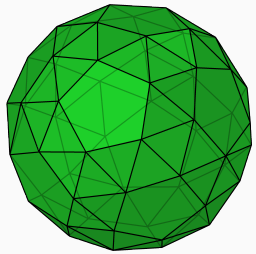
Mạng đối nghịch – tạo sinh



PiMA 2025: Toán học trong các Mô hình Tạo sinh

Khue Nguyen, École Polytechnique, Paris, France

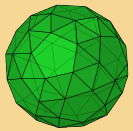
Quan Pham, Le Quy Don High School for the gifted, Da Nang, Vietnam

Anh Le, HUS High School for gifted students, Hanoi, Vietnam

Mentor: Duc Nguyen, FPT University, Hanoi, Vietnam

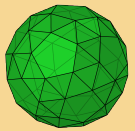
Ngày 19 tháng 8 năm 2025

Mục lục



Giới thiệu Mạng đối nghịch tạo sinh

1.1. GAN là gì và hoạt động thế nào?

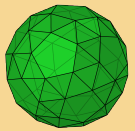


Mạng đối nghịch tạo sinh, hay GAN (Generative Adversarial Network), là một mô hình sinh được Ian Goodfellow và cộng sự đề xuất năm 2014.

Mô hình bao gồm hai thành phần chính:

- **Generator (G)**: tạo ra dữ liệu giống thật nhất có thể.
- **Discriminator (D)**: phân biệt đâu là dữ liệu sinh ra từ Generator và đâu là dữ liệu thật.

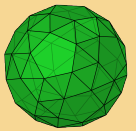
1.1. GAN là gì và hoạt động thế nào?



Ý tưởng của mô hình:

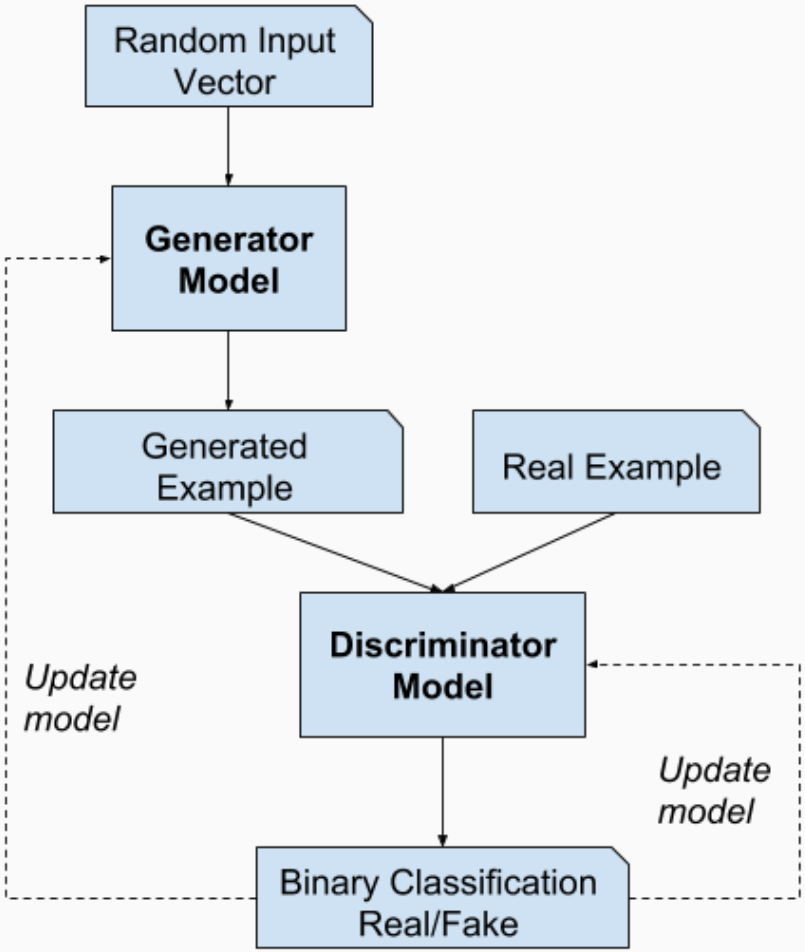
- Hai mạng này cạnh tranh với nhau
- Sự cạnh tranh thúc đẩy cả hai cải thiện phương pháp của mình cho đến khi Discriminator không còn phân biệt được dữ liệu do Generator tạo ra với dữ liệu thật được nữa.

1.1. GAN là gì và hoạt động thế nào?

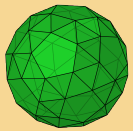


Generator (G):

- Đầu vào: vector ngẫu nhiên (noise vector z).
- Biến đổi noise ra thành một dữ liệu "giả".
- Dữ liệu "giả" được đưa vào *Discriminator*.
- Mục tiêu: khiến Discriminator tin rằng dữ liệu Generator sinh ra là thật.

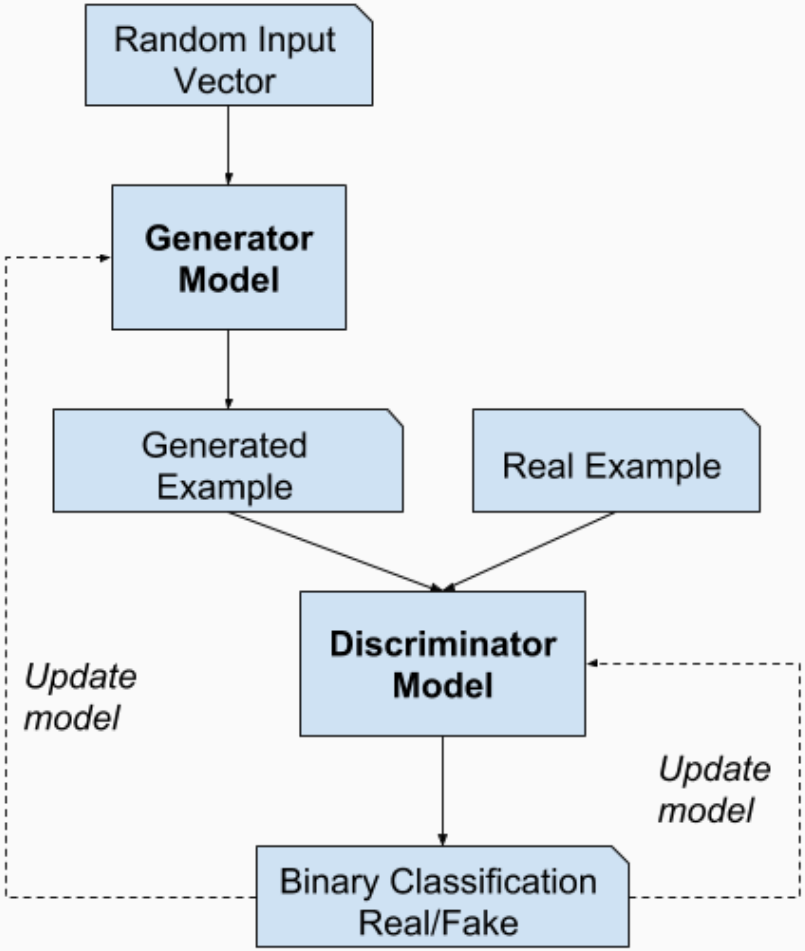


1.1. GAN là gì và hoạt động thế nào?

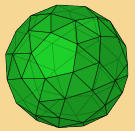


Discriminator (D):

- Đầu vào: dữ liệu (thật hoặc giả).
- Đầu ra: xác suất dự đoán, gần 1 nếu thật, gần 0 nếu giả.
- Mục tiêu: phân biệt chính xác dữ liệu thật và dữ liệu giả.



1.1. GAN là gì và hoạt động thế nào?

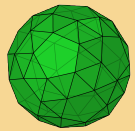


- **Hàm mục tiêu gốc** của GAN được viết như sau:

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

  $D(x)$: xác suất D dự đoán một mẫu x từ dữ liệu thật là *thật*.
  $D(G(z))$: xác suất D dự đoán mẫu $G(z)$ do G tạo ra là *thật*.
- **Mục tiêu của Discriminator:** cố gắng tối đa hóa giá trị tổng thể, tức là khiến cho $D(x) \to 1$ và $D(G(z)) \to 0$.
- **Mục tiêu của Generator:** làm cho xác suất $D(G(z)) \to 1$.

1.2. Một số ứng dụng và thành tựu của GAN



Một số ứng dụng của GAN:

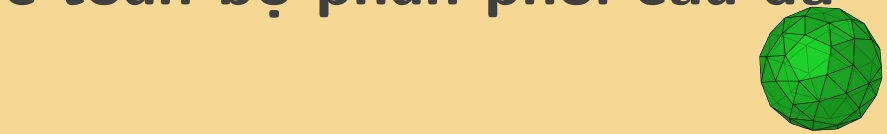
- **Hình ảnh**
  - Sinh ảnh mới (chân dung, tranh nghệ thuật).
  - Chỉnh sửa ảnh (image-to-image translation).
- **Video**
  - Deepfake video.
  - Tăng chất lượng video (super-resolution).
  - Sinh hoạt ảnh từ ảnh tĩnh.

Thành tựu nổi bật:

- Portrait of Edmond de Belamy của Obvious được đấu giá thành công tại Christie's với mức giá 432.500 USD vào 2018.
- RobotArt Competition (2016–2018), các tác phẩm GAN tham gia dự thi dc đánh giá không thua kém tranh do họa sĩ người vẽ.

Mối liên hệ giữa hàm mục tiêu của GAN và Jensen–Shannon divergence

2.1. Liệu GAN có thực sự học được toàn bộ phân phối của dữ liệu thật?



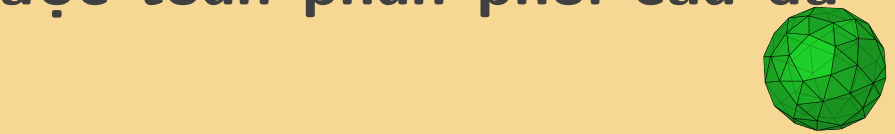
Liệu có gì đảm bảo rằng generator thực sự học được toàn bộ phân phối dữ liệu thật, chứ không chỉ vài mẫu giống bề ngoài?

Ví dụ: GAN có thể chỉ sinh ra một số ví dụ nhìn hợp lý (ví dụ chỉ sinh mèo trắng), nhưng không bao phủ phân phối dữ liệu (mọi loại mèo).

Vì vậy, ta cần một **thước đo xác suất** để khẳng định generator đang tiến gần đến phân phối dữ liệu thật.

Câu hỏi: "Nếu GAN hội tụ, thì nó đang thực sự tối ưu cái gì? Và thước đo nào đảm bảo nó học phân phối?"

2.2. Làm thế nào mà GAN học được toàn phân phối của dữ liệu thật?



Để trả lời câu hỏi này, Chính ở đây Goodfellow vào năm 2014 chứng minh:

"Với discriminator tối ưu, hàm mục tiêu GAN trở thành **Jensen–Shannon divergence** giữa p_{data} và p_g."

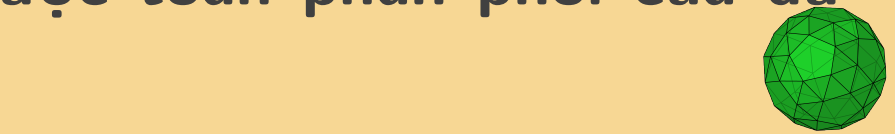
Chính mối liên hệ này mới cho thấy GAN thật sự có khả năng học phân phối dữ liệu, thay vì chỉ là một trò "đánh lừa nhau" không có cơ sở.

Ta cần chứng minh cụ thể về mối liên hệ giữa chúng:

Hàm mục tiêu của GAN:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}\big[\log\left(D(x)\right)\big] + \mathbb{E}_{z\sim p_z(z)}\big[\log\left(1-D(G(z))\right)\big].$$

2.2. Làm thế nào mà GAN học được toàn phân phối của dữ liệu thật?



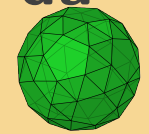
Để giải quyết bài toán này ta thường phải giải quyết từng phần. Đầu tiên ta đi tìm D tối ưu khi cố định G và đó là **Hàm tối ưu Discriminator(Opimal discriminator)**.

Khi cố định G, với mỗi x ta có thể tối đa hóa hàm $p_{\text{data}}(x) \log D(x) + p_g(x) \log (1 - D(x))$ với D(x).

Lấy đạo hàm và cho bằng 0 cho ta nghiệm đóng là **hàm tối ưu Discriminator**:

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}.$$

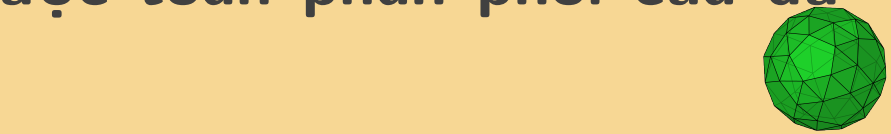
2.2. Làm thế nào mà GAN học được toàn phân phối của dữ liệu thật?



Tiếp đến ta sẽ thay **hàm tối ưu Discriminator** vào **hàm mục tiêu của GAN**:

$$C(G) := V(D^*, G) = \mathbb{E}_{x \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)} \right] + \mathbb{E}_{x \sim p_g} \left[\log \frac{p_g(x)}{p_{\text{data}}(x) + p_g(x)} \right].$$

2.2. Làm thế nào mà GAN học được toàn phân phối của dữ liệu thật?

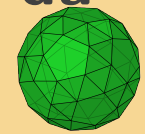


Mặt khác **Jensen–Shannon Divergence** được định nghĩa là hàm tính khoảng cách giữa 2 phân phối được tính với công thức sau:

$$\mathrm{JSD}(p \parallel q) = \tfrac{1}{2}\, KL(p \parallel m) + \tfrac{1}{2}\, KL(q \parallel m).$$

* $KL(p \parallel m)$ là Kullback–Leibler divergence giữa p và m, dùng để đo mức độ khác biệt (thông tin mất mát) giữa 2 phân phối).

2.2. Làm thế nào mà GAN học được toàn phân phối của dữ liệu thật?



Thay các phân phối:

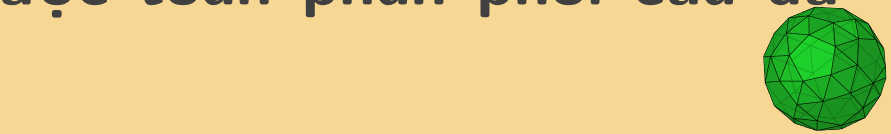
$$p = p_{\text{data}}(x), \quad q = p_g(x), \quad m = \tfrac{1}{2}(p_{\text{data}}(x) + p_g(x)).$$

Vào công thức **Jensen–Shannon Divergence**:

$$\text{JSD}(p \parallel q) = \tfrac{1}{2}\, KL(p \parallel m) + \tfrac{1}{2}\, KL(q \parallel m).$$

$$\Rightarrow \text{JSD}(p_{\text{data}} \parallel p_g) = \tfrac{1}{2}\, \mathbb{E}_{x \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(x)}{\frac{1}{2}(p_{\text{data}}(x) + p_g(x))} \right] + \tfrac{1}{2}\, \mathbb{E}_{x \sim p_g} \left[\log \frac{p_g(x)}{\frac{1}{2}(p_{\text{data}}(x) + p_g(x))} \right].$$

2.2. Làm thế nào mà GAN học được toàn phân phối của dữ liệu thật?



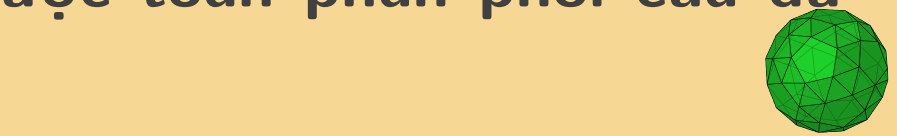
$$\Leftrightarrow \mathrm{JSD}(p_{\text{data}} \parallel p_g) = \tfrac{1}{2}\,\mathbb{E}_{x\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_g(x)}\right] + \tfrac{1}{2}\,\mathbb{E}_{x\sim p_g}\left[\log \frac{p_g(x)}{p_{\text{data}}(x)+p_g(x)}\right] + \log 2.$$

Kết hợp với **hàm mục tiêu của GAN** ở trên ta rút ra được công thức mối liên hệ giữa **hàm mục tiêu của GAN** và **Jensen–Shannon Divergence**:

$$C(G) = -\log 4 + 2\mathrm{JSD}(p_{\text{data}} \parallel p_g).$$

Như vậy, mối liên hệ này đã cho ta thấy Generator thật sự học được toàn bộ phân phối dữ liệu thật.

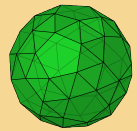
2.2. Làm thế nào mà GAN học được toàn phân phối của dữ liệu thật?



Đó là cách **hàm mục tiêu của GAN** trở thành **Jensen–Shannon Divergence**. Và cũng chính nhờ mối liên hệ này, ta giải thích được tại sao GAN hay bị mất mát gradient (gradient vanishing), mode collapse.

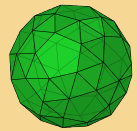
Vấn đề mất mát gradient của GAN (gradient vanishing)

3. Vấn đề mất mát gradient của GAN (gradient vanishing)



Vấn đề này xảy ra khi phân phối p_g và p_{data} là hoàn toàn khác nhau. Nói cách khác, Generator vẫn còn "yếu"và tạo ra các mẫu không giống với dữ liệu thực, khiến Discriminator dễ dàng phân biệt.

3. Vấn đề mất mát gradient của GAN (gradient vanishing)



Vậy ở hàm này:

$$\mathbb{E}_{x\sim p_{\text{data}}}\left[\log\frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_g(x)}\right] \qquad \mathbb{E}_{x\sim p_g}\left[\log\frac{p_g(x)}{p_{\text{data}}(x)+p_g(x)}\right]$$

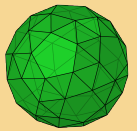
Ta sẽ có 2 hàm này sẽ tiến về 0.

Jensen–Shannon Divergence:

$$\begin{aligned}\mathrm{JSD}(p_{\text{data}}\parallel p_g) = {} & \tfrac{1}{2}\,\mathbb{E}_{x\sim p_{\text{data}}}\left[\log\frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_g(x)}\right] \\ & + \tfrac{1}{2}\,\mathbb{E}_{x\sim p_g}\left[\log\frac{p_g(x)}{p_{\text{data}}(x)+p_g(x)}\right] + \log 2.\end{aligned}$$

Do đó **Jensen–Shannon Divergence** sẽ tiến về $\log 2$

3. Vấn đề mất mát gradient của GAN (gradient vanishing)



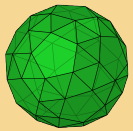
Khi điều này xảy ra $C(G)$ sẽ tiến về 0 vì theo công thức mối liên hệ giữa **hàm mục tiêu của GAN** và **Jensen–Shannon Divergence**:

$$C(G) = -\log 4 + 2\mathrm{JSD}(p_{\text{data}} \parallel p_g).$$
$$\parallel$$
$$log2$$

Dẫn đến việc gradient cho G sẽ trở nên rất yếu (gradient vanishing)

Vấn đề mode collapse của GAN

4. Vấn đề mode collapse của GAN



Khái niệm về mode collapse:

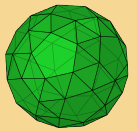
Phân phối dữ liệu thật p_{data} thường có nhiều "chế độ"(modes), tức là các vùng có mật độ xác suất cao.

Ví dụ: Trong tập dữ liệu MNIST, mỗi chữ số từ 0 đến 9 là một "chế độ". Một GAN thành công phải học được cách tạo ra dữ liệu từ tất cả các chế độ này.

Tuy nhiên, trong quá trình huấn luyện, Generator chỉ tìm thấy một hoặc một vài chế độ dễ đánh lừa Discriminator (D) nhất, và sau đó chỉ tập trung vào việc tạo ra dữ liệu thuộc các chế độ đó. Do đó, Generator tạo ra các mẫu rất giống nhau, không phản ánh được sự đa dạng của dữ liệu thật.

WGAN cải tiến GAN như thế nào

5. WGAN cải tiến GAN như thế nào



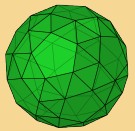
Điểm khác biệt lớn nhất giữa WGAN và GAN là thước đó được sử dụng trong 2 model:

- **GAN:** Jensen–Shannon Divergence
- **WGAN:** Wasserstein distance

Công thức Wasserstein distance (Earth Mover's Distance (EMD)):

$$\mathbb{W}(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

5. WGAN cải tiến GAN như thế nào



Wasserstein distance:

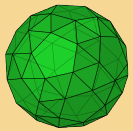
$$\mathbb{W}(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

Ưu điểm của Wasserstein Distances mang lại:

- **Tránh Vanishing Gradient:** Wasserstein distance đo khoảng cách địa lý giữa hai phân phối, nó liên tục và có gradient hữu ích ngay cả khi hai phân phối không chồng lấn.
- **Giải quyết Mode Collapse:** Nhờ có gradient hữu ích, Generator trong WGAN có thể học cách di chuyển phân phối của nó về phía phân phối thật một cách hiệu quả, thay vì chỉ tập trung vào một hoặc vài chế độ dễ lừa Discriminator.

Từ Implementation GAN đến Research DTN

Từ lý thuyết đến ứng dụng



Hành trình nhóm em:

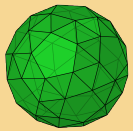
Sau khi xây dựng nền tảng lý thuyết GAN vững chắc, nhóm em quyết định **implement một paper from scratch** và phát triển **website demo**.

"Unsupervised Cross-Domain Image Generation"

Tại sao chọn paper này?

- Nền tảng cho nhiều nghiên cứu GAN sau này
- Đề xuất **constraint** để transfer giữa *unpaired domains*
- Paired data hiếm trong thực tế → ý nghĩa lớn
- Ứng dụng: **Style transfer** (Face→Emoji, SVHN→MNIST)
- Output: giữ đặc trưng gốc nhưng theo phân phối mục tiêu

6.1. Bài toán DTN cố gắng giải quyết



Mục tiêu chính:

- Transfer ảnh từ domain này sang domain khác
- Bảo toàn identity/đặc trưng quan trọng
- Không cần paired data (unsupervised)

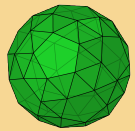
Ví dụ thực tế từ paper:

- Ảnh khuôn mặt thật $\rightarrow$ emoji cá nhân hóa
- SVHN digits $\rightarrow$ MNIST digits

Định nghĩa toán học (Taigman et al.):

- Cho 2 domains S và T, và feature extractor f
- Tìm mapping $G : S \rightarrow T$ sao cho $f(x) \approx f(G(x))$
- (Đặc trưng của ảnh gốc và ảnh sau chuyển đổi phải giống nhau)

6.2.Tại sao GAN gốc gặp khó khăn với bài toán này?



Không kiểm soát được content preservation

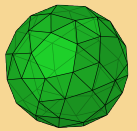
- GAN chỉ quan tâm: generated samples có **giống target distribution** không?
- Không quan tâm: generated samples có **giữ được identity từ input** không?
- Kết quả: Face $\rightarrow$ emoji tạo được emoji đẹp nhưng không giống người gốc

- Standard GAN:

$$\min_G \max_D \; \mathbb{E}[\log D(x)] + \mathbb{E}[\log(1 - D(G(z)))]$$

- Không có cơ chế đảm bảo $f(x) \approx f(G(x))$

6.3. Giải pháp DTN: Ý tưởng chính



Thay đổi cơ bản trong kiến trúc:

Traditional GAN:

$$\mathbf{z} \xrightarrow{G(\cdot)} \text{fake\_image}$$

vs

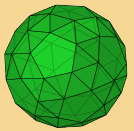
Domain Transfer Network (DTN):

$$\mathbf{x}_s \xrightarrow{f(\cdot)} \mathbf{f}\ (\text{feature}) \xrightarrow{g(\cdot)} \text{customized\_image}$$

Ký hiệu:

- $\mathbf{z}$: Vector noise nhiễu được generate randomly
- $\mathbf{x}_s$: Input image từ source domain
- $f(\cdot)$: Pre-trained feature extractor function
- $\mathbf{f}$: Feature vector (e.g., 512D từ FaceNet)
- $g(\cdot)$: Learnable generator function

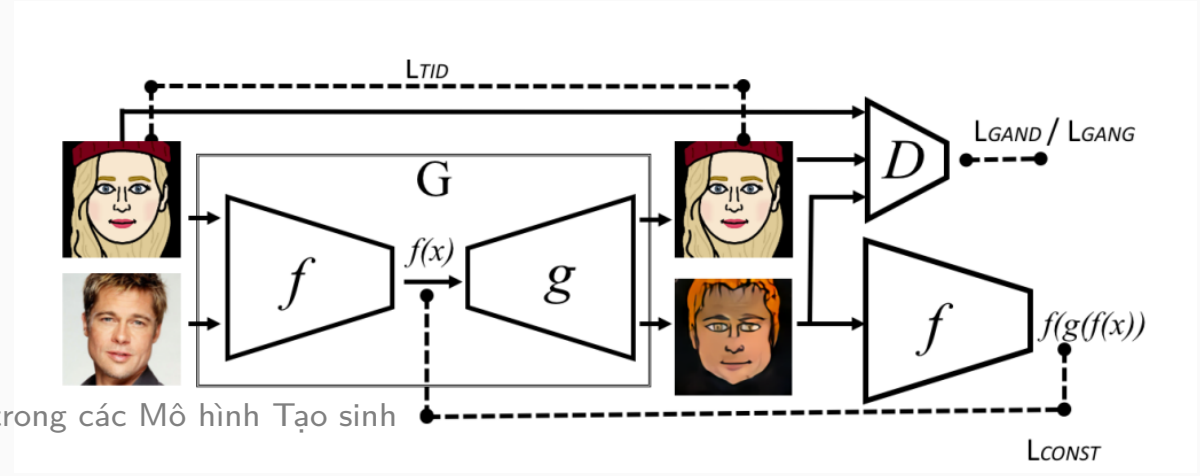
6.4. Kiến trúc DTN: Tổng quan



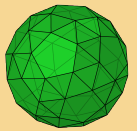
Ý tưởng: dùng pre-trained f làm feature extractor, học g map features sang target domain, giữ identity: $f(x_s) \approx f(g(f(x_s)))$

Các thành phần chính:

- f **(feature extractor):** trích xuất identity features
- g **(learnable):** map feature $\rightarrow$ target domain, 5 layers deconv + batch norm
- D **(ternary discriminator):** phân biệt fake từ source, fake từ target, real target



6.5. Discriminator Loss: Ternary Classification



GAN truyền thống: Binary Classification (2 classes)

Class	Input Type	Mô tả
0	Fake	Generated từ noise vector
1	Real	Samples từ target distribution

DTN Problem: Cần xử lý 2 loại fake samples khác nhau!

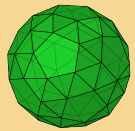
DTN Solution: Ternary Classification (3 classes)

Class	Input Type	Ví dụ
1	Fake từ Source	$g(f(x_s))$ - Face $\rightarrow$ Emoji
2	Fake từ Target	$g(f(x_t))$ - Emoji $\rightarrow$ Emoji (reconstruction)
3	Real Target	x_t - Emoji thật từ dataset

Hàm mất mát:

$$L_D = -\mathbb{E}_{x\sim S} \log D_1(g(f(x))) - \mathbb{E}_{x\sim T} \log D_2(g(f(x))) - \mathbb{E}_{x\sim T} \log D_3(x)$$

6.6. Generator Loss: Tổng quan và các thành phần chính

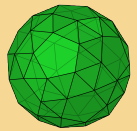


Công thức tổng quát:

$$L_G = \underbrace{L_{GAN}}_{\text{Thực tế}} + \underbrace{\alpha \cdot L_{CONST}}_{\text{Identity}} + \underbrace{\beta \cdot L_{TID}}_{\text{Ổn định}} + \underbrace{\gamma \cdot L_{TV}}_{\text{Mượt}}$$

1. **Adversarial Loss:** $L_{GAN} = -\mathbb{E}_{x_s} \log D_3(g(f(x_s))) - \mathbb{E}_{x_t} \log D_3(g(f(x_t)))$
   - Generator đánh lừa discriminator (class 3 = "real target")
   - Đảm bảo generated images thuộc target domain
2. **F-Constancy Loss (quan trọng nhất):** $L_{\text{CONST}} = \sum_{x \in S} d(f(x), f(g(f(x))))$
   - Face $\rightarrow$ Features $\rightarrow$ Emoji $\rightarrow$ Features' (phải giống nhau)
   - Bảo toàn identity, α (cao nhất)
   - Trái tim DTN - không có thì accuracy giảm 16%

6.6. Generator Loss: Các thành phần regularization



3. Target Identity Loss: $L_{\text{TID}} = \sum_{x \in T} d_2(x, G(x))$

- Identity mapping: Real emoji $\rightarrow$ Generated emoji $\approx$ Real emoji
- Tránh làm hỏng target domain, chống mode collapse
- β ít quan trọng hơn L_{CONST} nhưng vẫn cần thiết

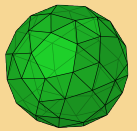
4. Total Variation Loss: $L_{TV}(z) = \sum_{i,j} \left[(z_{i,j+1} - z_{i,j})^2 + (z_{i+1,j} - z_{i,j})^2 \right]^{\frac{1}{2}}$

- Smoothing regularizer: Giảm noise, làm ảnh mượt
- Penalty cho biến đổi đột ngột giữa pixels lân cận
- γ (nhỏ nhất) - chỉ để làm đẹp, không over-smooth

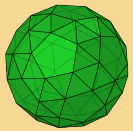
Thứ tự quan trọng: $L_{CONST} > L_{GAN} > L_{TID} > L_{TV}$

Demo

7. Demo



Demo hình ảnh / Video



Tài liệu

[TPW17] Yaniv Taigman, Adam Polyak **and** Lior Wolf. **"Unsupervised Cross-Domain Image Generation".** in*arXiv preprint arXiv:1611.02200*: (2017).