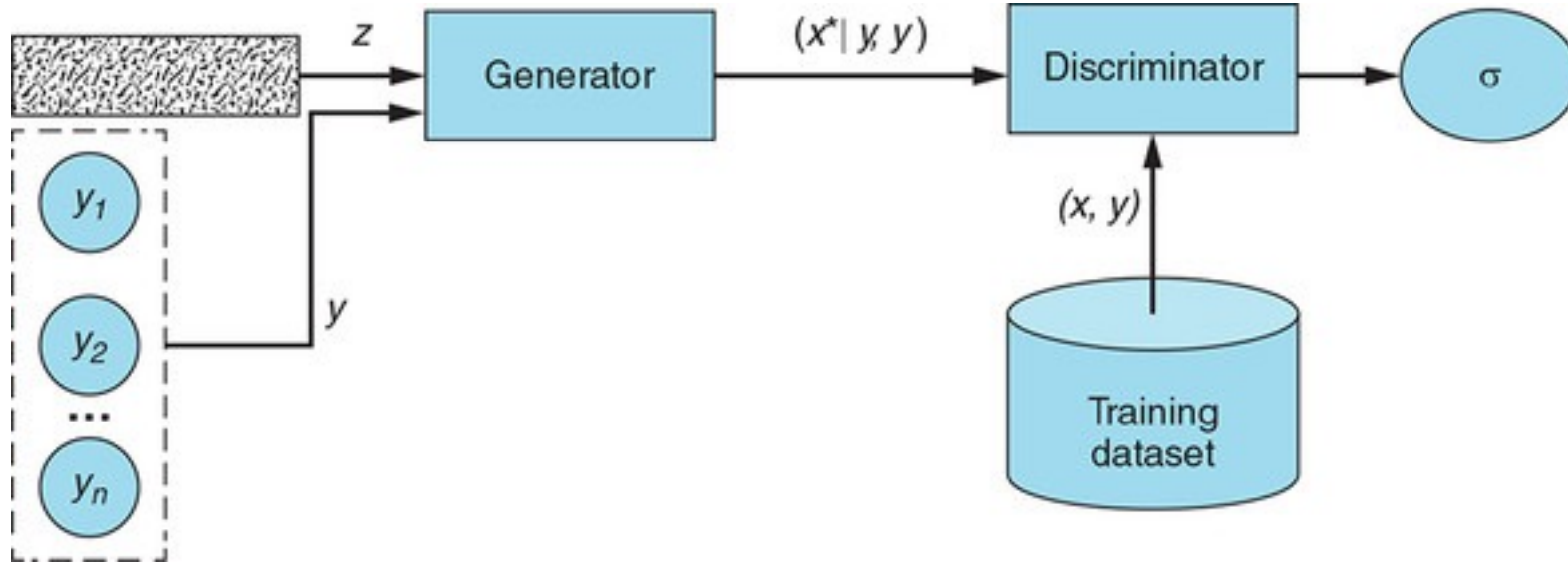
- Idea: utilizziamo il generator per irrobustire un discriminatore



$$L_D(\phi, \theta) = \mathbb{E}_{x, y \sim \mathbb{P}_r} [\log D_\phi(y|x)] + \mathbb{E}_{x \sim \mathbb{P}_\theta} \left[ \log \left( 1 - \sum_y D_\phi(y|x) \right) \right]$$

$$L_G(\phi, \theta) = \mathbb{E}_{x \sim \mathbb{P}_\theta} \left[ \log \left( \sum_y D_\phi(y|x) \right) \right]$$

# Conditional GAN

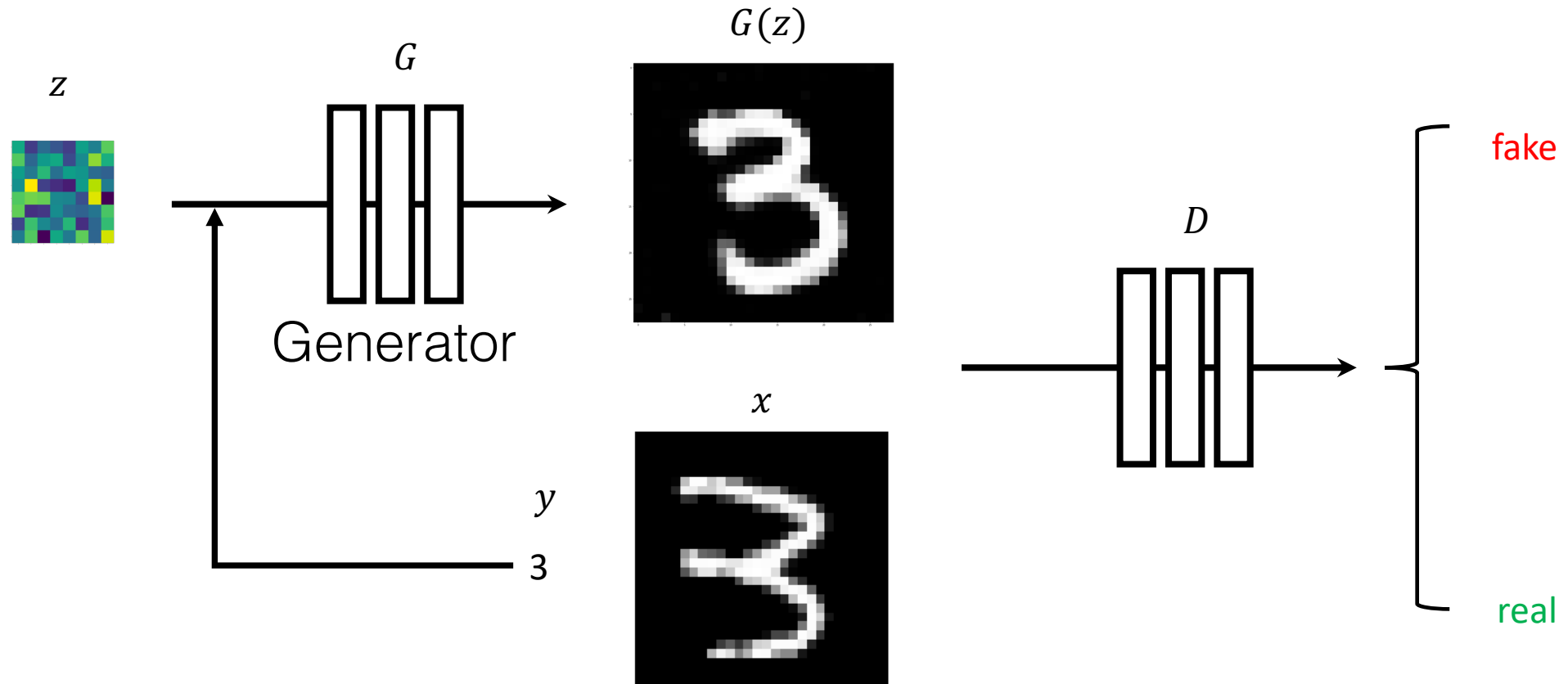


$$L_D(\phi, \theta) = \mathbb{E}_{x, y \sim \mathbb{P}_r} [\log D_\phi(y, x)] + \mathbb{E}_{x \sim \mathbb{P}_\theta(y)} \left[ \log \left( 1 - \sum_y D_\phi(y, x) \right) \right]$$

$$L_G(\phi, \theta) = \mathbb{E}_{x \sim \mathbb{P}_\theta(y)} \left[ \log \left( \sum_y D_\phi(y, x) \right) \right]$$



# Supervised GAN-MNIST



# Esempi: Super Resolution

original



bicubic  
(21.59dB/0.6423)



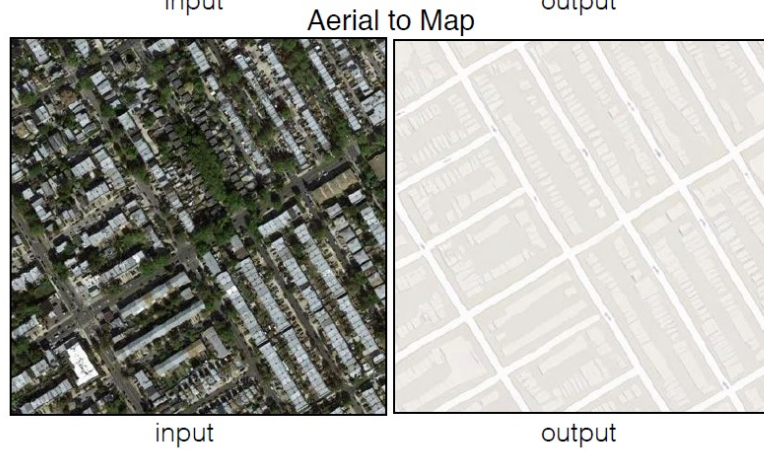
SRResNet  
(23.44dB/0.7777)



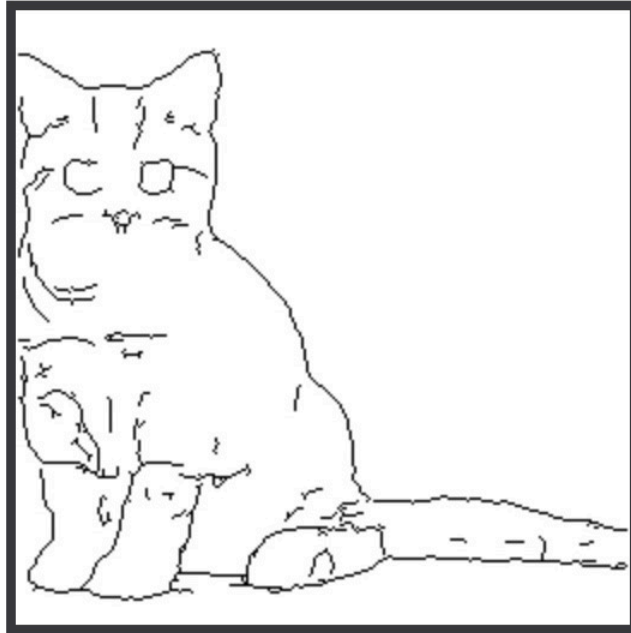
SRGAN  
(20.34dB/0.6562)



# Esempi: Image to Image Transalation



INPUT



undo

clear

random



OUTPUT

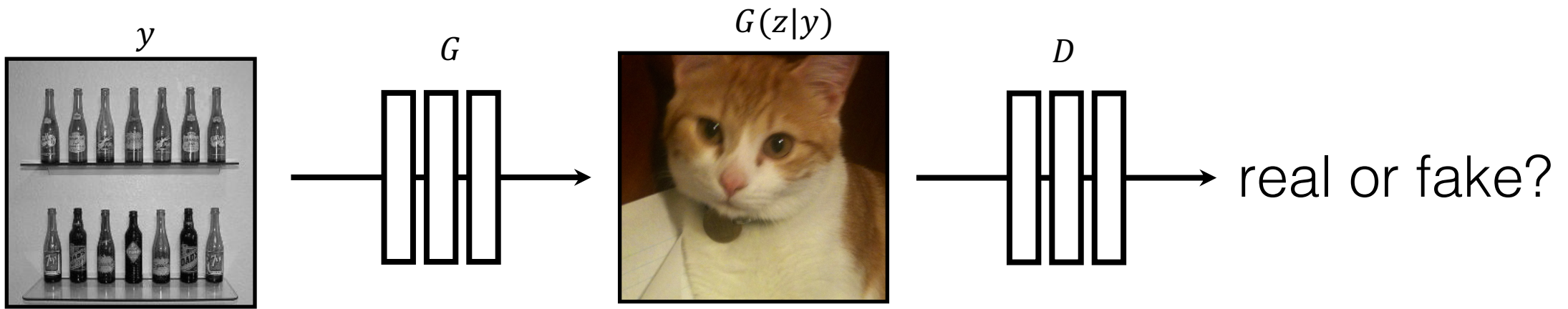


save

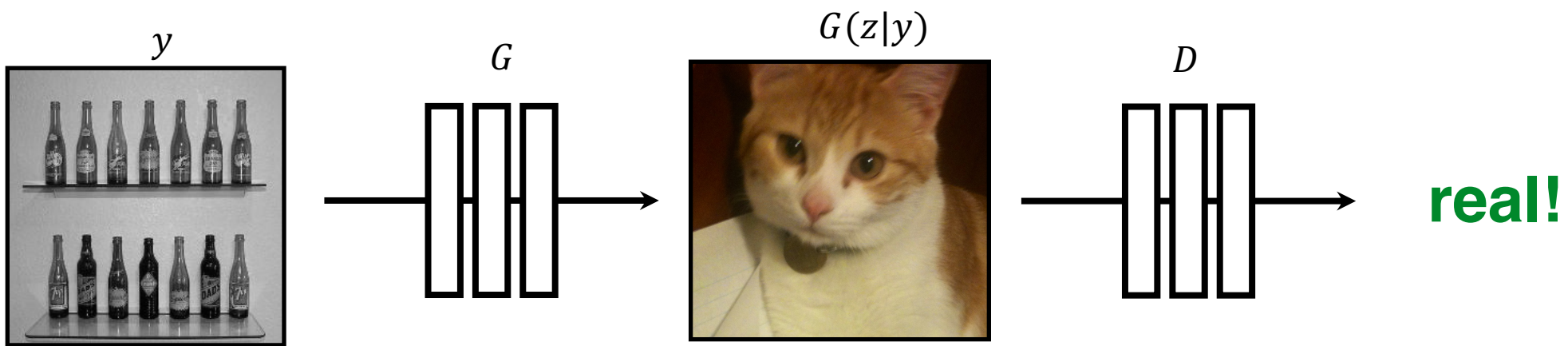
<https://affinelayer.com/pixsrv/>

# Esempio: Colorization

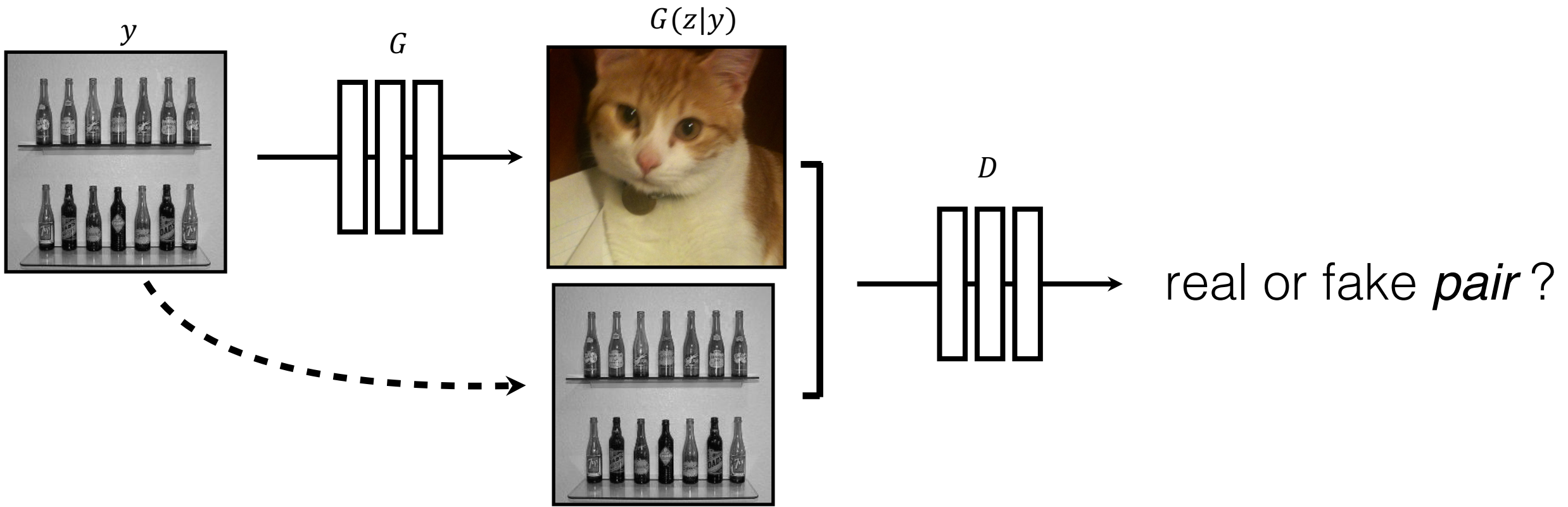




$$\operatorname{argmax}_D \operatorname{argmin}_G \mathbb{E}_{x,y} [\log D(x, y) + \log(1 - D(G(x|y), y))] ]$$

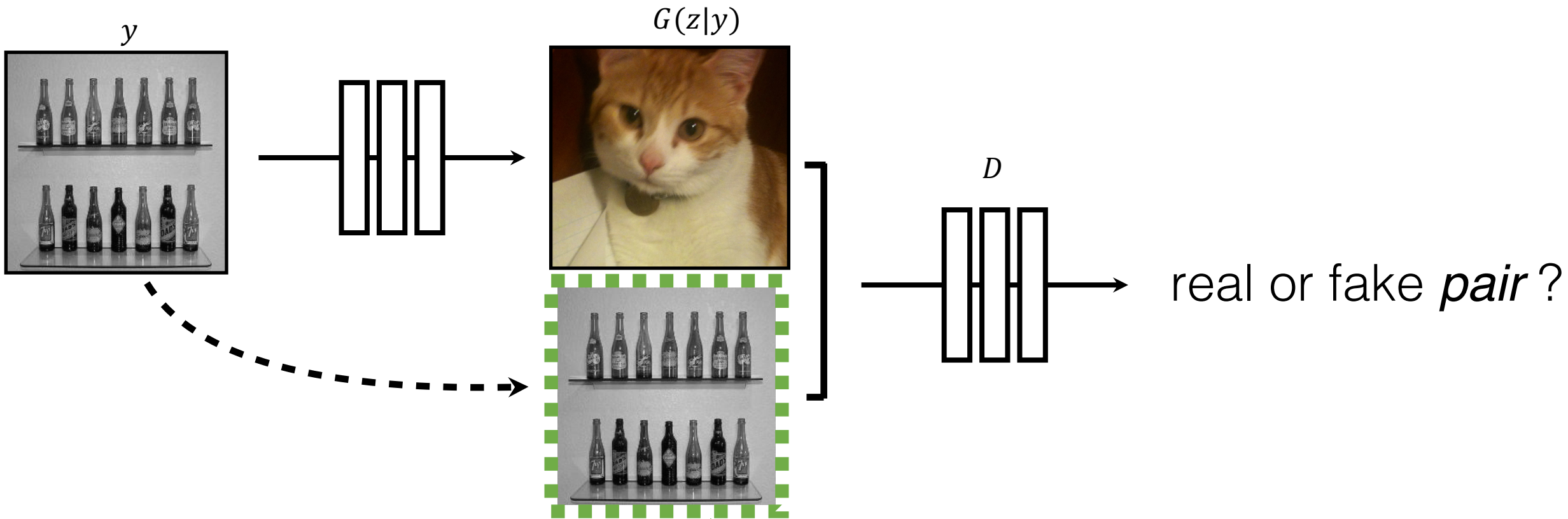


$$\operatorname{argmax}_D \operatorname{argmin}_G \mathbb{E}_{x,y} [\log D(x) + \log(1 - D(G(z|y)))]$$

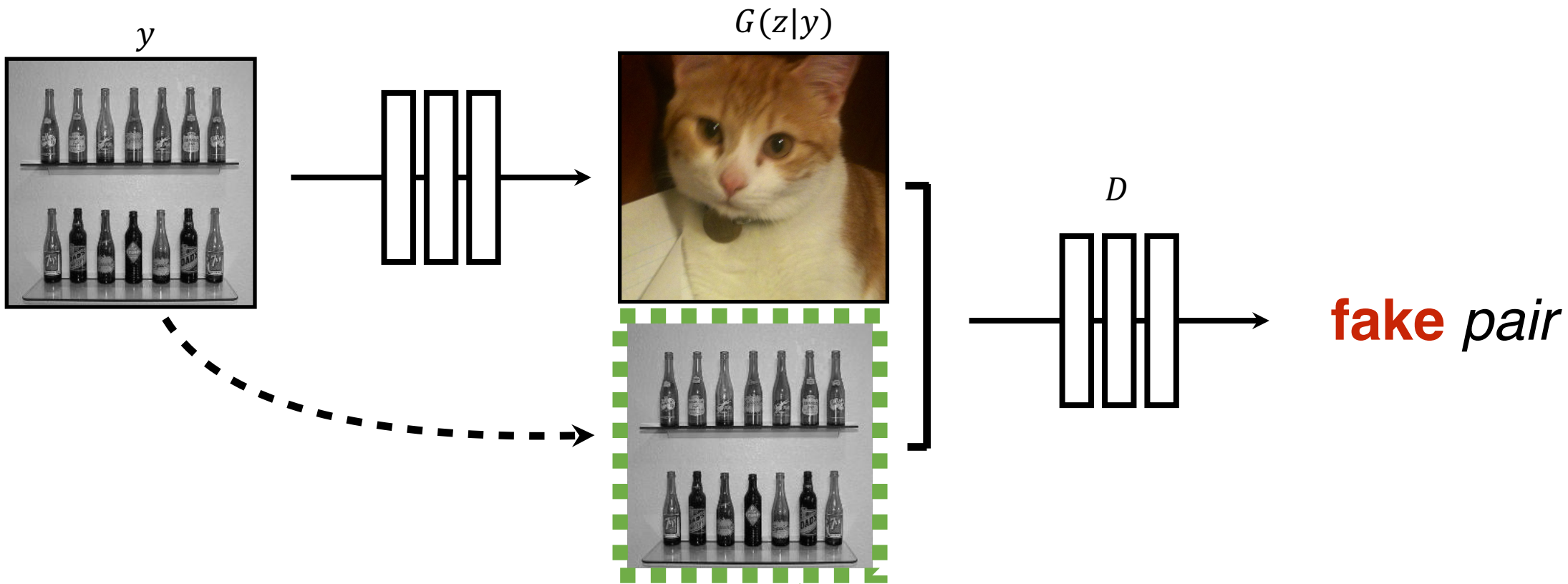


$$\operatorname{argmax}_D \operatorname{argmin}_G \mathbb{E}_{x,y} [\log D(x, y) + \log(1 - D(G(z|y), y))]$$

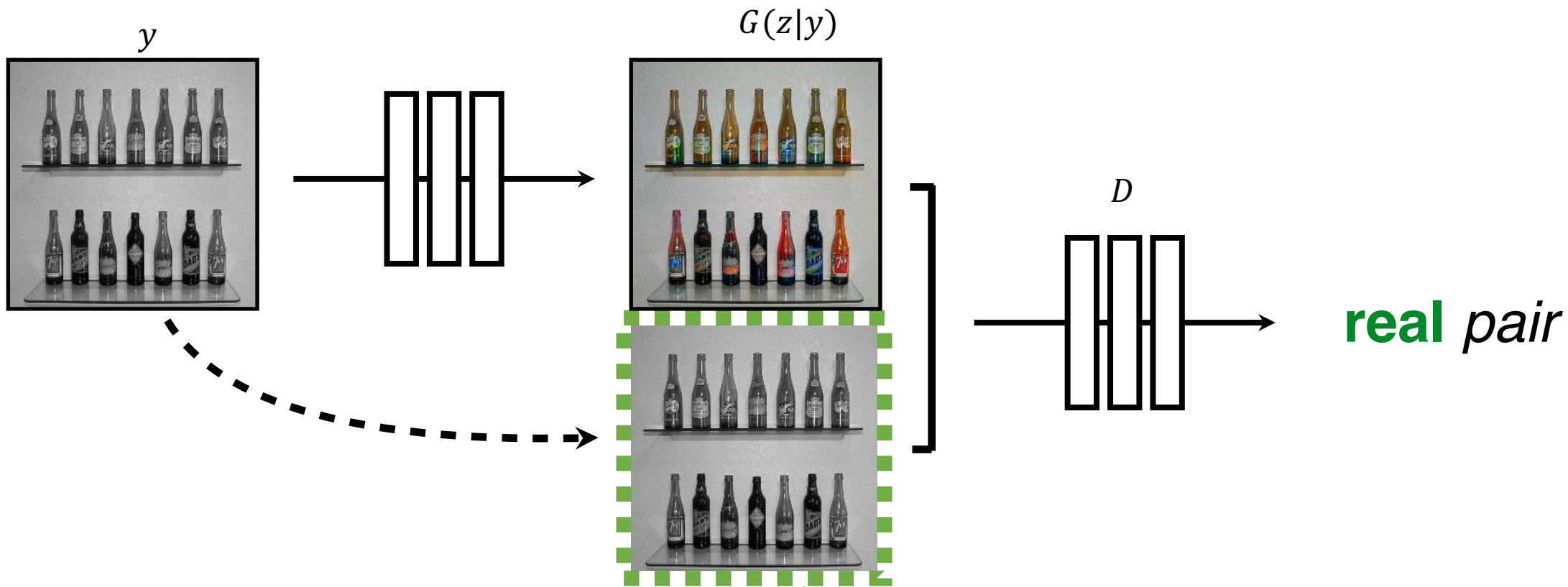




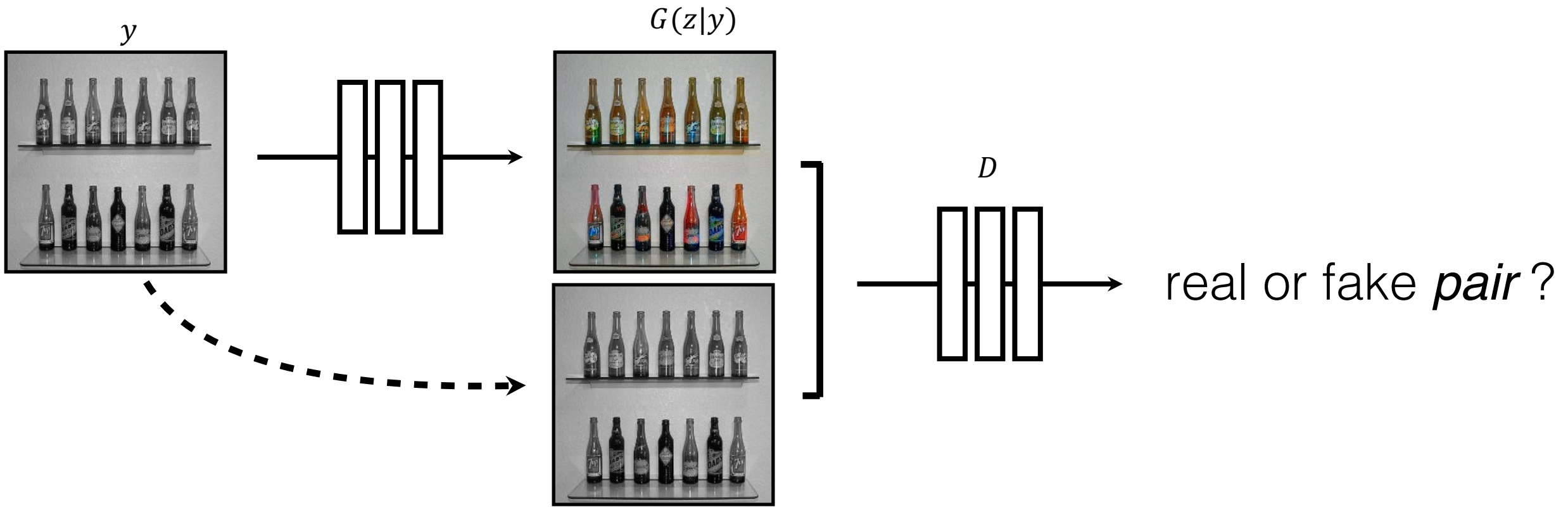
$$\operatorname{argmax}_D \operatorname{argmin}_G \mathbb{E}_{x,y} [\log D(x, \boxed{y}) + \log(1 - D(G(z|y), \boxed{y}))]$$



$$\operatorname{argmax}_D \operatorname{argmin}_G \mathbb{E}_{x,y} [\log D(x, \boxed{y}) + \log(1 - D(G(z|y), \boxed{y}))]$$



$$\operatorname{argmax}_D \operatorname{argmin}_G \mathbb{E}_{x,y} [\log D(x, \mathbf{y}) + \log(1 - D(G(z|y), \mathbf{y}))]$$



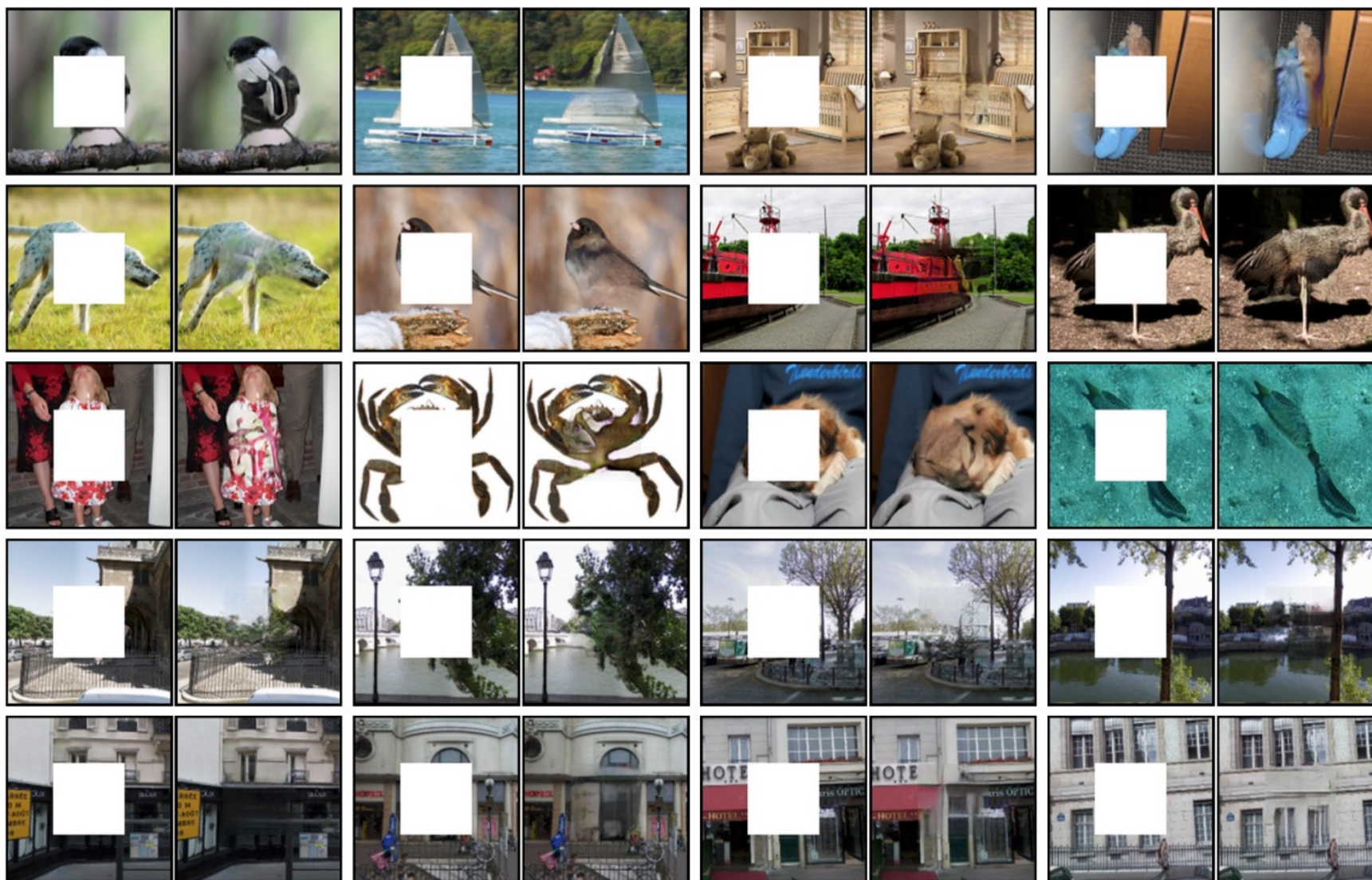
$$\operatorname{argmax}_D \operatorname{argmin}_G \mathbb{E}_{x,y} [\log D(x, y) + \log(1 - D(G(z|y), y))]$$

# Qual è il ruolo del random noise?

- Se l'input è sufficientemente complesso, CGAN non hanno bisogno di  $z$

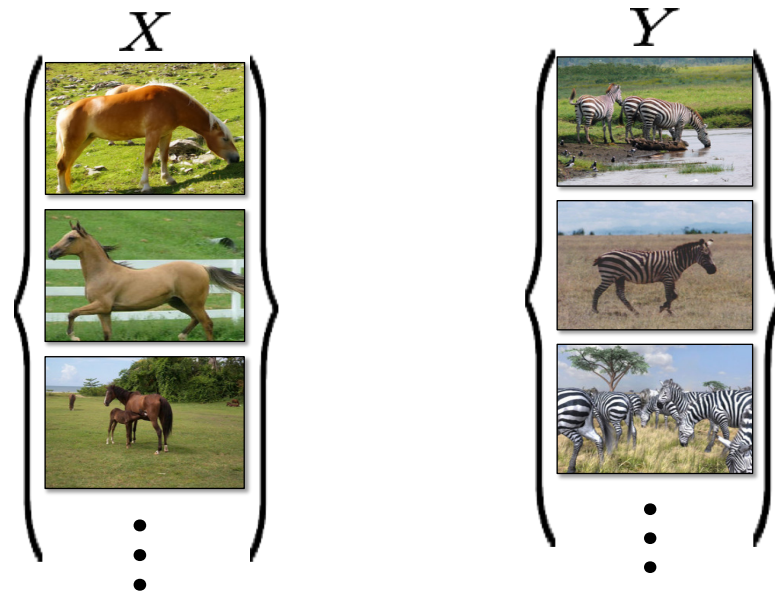
$$[\log D(x, y) + \log(1 - D(G(y), y))]$$

# Image Inpainting

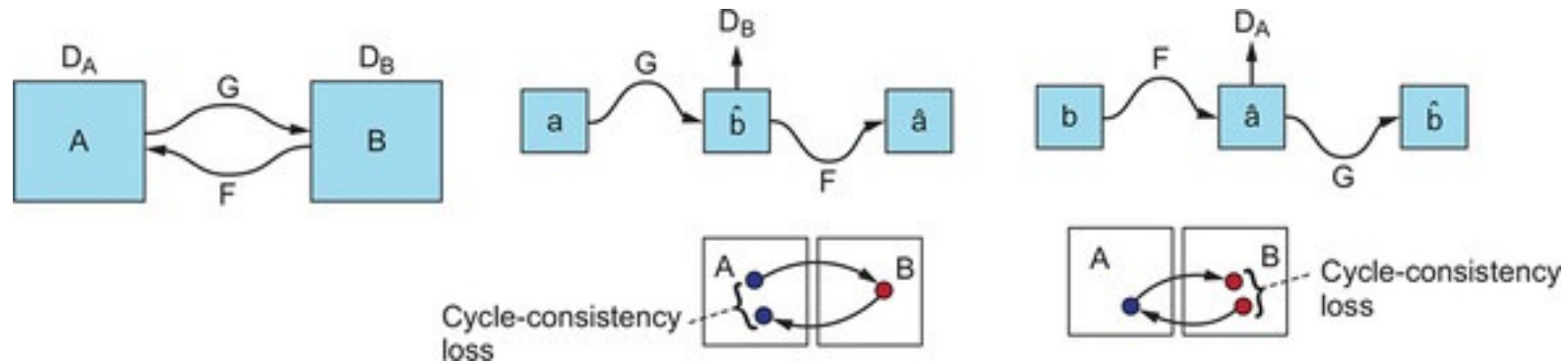


# Unpaired image translation

- Tutti gli esempi precedenti si basano sul fatto di poter accoppiare l'input all'output desiderato
- Che succede se le coppie sono inconsistenti?

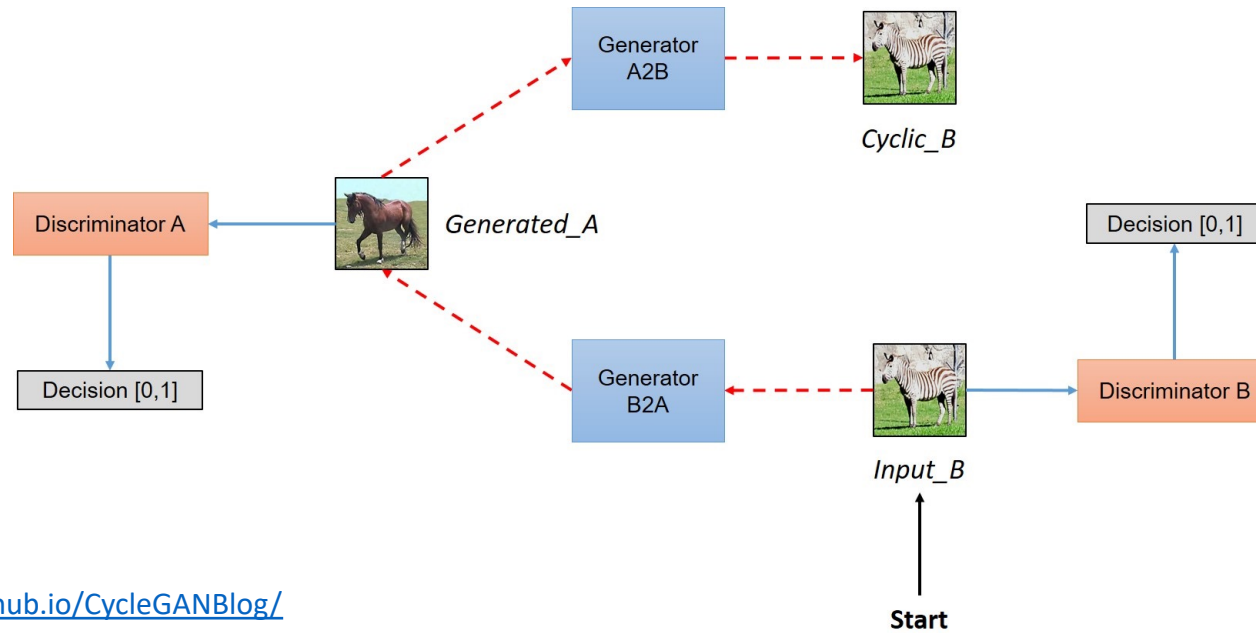
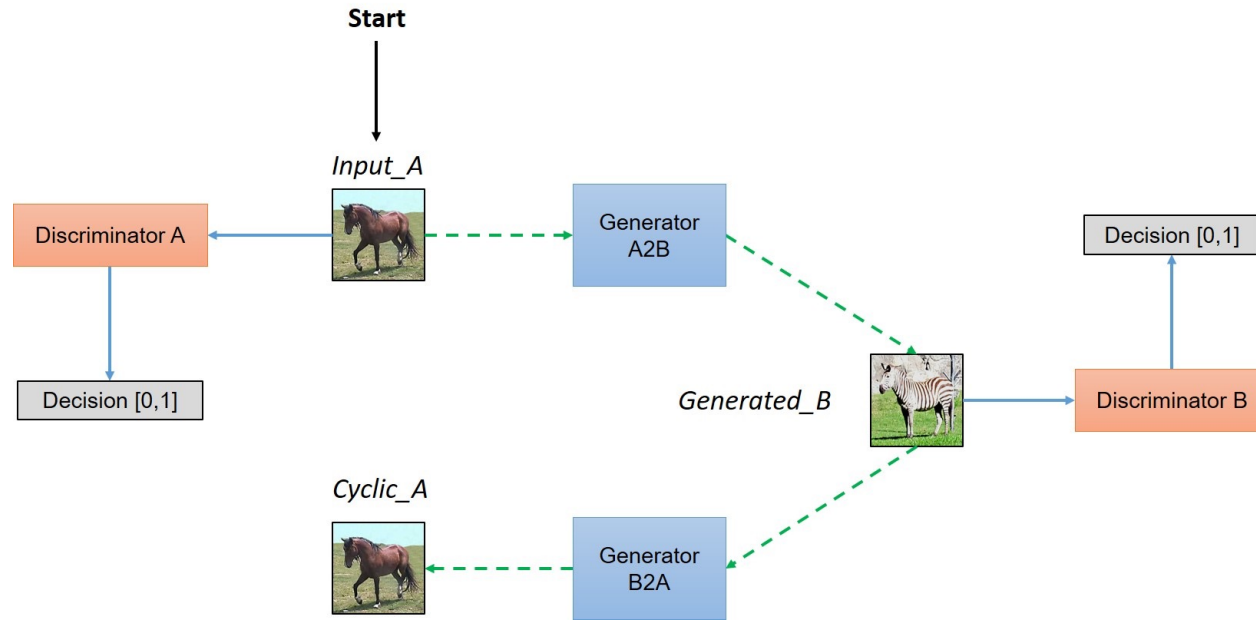


# CycleGAN



- Due Conditional GAN combine
  - Ogni discriminatore accetta due input
    - L'immagine originale corrispondente a quel dominio e l'immagine generata
- Aggiustamenti
  - Cycle-consistency loss
  - Identity loss





# CycleGAN

- Component-wise loss:

- $L_{GAN}(D_A, G) = \mathbb{E}_x [\log D_A(x) + \log(1 - D_A(G(x)))]$

- $L_{GAN}(D_B, F) = \mathbb{E}_y [\log D_B(x) + \log(1 - D_B(F(y)))]$

- Consistency loss:

- $L_{CYCLE}(G, F) = \mathbb{E}_x [\|F(G(x)) - x\|_1] + \mathbb{E}_y [\|G(F(y)) - y\|_1]$

- Overall loss:

- $L_{GAN}(D_A, D_B, G, F) = L_{GAN}(D_A, G) + L_{GAN}(D_B, F) + \lambda L_{CYCLE}(G, F)$

# Applicazioni

