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 $X = \{ x_1, x_2, \ldots, x_n \}$
  - Generato da una distribuzione (sconosciuta) $\mathbb{P}_{\mathrm{r}}$
- E data una distribuzione candidata $\mathbb{P}_\theta$ parametrizzata da θ

- Troviamo il parametro $\hat{\theta}$ che ottimizza la verosimiglianza:

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

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 ML è equivalente a minimizzare la Kullback-Leibler (KL) divergence tra la vera distribuzione $\mathbb{P}_{\mathrm{r}}$ e la distribuzione candidata $\mathbb{P}_{\theta}$

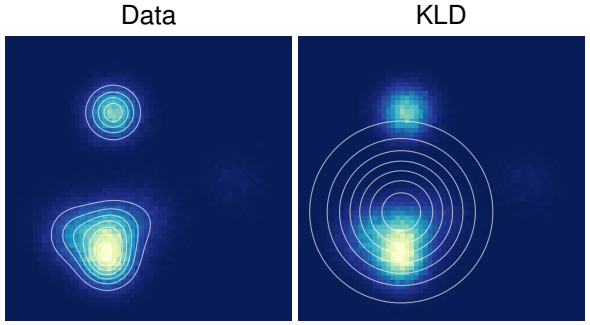
$$KL[\mathbb{P}_{\mathrm{r}}|\mathbb{P}_{\theta}] = \int P_r(x) \log \frac{P_r(x)}{P_{\theta}(x)} dx$$

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

Problemi con ML

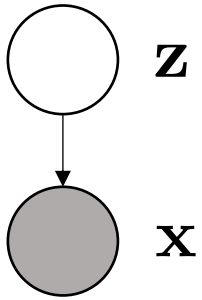
$$KL[\mathbb{P}_{\mathrm{r}}|\mathbb{P}_\theta] = \int P_r(x) \log \frac{P_r(x)}{P_\theta(x)} dx$$

- Quando $P_r(x) > P_\theta(x)$, larghe regioni di $\mathbb{P}_{\mathrm{r}}$ assumono valori bassi in $\mathbb{P}_\theta$. Il loro contributo sulla $KL[\mathbb{P}_{\mathrm{r}}|\mathbb{P}_\theta]$ tende a inifinito.
- 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 $KL[\mathbb{P}_{\mathrm{r}}|\mathbb{P}_\theta]$ tende a 0.



Source: [Theis at al 2016]

Latent generative models



- Assumiamo un processo stocastico, governato da variabili latent z

- Deep latent generative models:
  - $P(\boldsymbol{x}|\boldsymbol{z})$ è definite da una rete neurale
  - Variational Autoencoders
  - Generative Adversarial Networks

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

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

Variational Autoencoders

- Definiamo una proposal distribution Q_ϕ parametrizzata da ϕ
- Approssimiamo la log-likelihood con

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

- Ottimizziamo ELBO su ϕ e θ

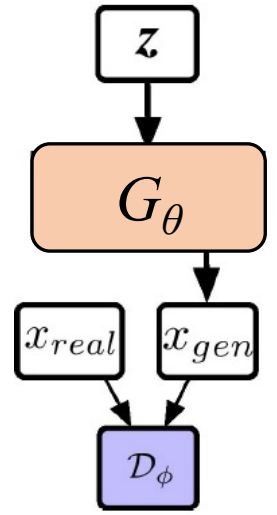
Variational Autoencoders

- Vantaggi:
  - Robusto all'overfitting (effetto regolarizzazione sulle variabili latenti)
  - Le variabili latent sono interpretabili (tramite $P(z|x)$)

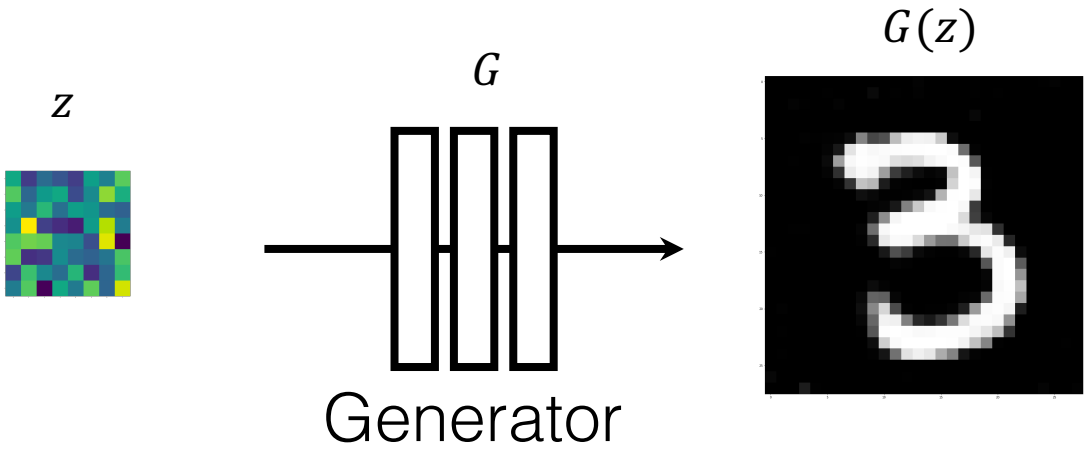
- Svantaggi: esempi imprecisi (blurry)
  - ML-oriented training
  - ELBO è un'approssimazione della log-likelihood
  - Assunzioni sulla prior di z

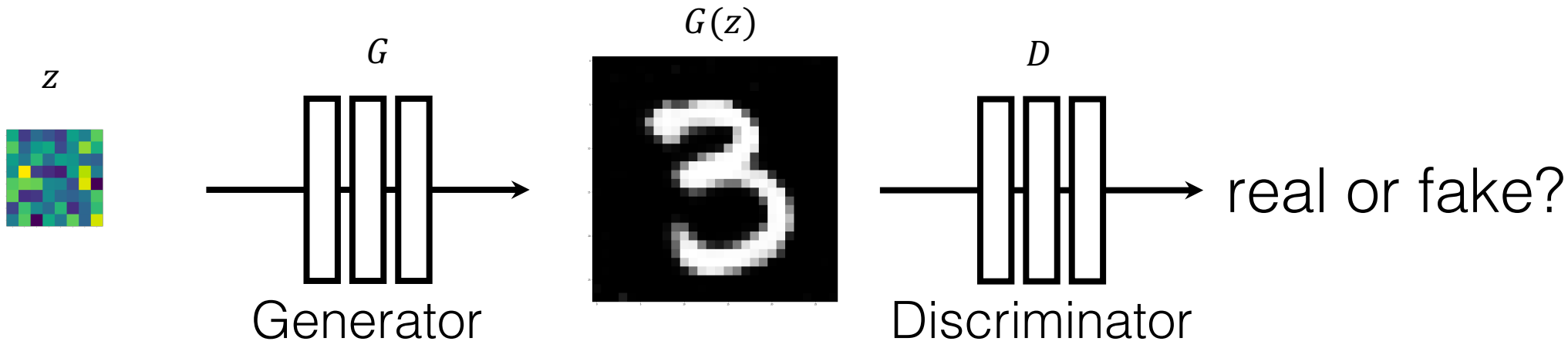
Generative Adversarial Networks (GANs)

- **Modello**
  - Obiettivo: addestrare una funzione G_θ 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_θ è sufficientemente complessa



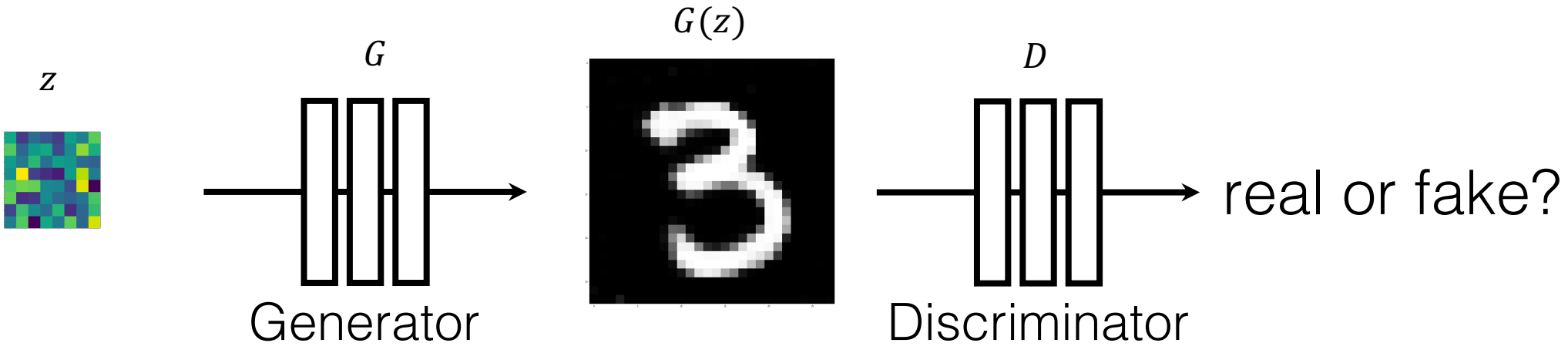
- **Learning as a two-player game**
  - *Discriminator* D_ϕ: Addestrato a discriminare tra dati reali e dati generati
  - *Generator* G_θ : Addestrato a generare esempi realistici per confondere il discriminatore

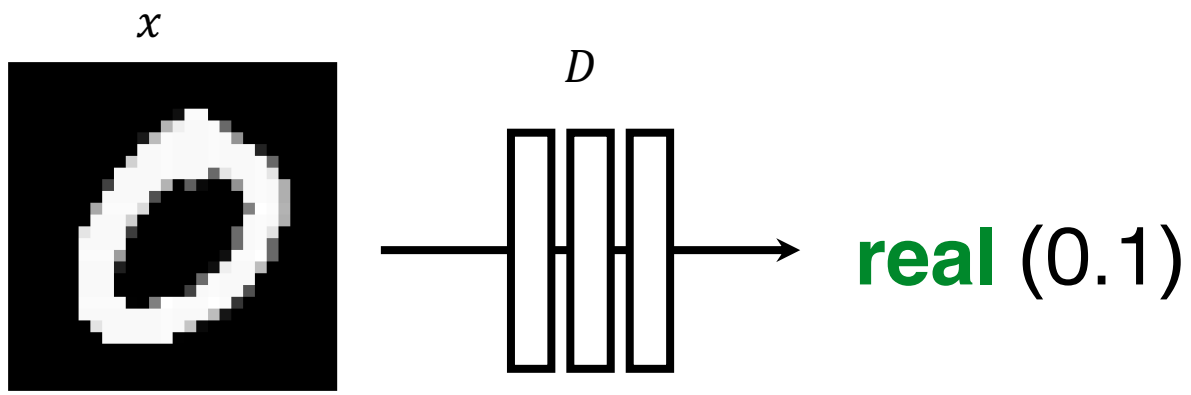

z
G
$G(z)$
Generator



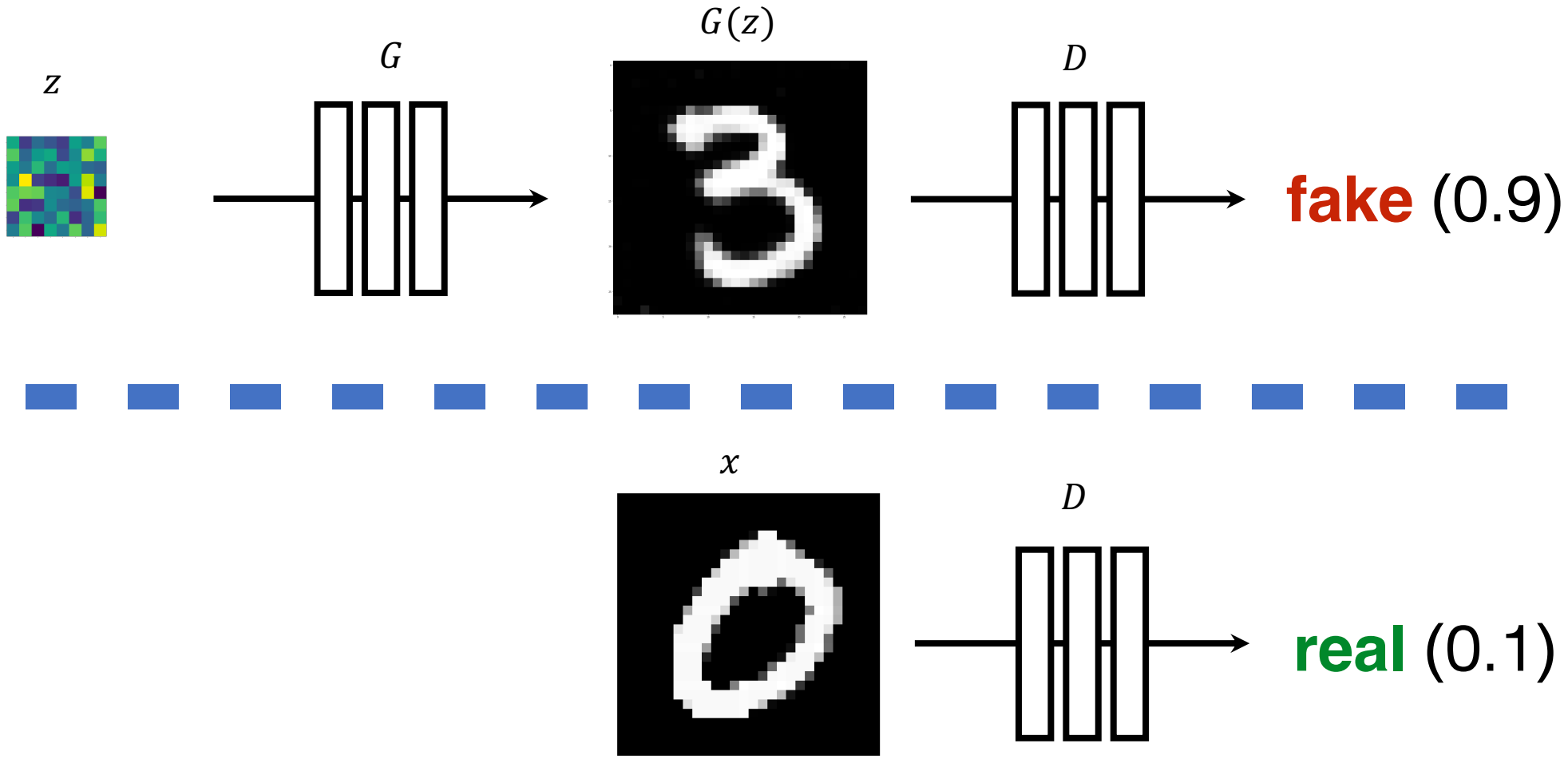
G sintetizza immagini che confondono **D**

D identifica le fakes

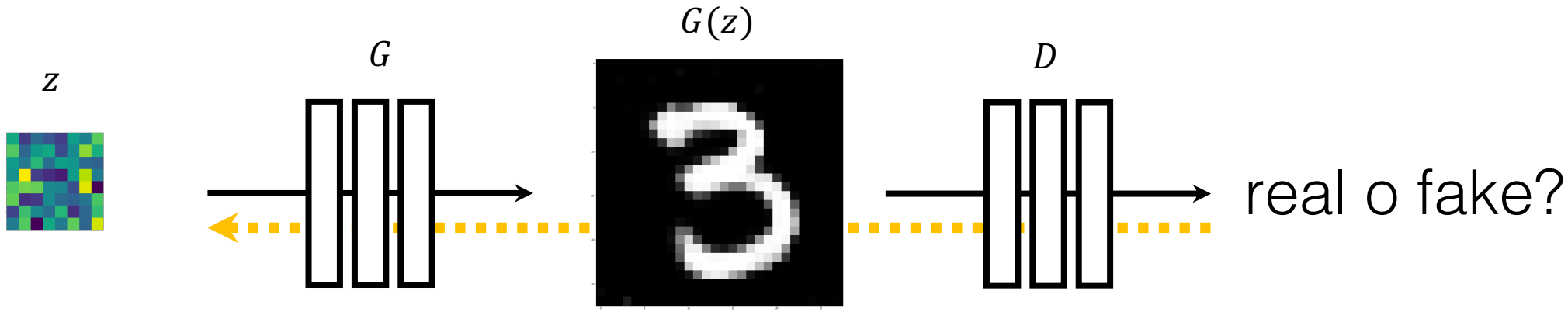




$$argmax_D \quad \mathbb{E}_x[\log D(x)]$$

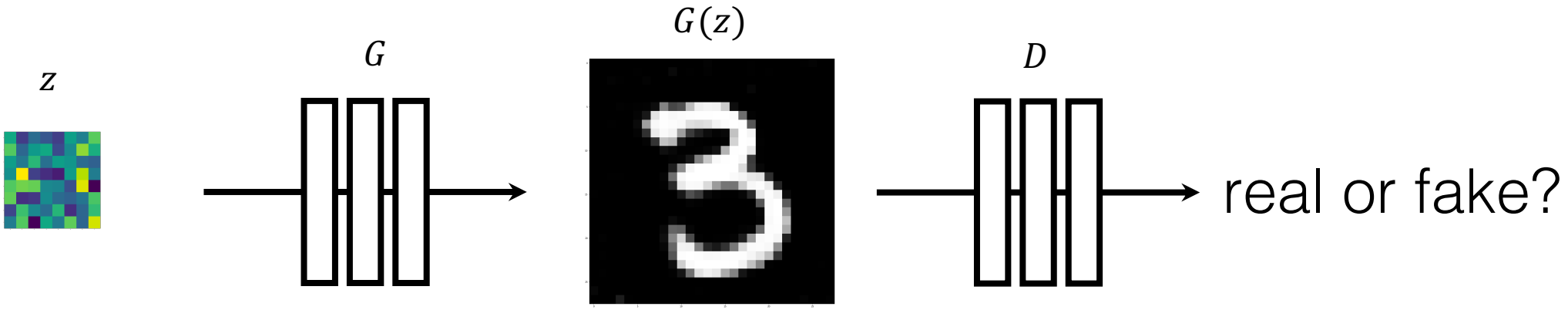


$$argmax_D\ argmin_G \mathbb{E}_x[\log D(x) + \log(1 - D(G(z))\ ]$$



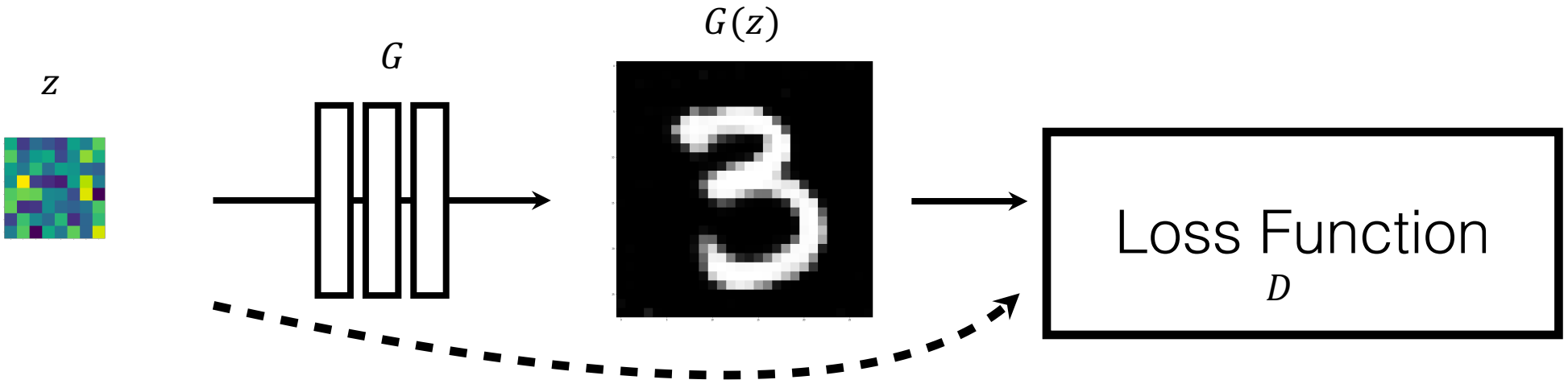
G sintetizza le immaginin che ***confondono*** **D**:

$$argmax_D \boxed{argmin_G} \mathbb{E}_x[\log D(x) + \log(1 - D(G(z))\ ]$$



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

$$\boxed{argmax_D}\ \boxed{argmin_G}\ \mathbb{E}_x[\log D(x) + \log(1 - D(G(z))\ ]$$



Per G, **D** è una funzione di loss.

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

Perché è meglio della ML?

- Discriminator Loss:

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

- Generator loss

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

- Adversarial Game

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

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'integrare rispetto a $D(x)$:

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

Perché è meglio della ML?

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

$$= \int p_r(x) \log \frac{P_r(x)}{P_r(x)+P_\theta(x)} + p_\theta(x) \log \frac{P_\theta(x)}{P_r(x)+P_\theta(x)}\, dx$$

$$= KL\left[\mathbb{P}_\mathrm{r} | \frac{\mathbb{P}_\theta+\mathbb{P}_\mathrm{r}}{2}\right] + KL\left[\mathbb{P}_\theta | \frac{\mathbb{P}_\theta+\mathbb{P}_\mathrm{r}}{2}\right] - 2\log 2$$

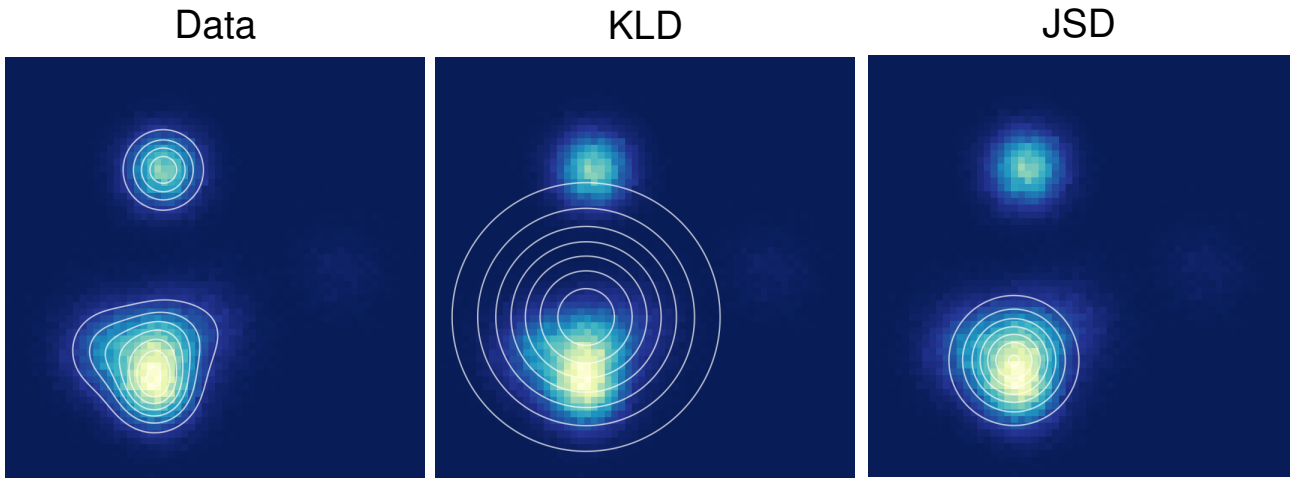
$$= 2JS[\mathbb{P}_\mathrm{r} | \mathbb{P}_\theta] - 2\log 2$$

Perché è meglio della ML?

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

- Minimizzare su θ 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 at al 2016]

Schema GAN

Algorithm 1 Inference algorithm.

Initialize ϕ and θ
for number of epochs **do**
  for k steps **do**
    Sample $\left\{\tilde{x}_\theta^{(1)}, \ldots, \tilde{x}_\theta^{(m)}\right\}$ from $\mathbb{P}_\theta$;
    Sample $\left\{x^{(1)}, \ldots, x^{(1)}\right\}$ from $\mathbb{P}_r$;
    Update ϕ by ascending its stochastic gradient:

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

  end for
  Sample $\left\{\tilde{x}_\theta^{(1)}, \ldots, \tilde{x}_\theta^{(m)}\right\}$ from $\mathbb{P}_\theta$;
  Update θ by descending its stochastic gradient:

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

end for
Return ϕ and θ.

Schema GAN

- Critico: Backpropagation dagli esempi

Algorithm 1 Inference algorithm.

Initialize ϕ and θ
for number of epochs **do**
  for k steps **do**
    Sample $\left\{\tilde{x}_\theta^{(1)}, \ldots, \tilde{x}_\theta^{(m)}\right\}$ from $\mathbb{P}_\theta$;
    Sample $\left\{x^{(1)}, \ldots, x^{(1)}\right\}$ from $\mathbb{P}_r$;
    Update ϕ by ascending its stochastic gradient:

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

  end for
  Sample $\left\{\tilde{x}_\theta^{(1)}, \ldots, \tilde{x}_\theta^{(m)}\right\}$ from $\mathbb{P}_\theta$;
  Update θ by descending its stochastic gradient:

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

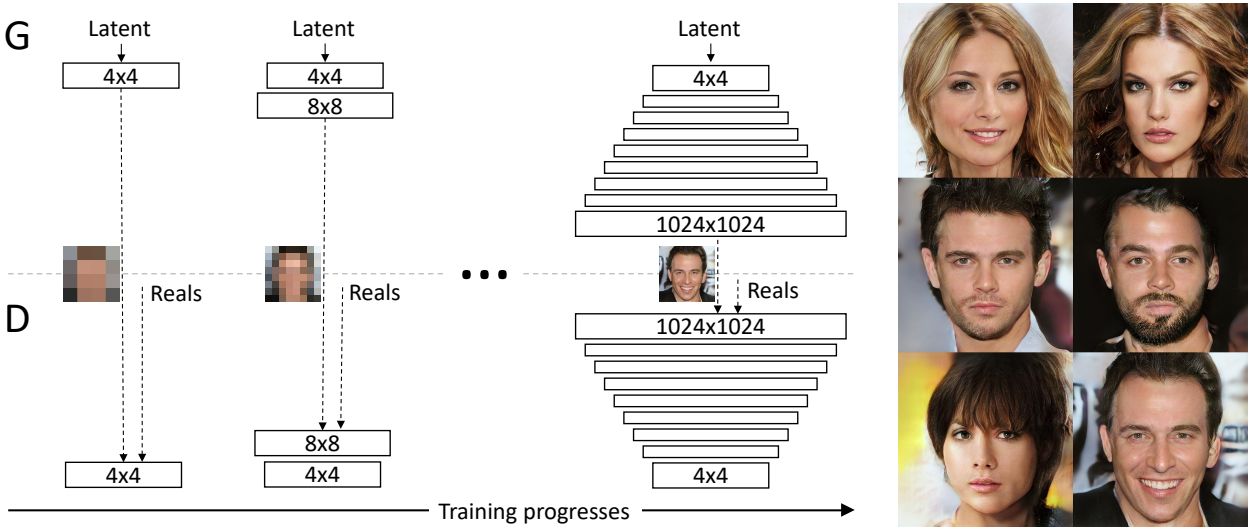
end for
Return ϕ and θ.

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

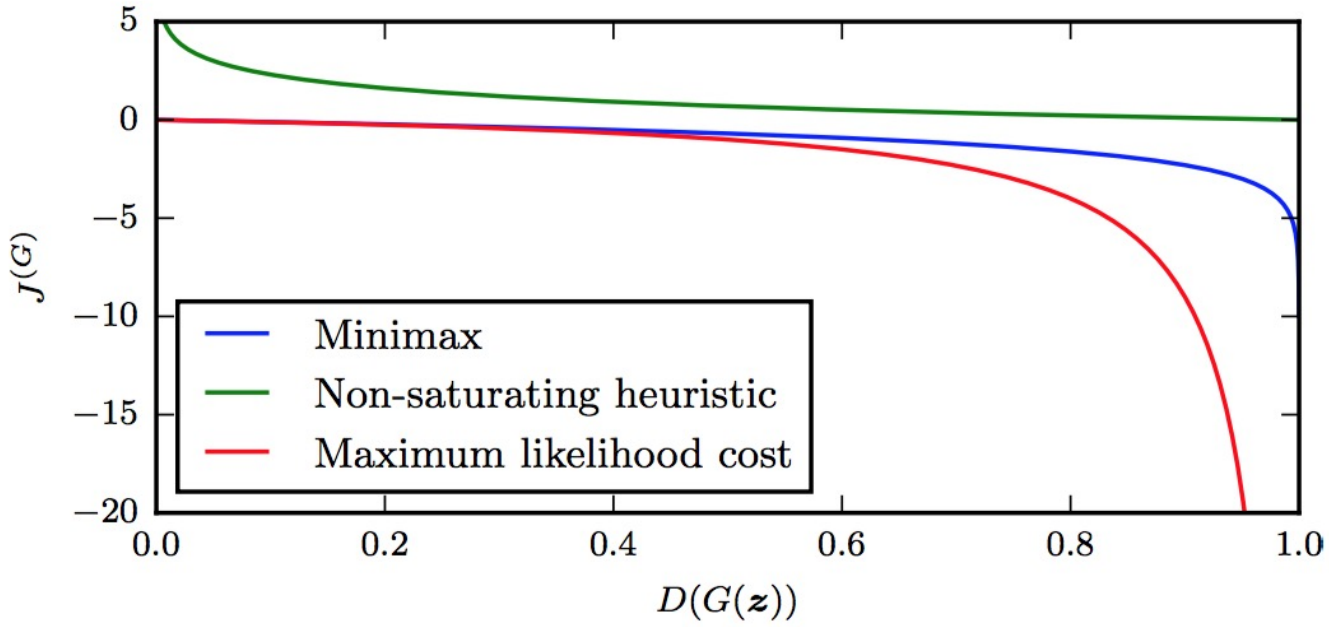
- Min-Max Gaming è instabile

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

- Non-Saturating GAN

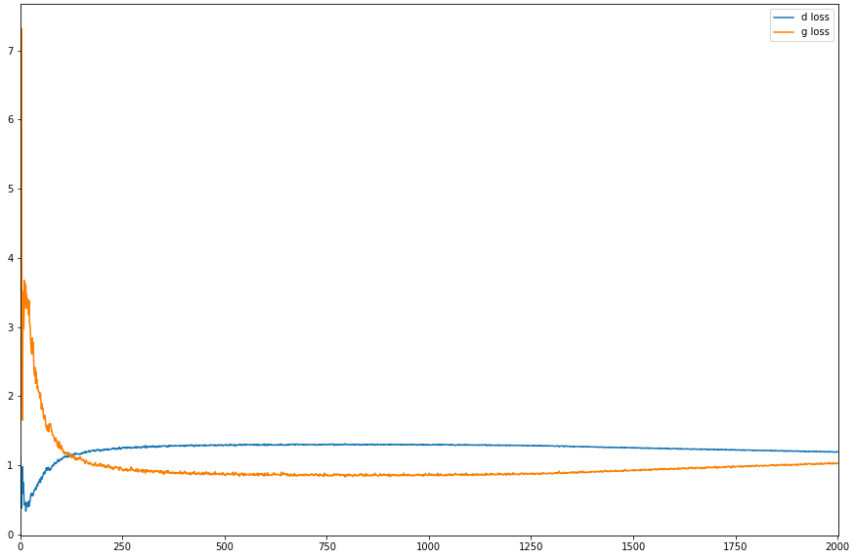
$$L_G(\phi,\theta) = \mathbb{E}_{x\sim\mathbb{P}_\theta}\left[\log\left(D_\phi\right)\right]$$

Game setup


5
0
−5
−10
−15
−20
$J^{(G)}$
0.0
0.2
0.4
0.6
0.8
1.0
$D(G(\boldsymbol{z}))$
Minimax
Non-saturating heuristic
Maximum likelihood cost

Instabilità

- Quando fermare il training?



- Wassersteing GAN
  - Sostituisce la JS con EM distance

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

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

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

Hacks

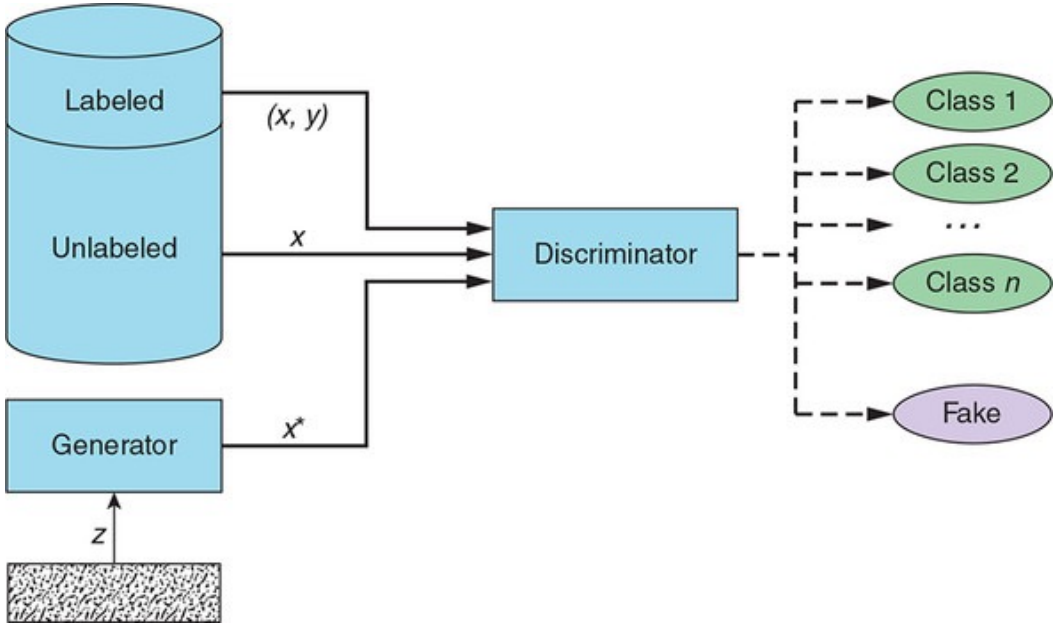
- Normalizzazione degli input
- Batch Normalization
- Gradient penalties
- Variare (aumentare) le iterazioni sul discriminatore
- Soft/noisy labels sul discriminator
- Evitare gradienti sparsi
  - No ReLU/MaxPool

Sviluppi

- Semi-Supervised GAN
- Conditional GAN
- CycleGAN

Semi-supervised GAN

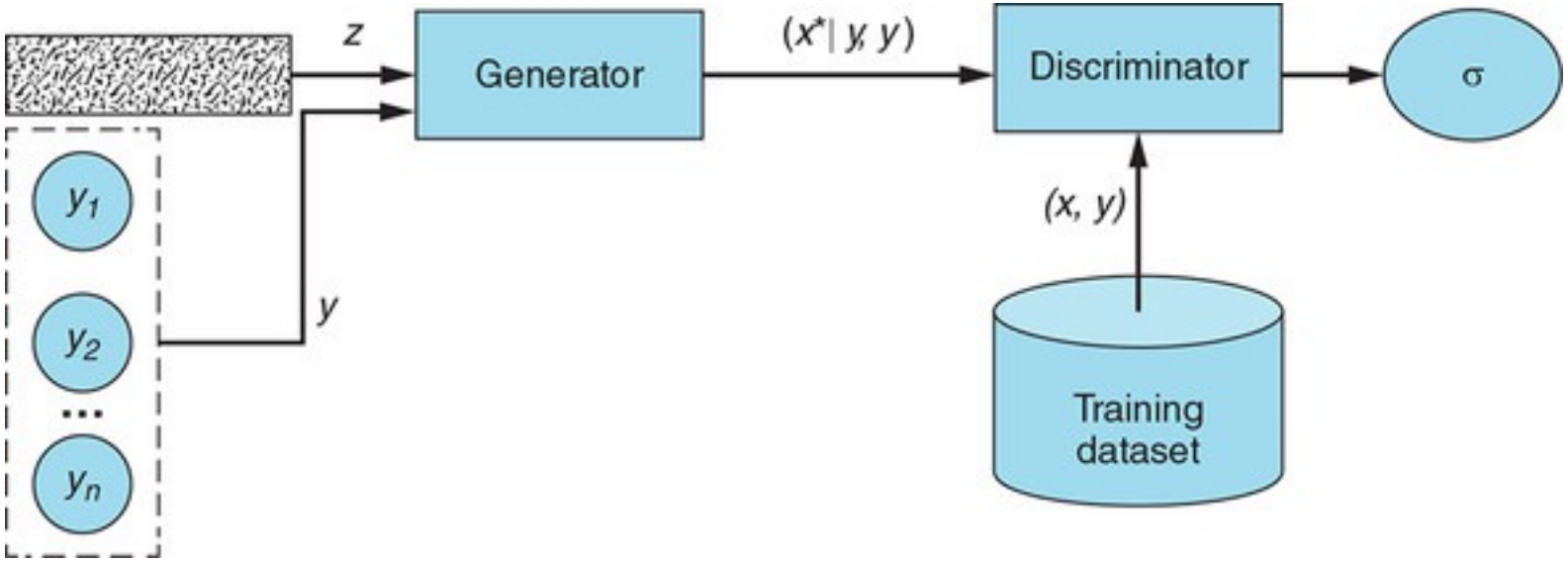
- Idea: utilizziamo il generator per irrobustire un discriminatore



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

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

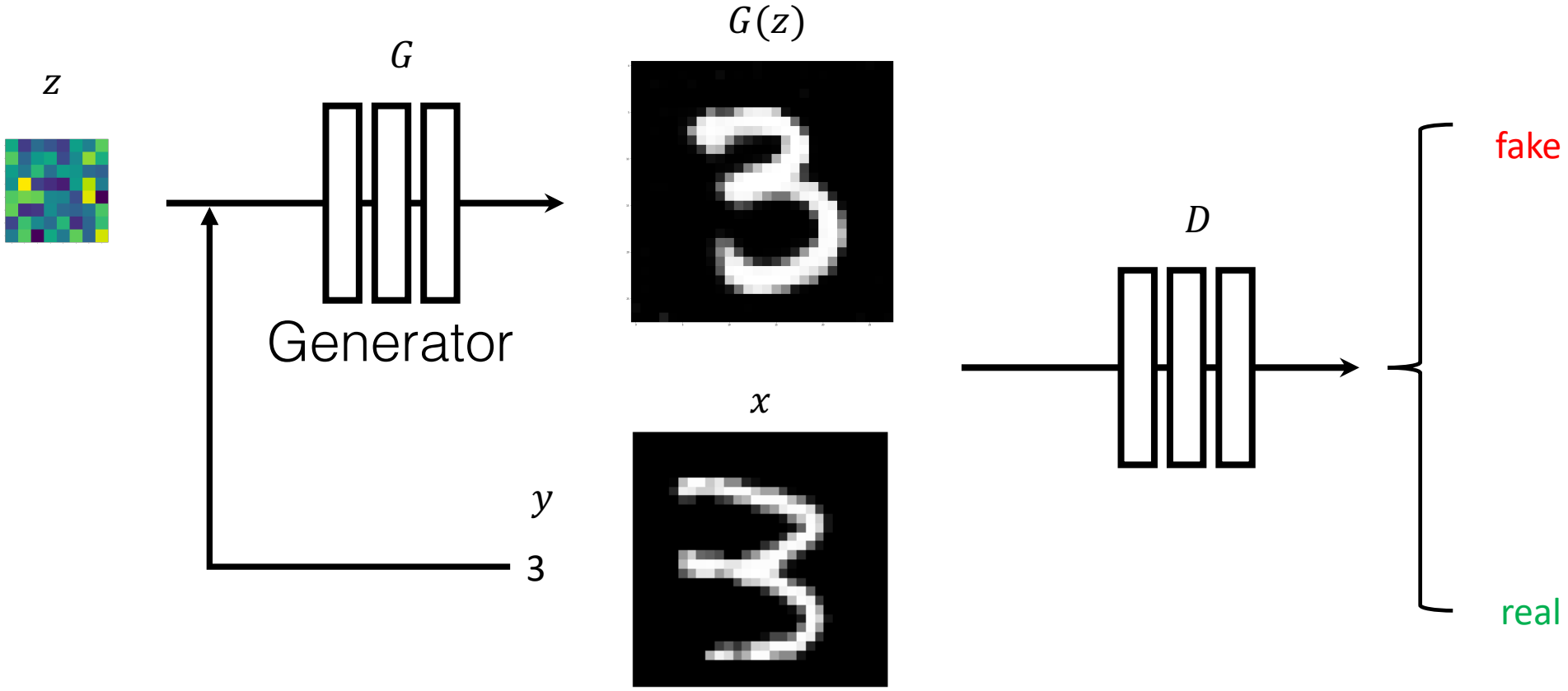
Conditional GAN



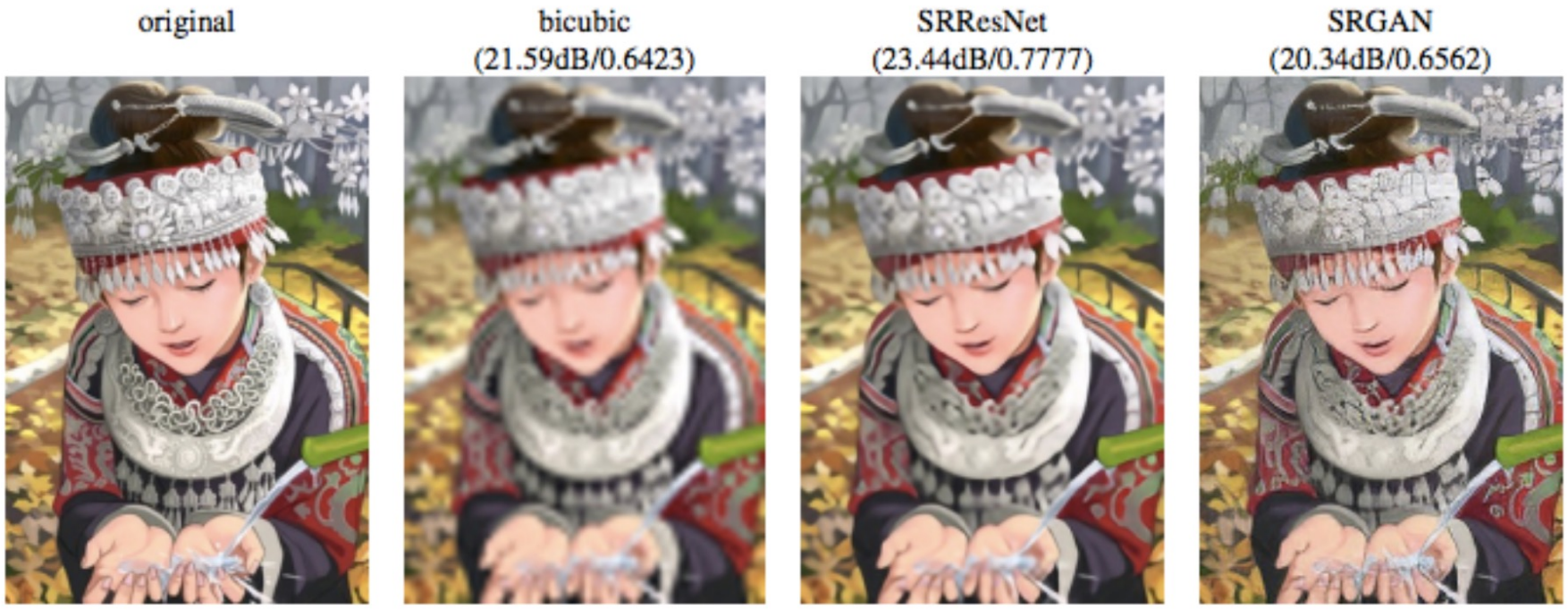
$$L_D(\phi,\theta) = \mathbb{E}_{x,y\sim\mathbb{P}_r}\left[\log D_\phi(y,x)\right] + \mathbb{E}_{x\sim\mathbb{P}_\theta(y)}\left[\log\left(1-\sum_y D_\phi(y,x)\right)\right]$$

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

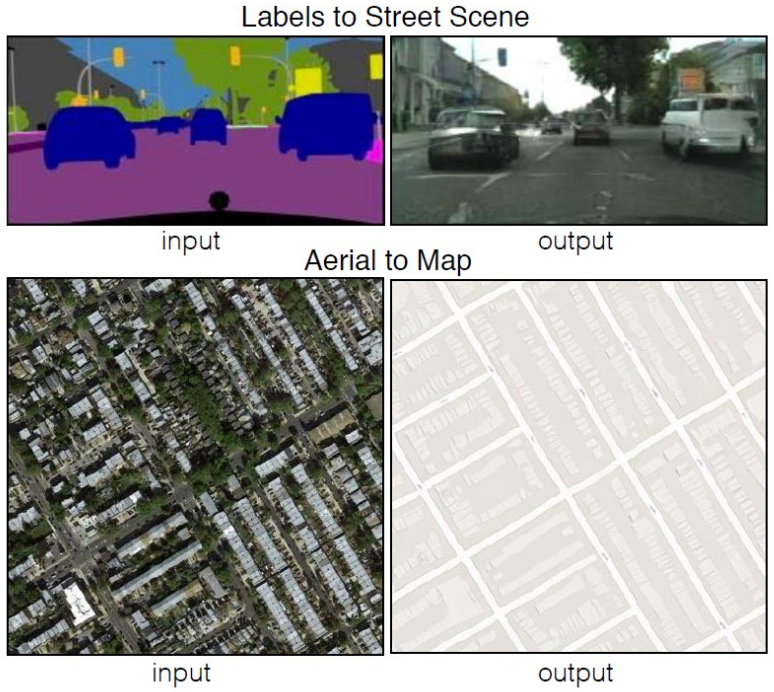
Supervised GAN-MNIST

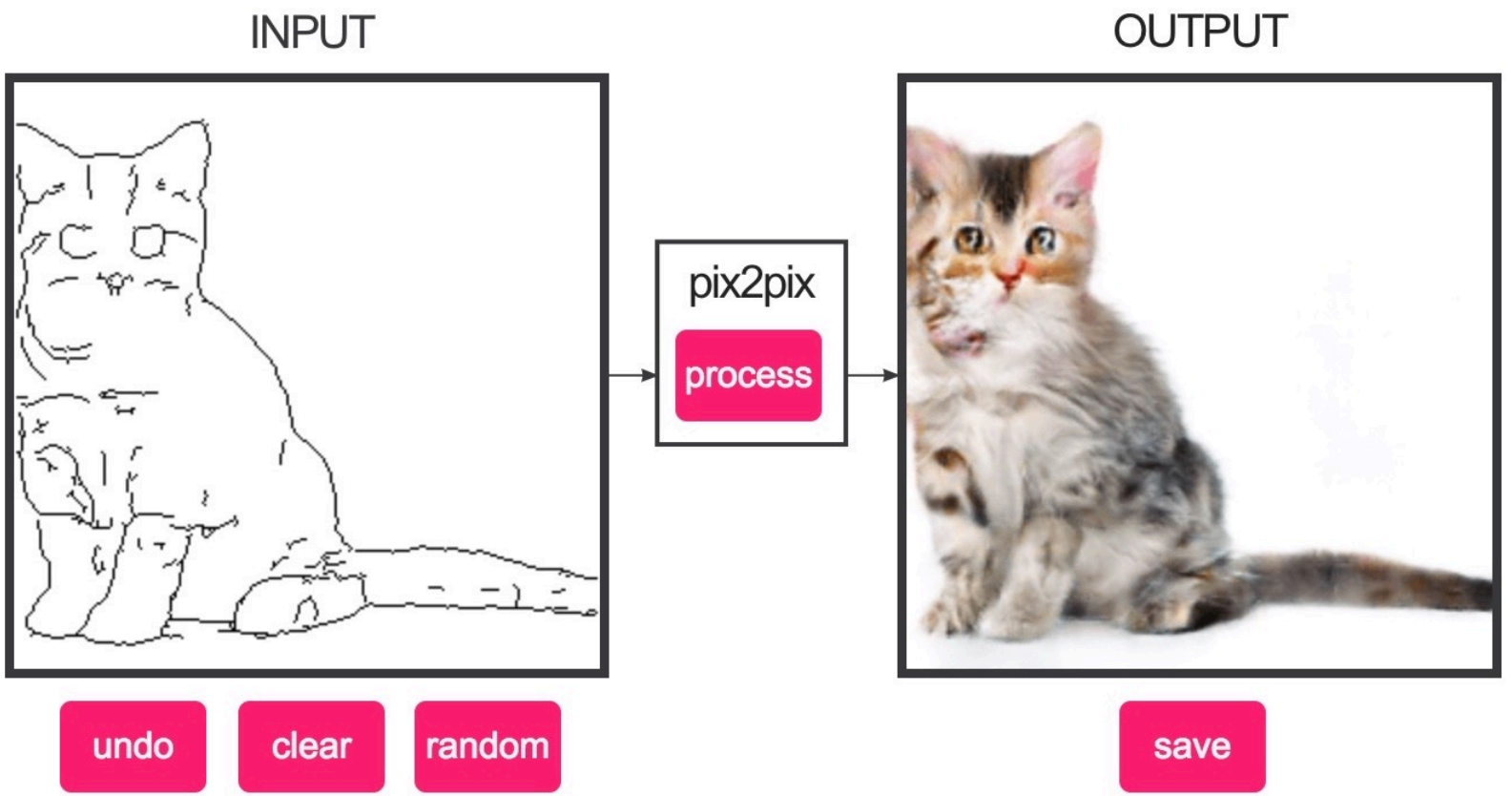


Esempi: Super Resolution



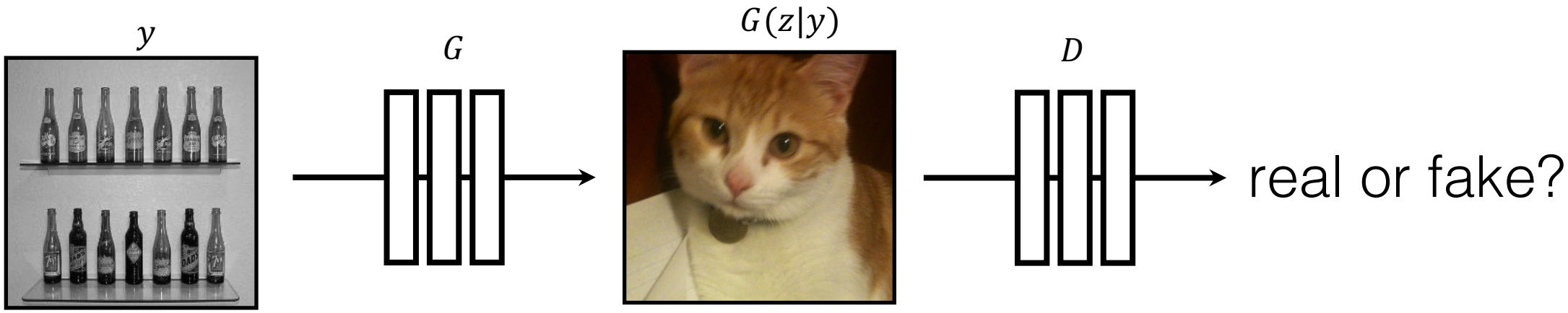
Esempi: Image to Image Transalation



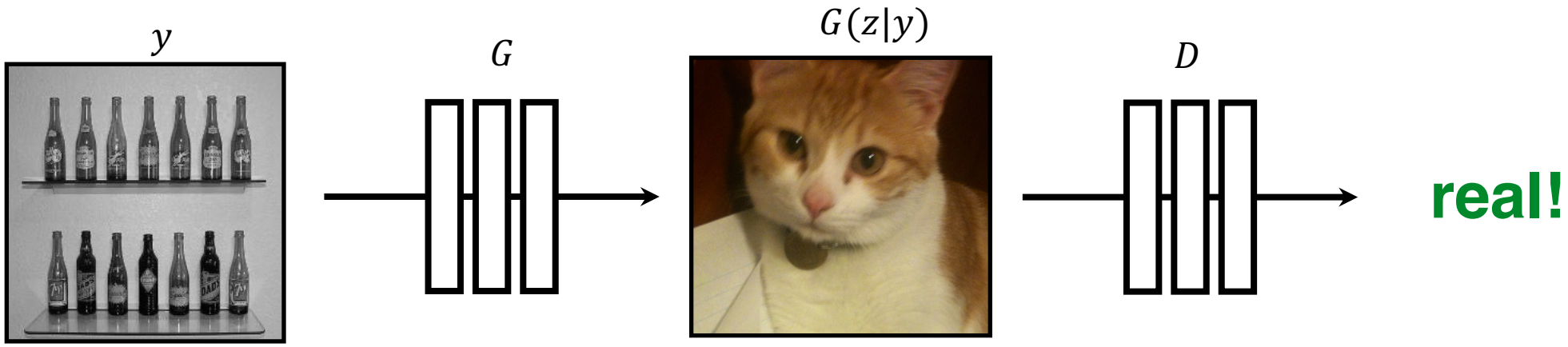


https://affinelayer.com/pixsrv/

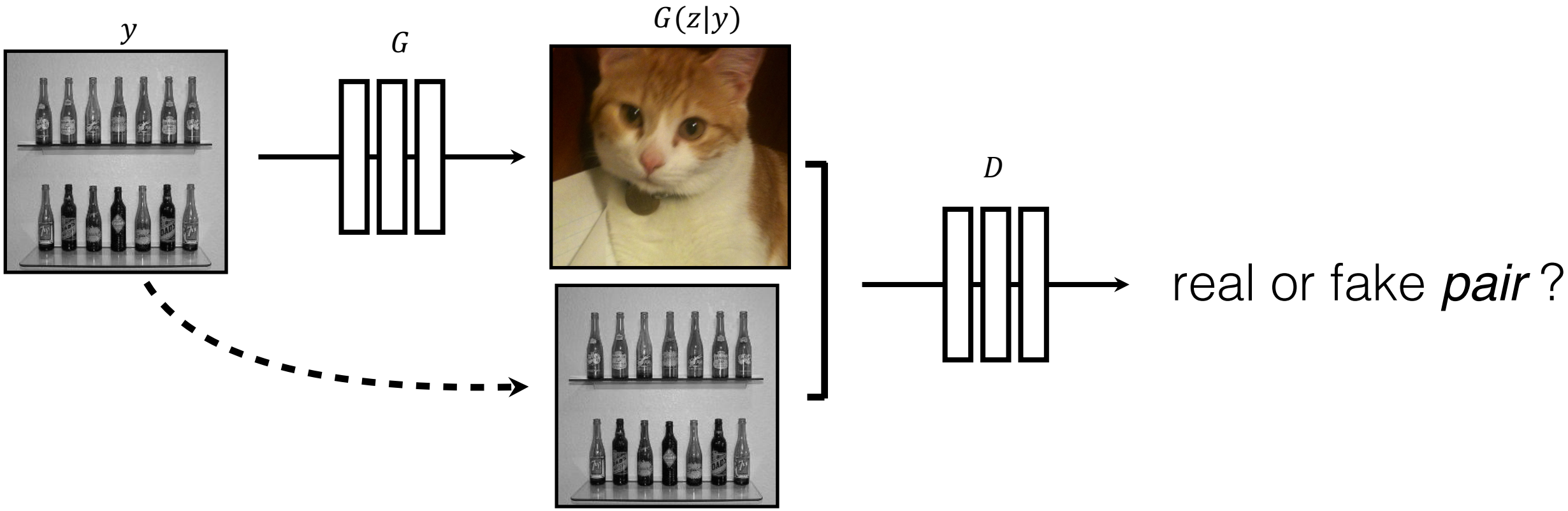
Esempio: Colorization



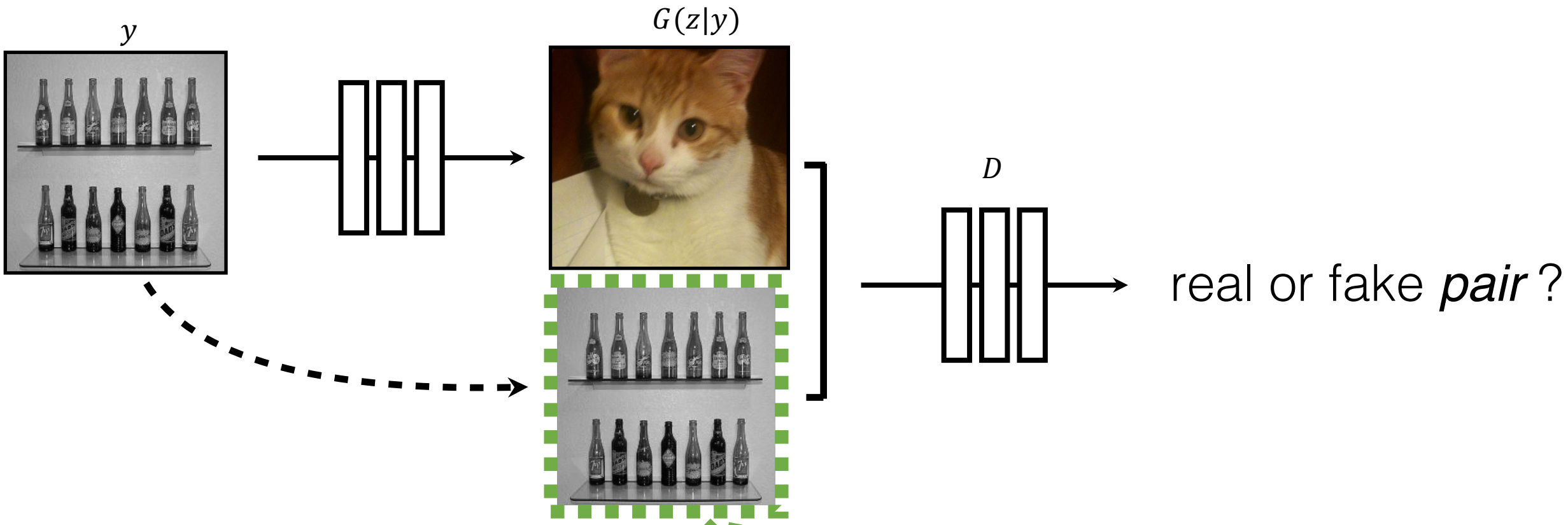
$$argmax_D\ argmin_G \mathbb{E}_{x,y}[\log D(x,y) + \log(1 - D(G(x|y),y)\ ]$$



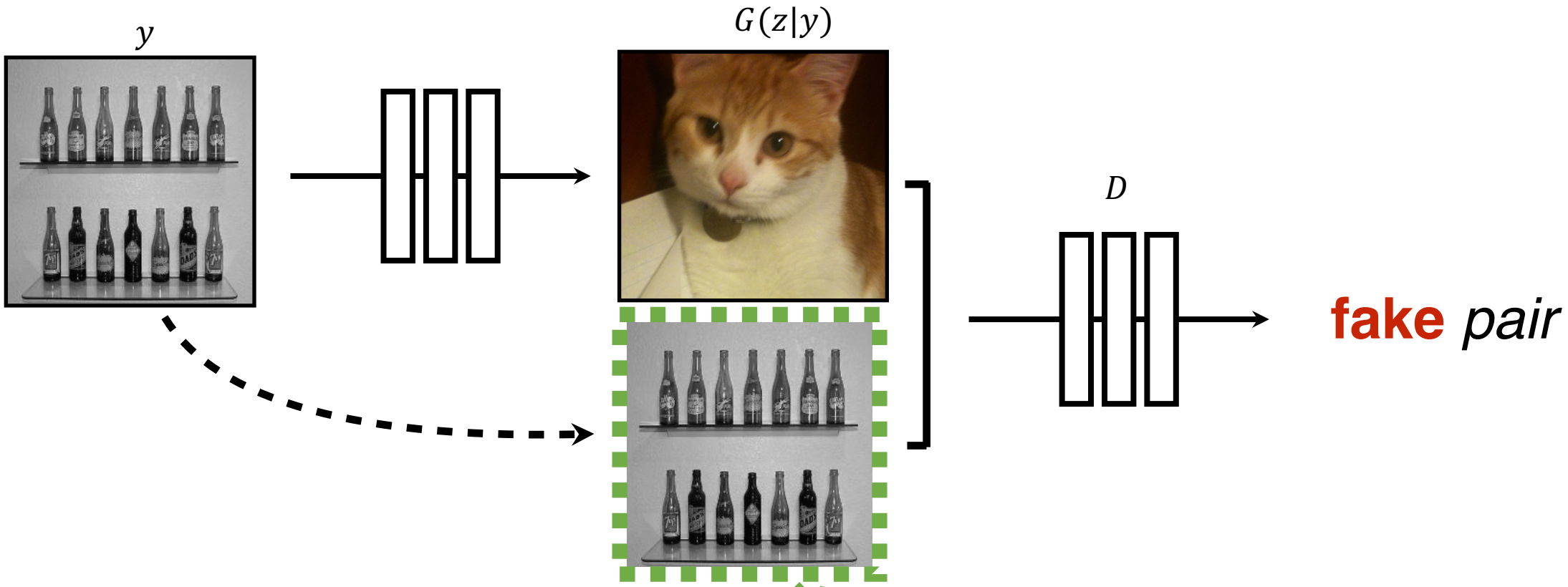
$$argmax_D\ argmin_G \mathbb{E}_{x,y}[\log D(x) + \log(1 - D(G(z|y))]$$



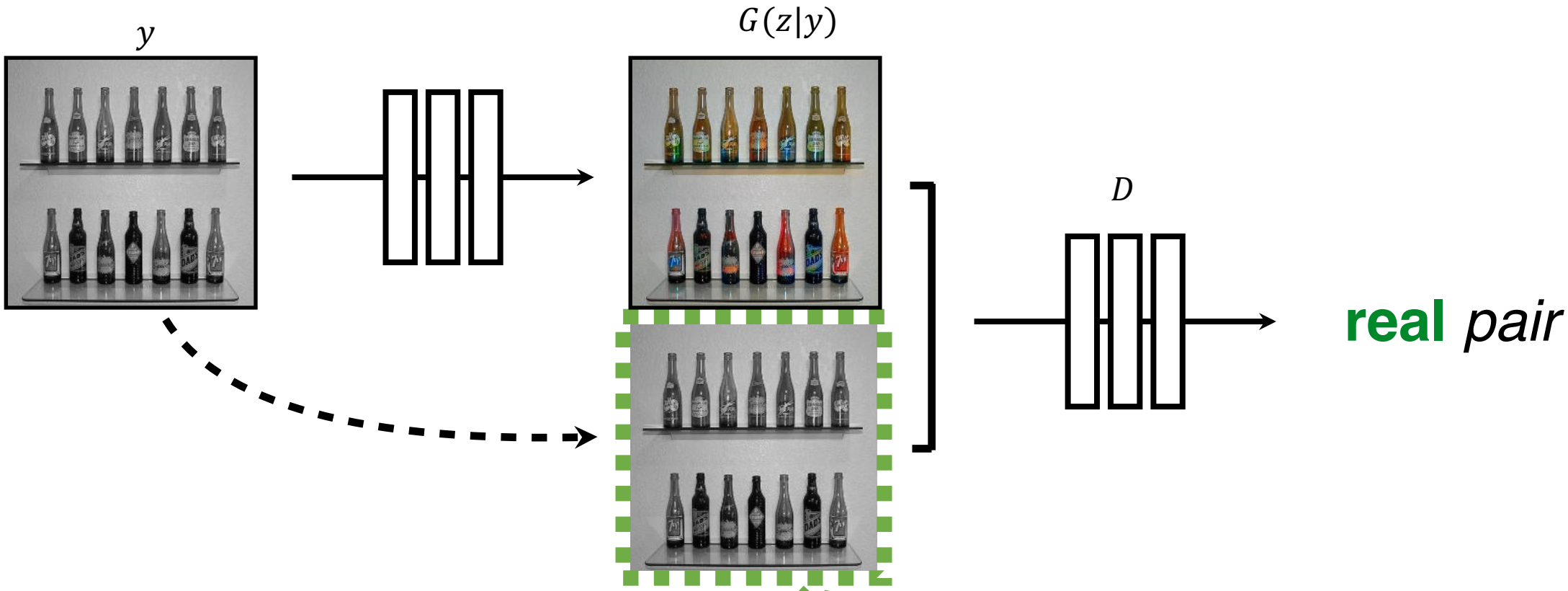
$$argmax_D\ argmin_G \mathbb{E}_{x,y}[\log D(x,y) + \log(1 - D(G(z|y),y))]$$



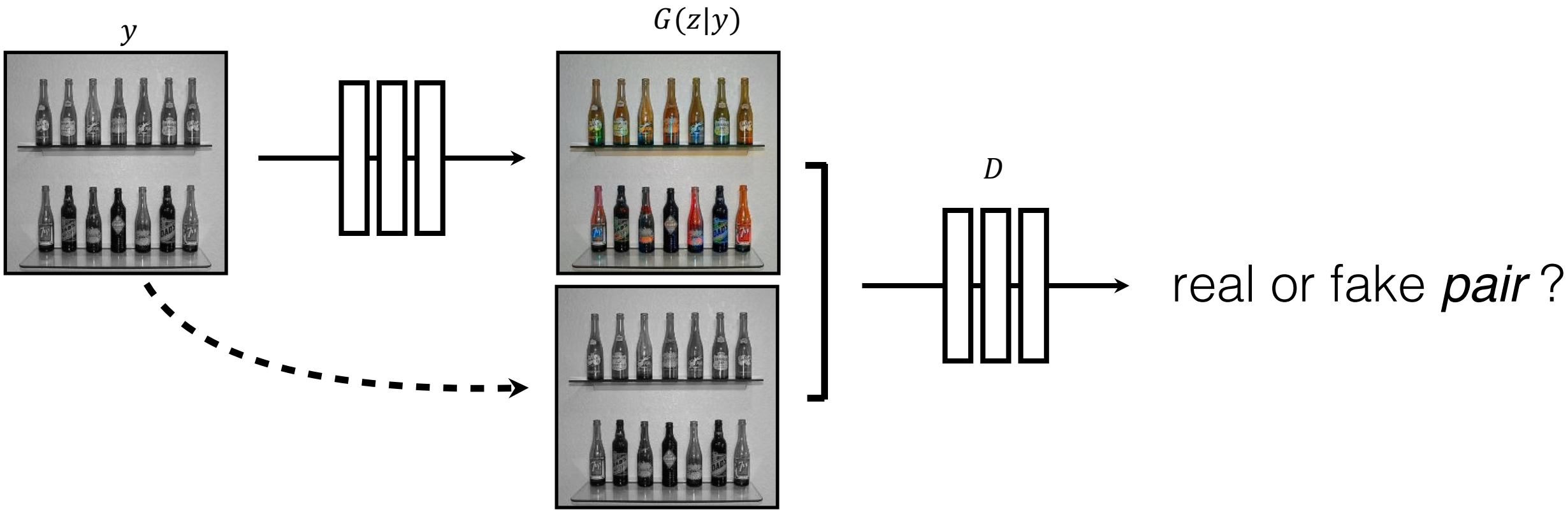
$$argmax_D\ argmin_G \mathbb{E}_{x,y}[\log D(x, y) + \log(1 - D(G(z|y), y))]$$



$$argmax_D\ argmin_G \mathbb{E}_{x,y}[\log D(x, y) + \log(1 - D(G(z|y), y))]$$



$$argmax_D\ argmin_G \mathbb{E}_{x,y}[\log D(x, y) + \log(1 - D(G(z|y), y))]$$



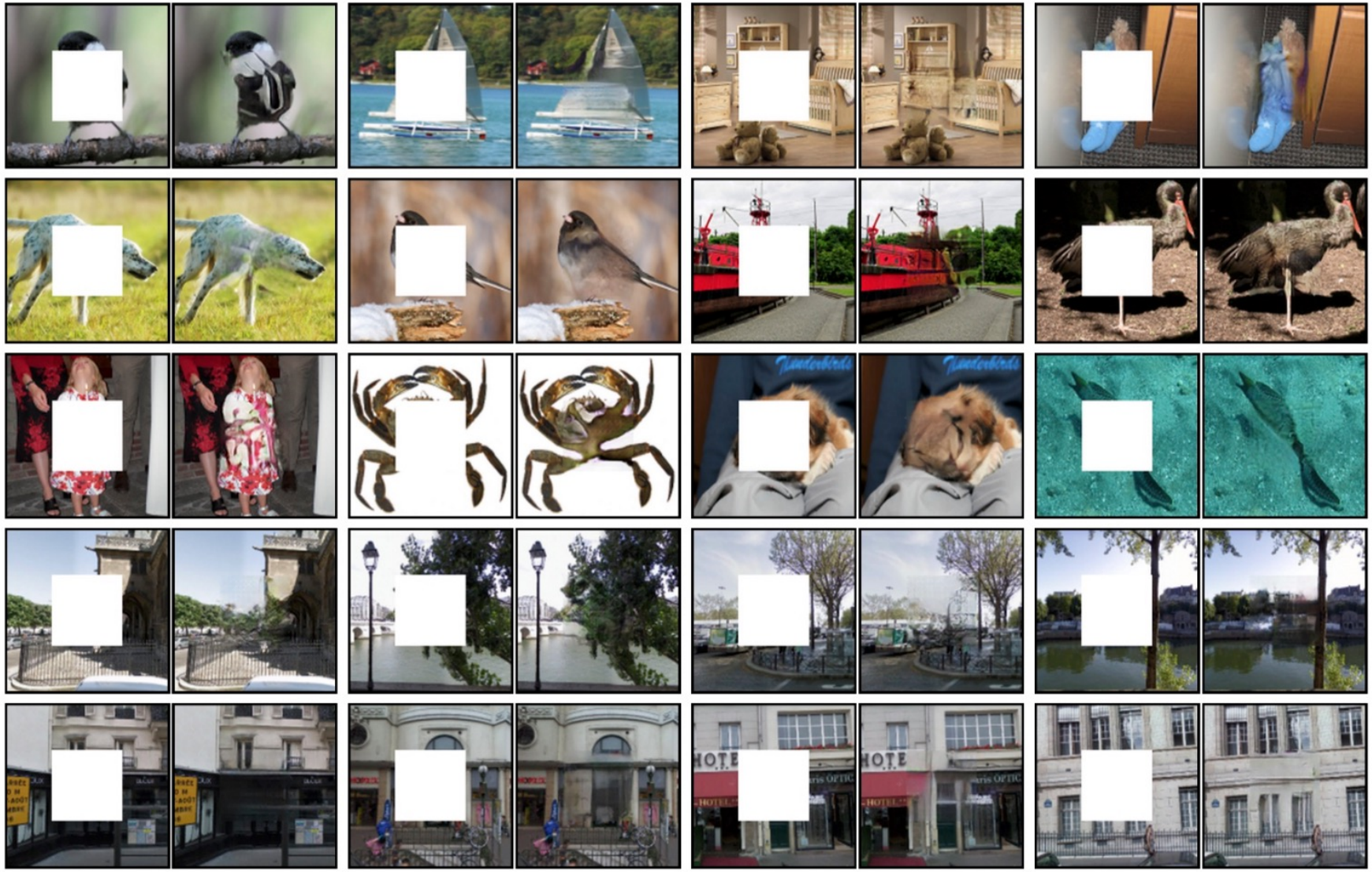
$$argmax_D\ argmin_G \mathbb{E}_{x,y}[\log D(x,y) + \log(1 - D(G(z|y),y))]$$

Qual è il ruolo del random noise?

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

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

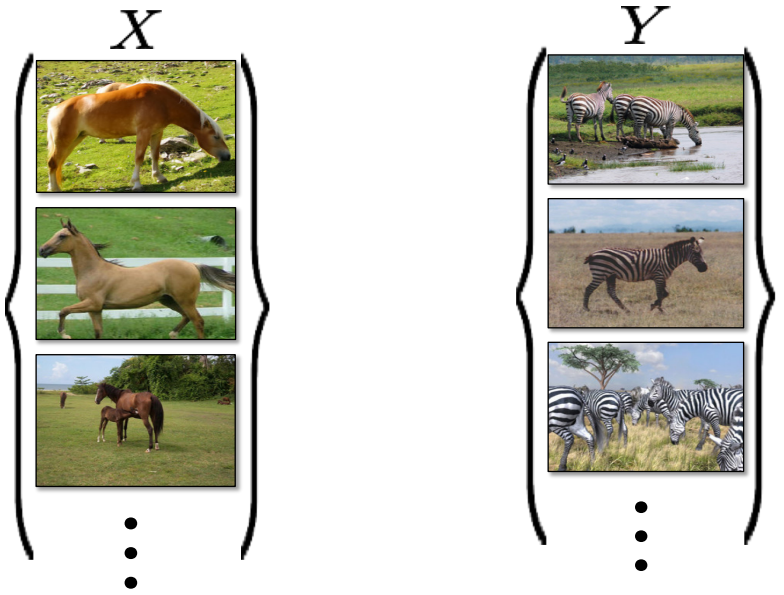
Image Inpainting



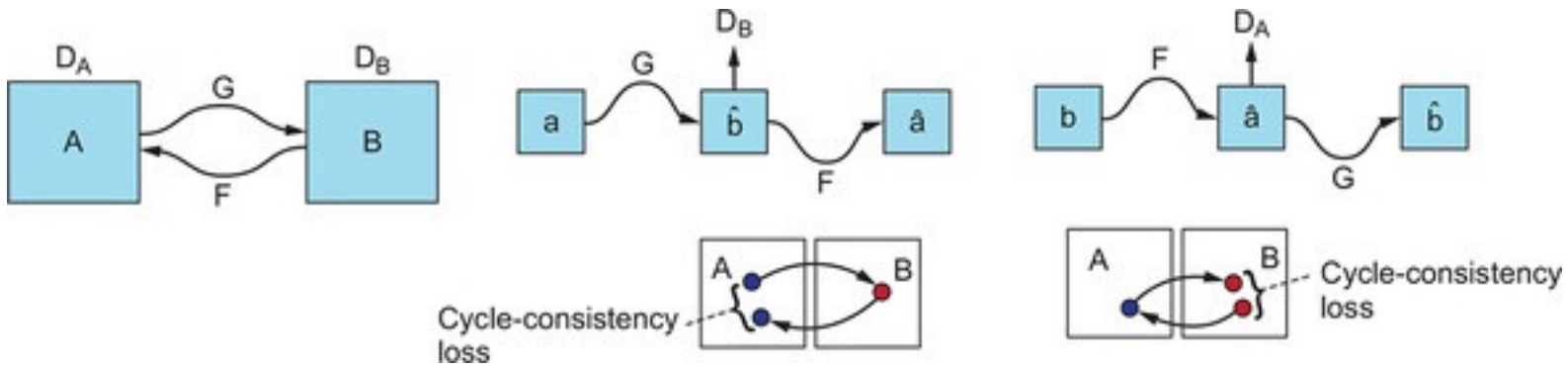
Unpaired image translation

- Tutti gli esempi precedenti si basano sul fatto di poter accoppiare l'input all'output desiderato

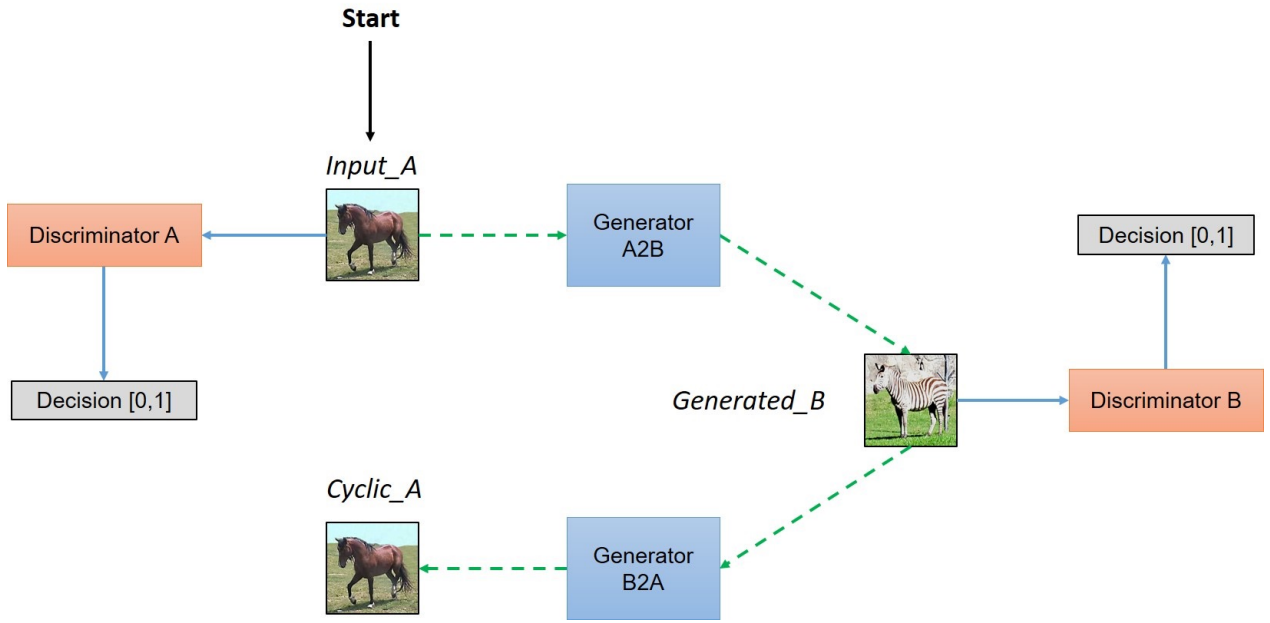
- Che succede se le coppie sono inconsistenti?

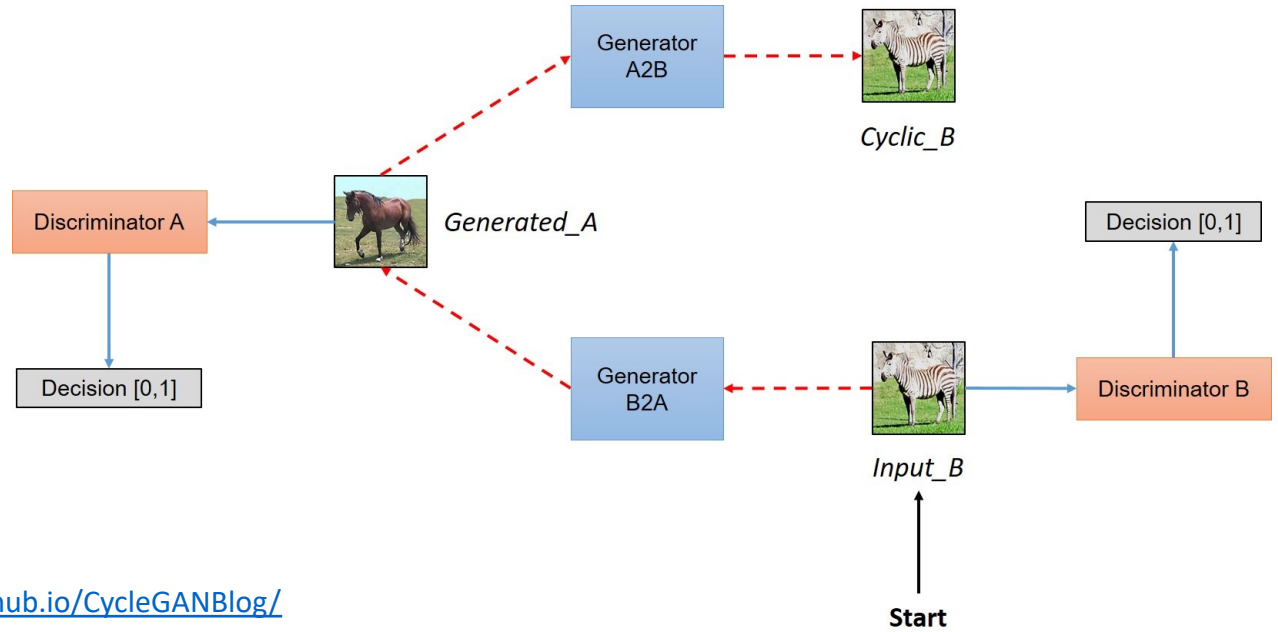


CycleGAN



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





https://hardikbansal.github.io/CycleGANBlog/

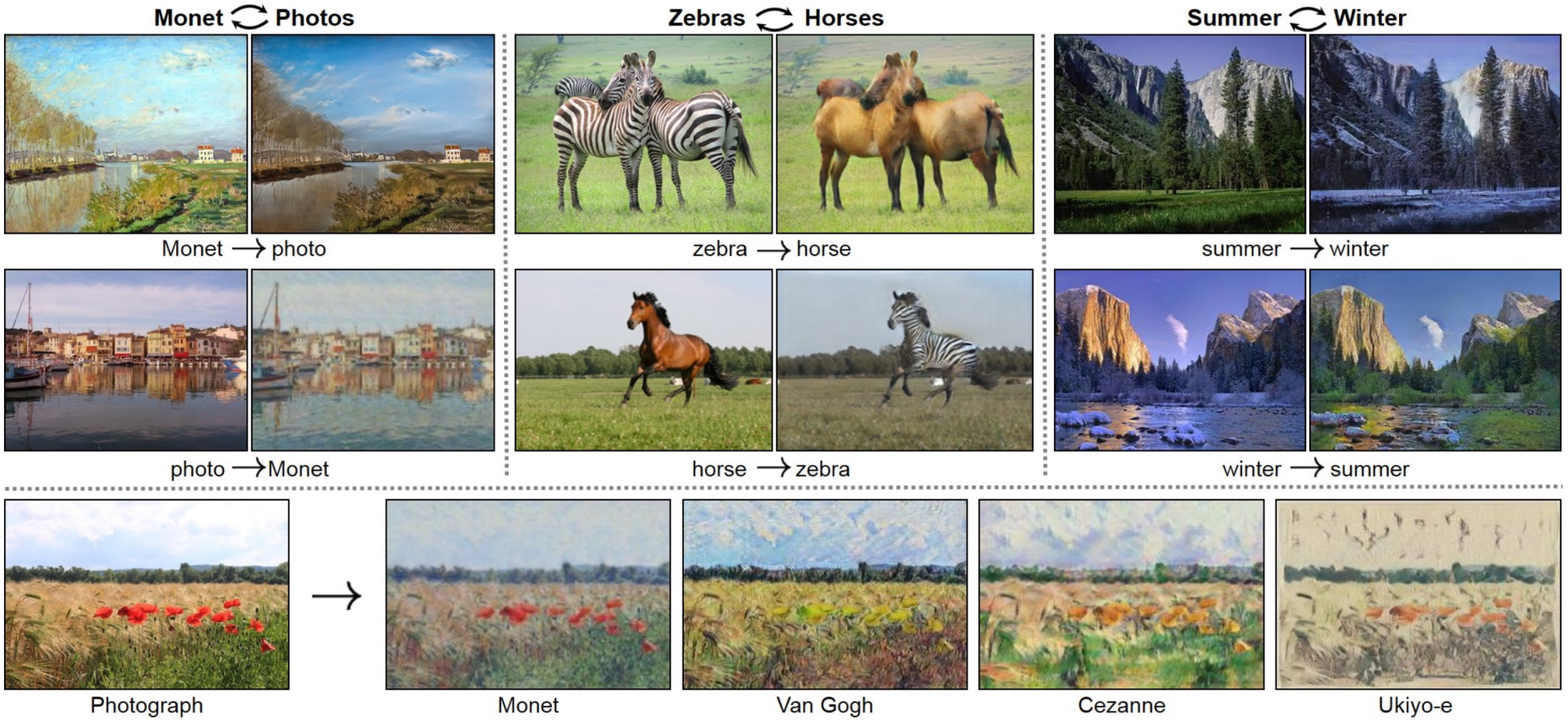
CycleGAN

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

- 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



https://junyanz.github.io/CycleGAN/