Uczenie Maszynowe 2

Modele generatywne

dr hab. Piotr Duda, prof. AGH i PCz

Plan wykładu

Wykład 1

- **Organiczone Maszyny Boltzmanna**

Wykład 2

- **Autoenkodery, VAE, WAE**

Wykład 3

- **GAN, Normalizing Flows**

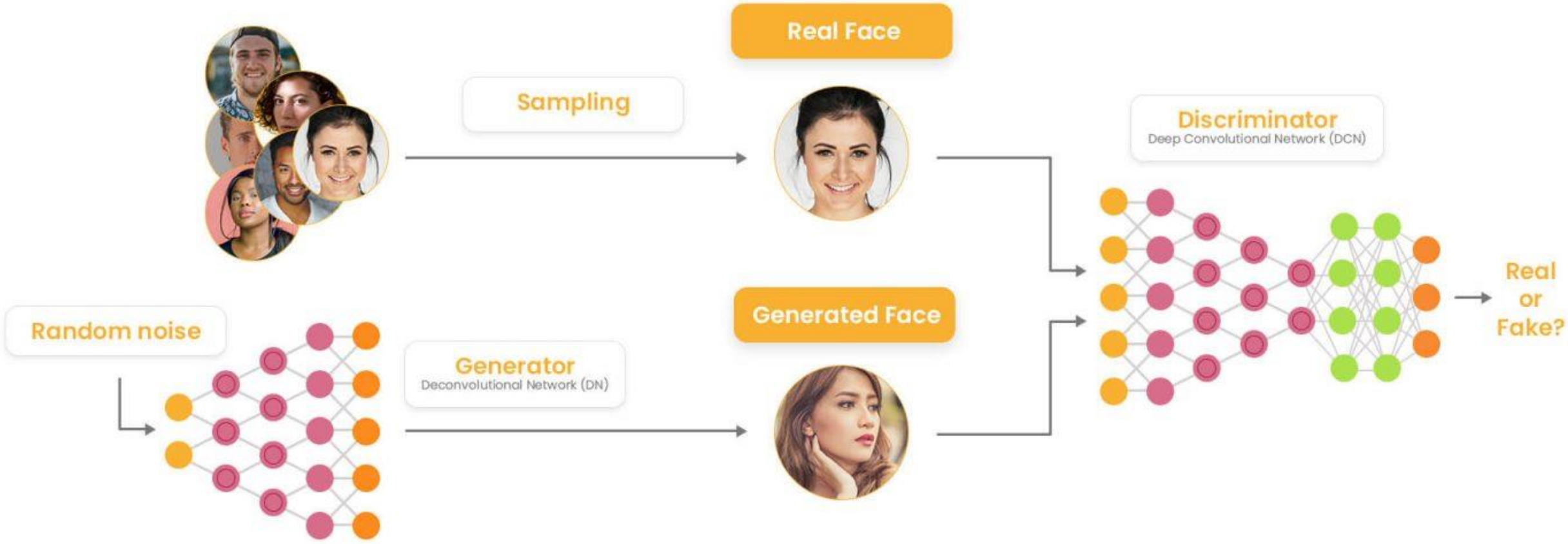
Wykład 4

- **Podejście dyfuzyjne, Przegląd „aktualnych” modeli**

Wykład 2 – Modele generatywne

- GAN
- Problemy z GANami
- Normalizing Flows
- Wybrane podejścia

Generative adversarial network (GAN)



Źródło: https://www.clickworker.com/ai-glossary/generative-adversarial-networks/

Generative adversarial network (GAN)

Został definiowany jako jako gra pomiędzy dwoma graczami:

- **Generator** – ma za zadnie generowanie nowych obrazów
- **Dyskryminator** – ma za zadanie określić czy podany mu obraz jest prawdziwy, czy wygenerowany

Generative adversarial network (GAN)

Theorem (the unique equilibrium point) — For any GAN game, there exists a pair $(\hat{\mu}_D, \hat{\mu}_G)$ that is both a sequential equilibrium and a Nash equilibrium:

$$L(\hat{\mu}_G, \hat{\mu}_D) = \min_{\mu_G} \max_{\mu_D} L(\mu_G, \mu_D) = \qquad \max_{\mu_D} \min_{\mu_G} L(\mu_G, \mu_D) = -2\ln 2$$

$$\hat{\mu}_D \in \arg\max_{\mu_D} \min_{\mu_G} L(\mu_G, \mu_D), \qquad \hat{\mu}_G \in \arg\min_{\mu_G} \max_{\mu_D} L(\mu_G, \mu_D)$$

$$\hat{\mu}_D \in \arg\max_{\mu_D} L(\hat{\mu}_G, \mu_D), \qquad \hat{\mu}_G \in \arg\min_{\mu_G} L(\mu_G, \hat{\mu}_D)$$

$$\forall x \in \Omega, \hat{\mu}_D(x) = \delta_{\frac{1}{2}}, \qquad \hat{\mu}_G = \mu_{\text{ref}}$$

That is, the generator perfectly mimics the reference, and the discriminator outputs $\frac{1}{2}$ deterministically on all inputs.

Funkcja straty

- Funkcja straty dyskryminatora:

$$L_D = -\mathbb{E}_x \log D(x) - \mathbb{E}_z \log(1 - D\big(G(z)\big))$$

- Funkcja straty generatora:

$$L_G = -\mathbb{E}_z \log D((G(z))$$

$D(x)$ to prawdopodobieństwo, że dyskryminator poprawnie klasyfikuje rzeczywiste dane x.

$G(z)$ to próbki wygenerowane przez generator na podstawie z.

Generator

```
def generator(latent_dim):
    model = models.Sequential([
        layers.Dense(7 * 7 * 256, input_dim=latent_dim),
        layers.Reshape((7, 7, 256)),
        layers.BatchNormalization(),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(128, 4, strides=2, padding="same"),
        layers.BatchNormalization(),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(64, 4, strides=2, padding="same"),
        layers.BatchNormalization(),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2D(1, 7, activation="sigmoid", padding="same")
    ])
    return model
```

Dyskryminator

```
def discriminator():
    model = models.Sequential([
        layers.Conv2D(64, 4, strides=2, padding="same", input_shape=(28, 28, 1)),
        layers.LeakyReLU(alpha=0.2),
        layers.Dropout(0.3),
        layers.Conv2D(128, 4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Dropout(0.3),
        layers.Flatten(),
        layers.Dense(1, activation="sigmoid")
    ])
    return model
```

GAN

```python
def generator(latent_dim):
    model = models.Sequential([
        layers.Dense(7 * 7 * 256, input_dim=latent_dim),
        layers.Reshape((7, 7, 256)),
        layers.BatchNormalization(),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(128, 4, strides=2, padding="same"),
        layers.BatchNormalization(),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(64, 4, strides=2, padding="same"),
        layers.BatchNormalization(),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2D(1, 7, activation="sigmoid", padding="same")
    ])
```

```python
G = generator(latent_dim)
D = build_discriminator()
D.compile(optimizer="Adam", loss="binary_crossentropy", metrics=["accuracy"])

gan_input = layers.Input(shape=(latent_dim,))
fake_image = G(gan_input)
gan_output = D(fake_image)
gan = Model(gan_input, gan_output)
gan.compile(optimizer="Adam", loss="binary_crossentropy")
```

```python
def discriminator():
    model = models.Sequential([
        layers.Conv2D(64, 4, strides=2, padding="same", input_shape=(28, 28, 1)),
        layers.LeakyReLU(alpha=0.2),
        layers.Dropout(0.3),
        layers.Conv2D(128, 4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Dropout(0.3),
        layers.Flatten(),
        layers.Dense(1, activation="sigmoid")
    ])
    return model
```

GAN

```
def train_gan(generator, discriminator, gan, data, epochs=10000, batch_size=64):
    for epoch in range(epochs):
        # Pobierz rzeczywiste dane
        idx = np.random.randint(0, data.shape[0], batch_size)
        real_images = data[idx]

        # Generuj fałszywe obrazy
        random_latent_vectors = np.random.normal(size=(batch_size, latent_dim))
        fake_images = generator.predict(random_latent_vectors)

        # Połącz dane i stwórz etykiety
        x = np.concatenate([real_images, fake_images])
        y = np.concatenate([np.ones((batch_size, 1)), np.zeros((batch_size, 1))])

        # Trenuj dyskryminator
        d_loss, d_acc = discriminator.train_on_batch(x, y)

        # Trenuj generator (z dyskryminatorem jako częścią GAN)
        misleading_labels = np.ones((batch_size, 1))
        g_loss = gan.train_on_batch(random_latent_vectors, misleading_labels)

# Zamrażamy dyskryminator podczas treningu generatora
gan_input = layers.Input(shape=(latent_dim,))
fake_image = G(gan_input)
D.trainable = False
gan_output = D(fake_image)
gan = Model(gan_input, gan_output)
gan.compile(optimizer="Adam", loss="binary_crossentropy")
train_gan(G, D, gan, x_train)
```

Problemy z GAMami

- Mode Collapse (mode dropping) - generator generuje ograniczony zbiór bardzo podobnych próbek, ignorując pełną różnorodność danych rzeczywistych.

- Overfitting Dyskryminatora - dyskryminator staje się zbyt potężny i szybko adaptuje się do generatora, utrudniając poprawne szkolenie GAN.

VAE-GUN

$$p(\mathrm{Dis}_l(\boldsymbol{x})|\boldsymbol{z}) = \mathcal{N}(\mathrm{Dis}_l(\boldsymbol{x})|\,\mathrm{Dis}_l(\tilde{\boldsymbol{x}}), \mathbf{I})$$

$$\mathcal{L}_{\text{llike}}^{\mathrm{Dis}_l} = -\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\mathrm{Dis}_l(\boldsymbol{x})|\boldsymbol{z})\right]$$

$$L_D = -\mathbb{E}_x \log D(x) - \mathbb{E}_z \log(1 - D\big(G(z)\big))$$

$$L_{KL} = -\int q\,(z|x) log\,\frac{q(z|x)}{p(z)}\,dz$$

$$L = L_D + L_{llike}^{Dis_l} + L_{KL}$$

Algorithm 1 Training the VAE/GAN model

$\boldsymbol{\theta}_{\text{Enc}}, \boldsymbol{\theta}_{\text{Dec}}, \boldsymbol{\theta}_{\text{Dis}} \leftarrow$ initialize network parameters

repeat

$\boldsymbol{X} \leftarrow$ random mini-batch from dataset

$\boldsymbol{Z} \leftarrow \text{Enc}(\boldsymbol{X})$

$\mathcal{L}_{\text{prior}} \leftarrow D_{\text{KL}}(q(\boldsymbol{Z}|\boldsymbol{X})\|p(\boldsymbol{Z}))$

$\tilde{\boldsymbol{X}} \leftarrow \text{Dec}(\boldsymbol{Z})$

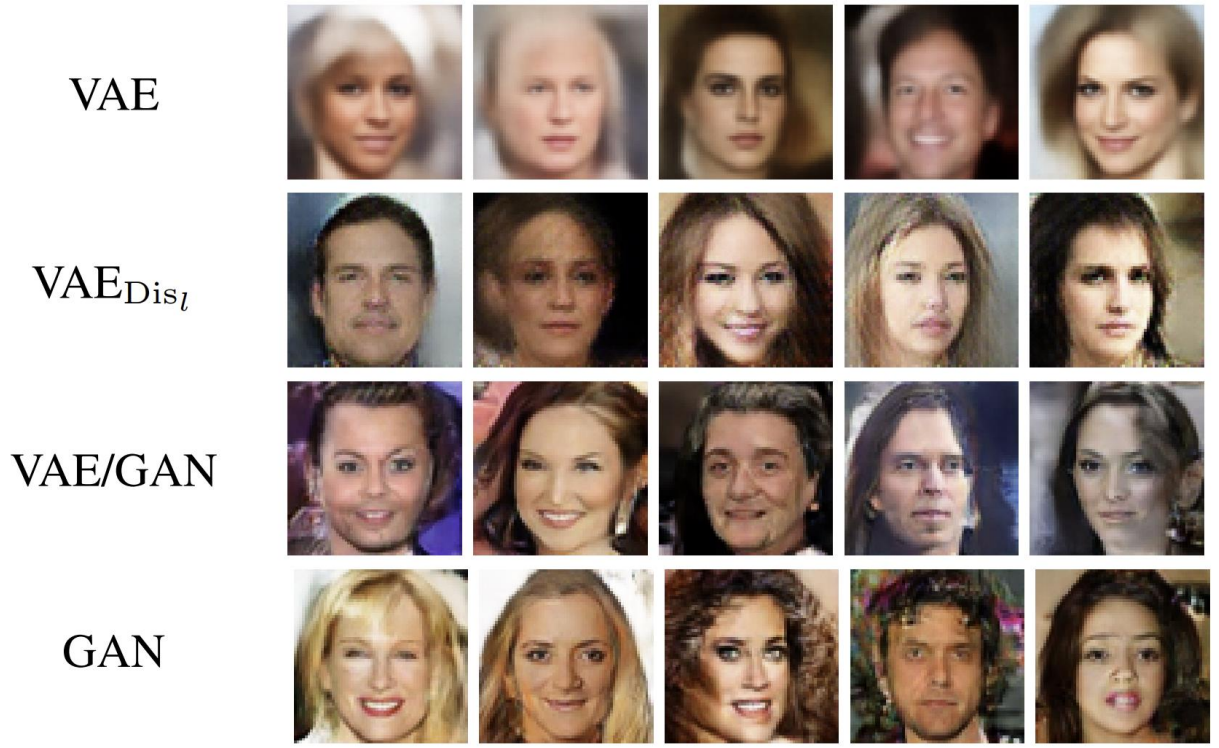
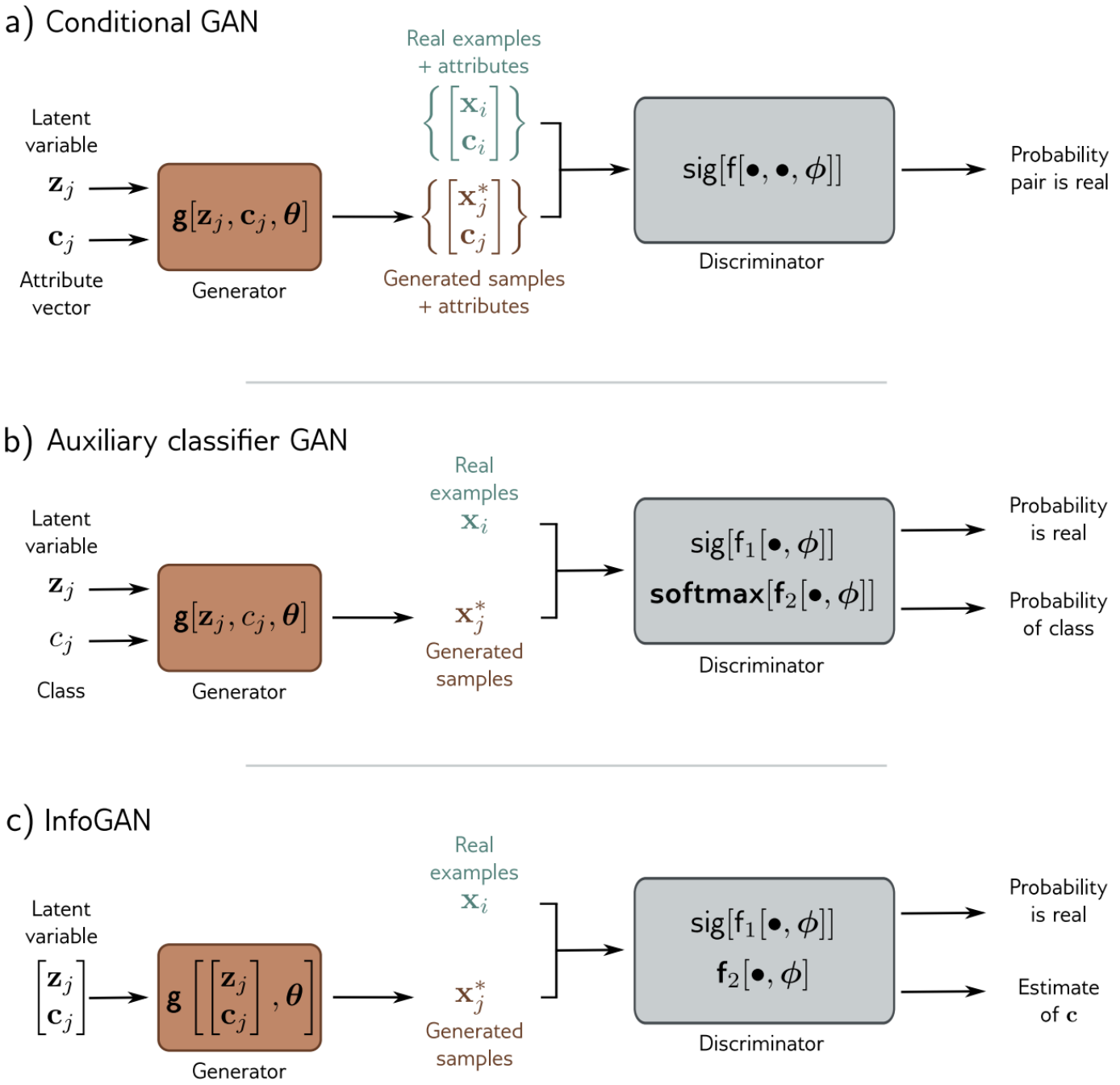


Figure 3. Samples from different generative models.

Warunkowanie generowanego obiektu



Prince, Simon JD. *Understanding deep learning*. MIT press, 2023.

Szczególne przypadki:

- Wasserstein GAN

Arjovsky, Martin, Soumith Chintala, and Léon Bottou. "Wasserstein generative adversarial networks." International conference on machine learning. PMLR, 2017.

- Conditional GAN

Mirza, Mehdi. "Conditional generative adversarial nets." arXiv preprint arXiv:1411.1784 (2014).

- CycleGAN

Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros, Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, arXiv preprint arXiv:1703.10593

- StyleGAN

Tero Karras, Samuli Laine and Timo Aila, A Style-Based Generator Architecture for Generative Adversarial Networks, arXiv preprint arXiv: 1812.04948 (2018)

- Progressive Growing GAN

Tero Karras, Timo Aila, Samuli Laine and Jaakko Lehtinen, Progressive Growing of GANs for Improved Quality, Stability, and Variation, arXiv preprint arXiv: 1710.10196 (2017)

- BigGAN

Andrew Brock, Jeff Donahue, Karen Simonyan, Large Scale GAN Training for High Fidelity Natural Image Synthesis, arXiv preprint arXiv: 1809.11096 (2018)

Nie tylko MSE, MAE, CE

Net – wstępnie wytrenowana sieć (np. VGG)

Net_l - wyjście z l-tej warstwy ukrytej sieci (konwolucyjnej)

x – obraz o pożądanych cechach

$\tilde{x}$ – wygenerowany obraz

$$L_{perceptual} = \|Net_l(x) - Net_l(\tilde{x})\|^2$$

Style Loss

x – obraz wzorcowy

$\tilde{x}$ – wygenerowany obraz

$$L_{perceptual} = \|G(Net_l(x)) - G(Net_l(\tilde{x}))\|^2,$$

gdzie G to macierz Grama

Przykład: macierz Grama

1. Załóżmy, że mamy feature mapę o wymiarach $C \times H \times W$,

C – liczba kanałów

H – wysokość

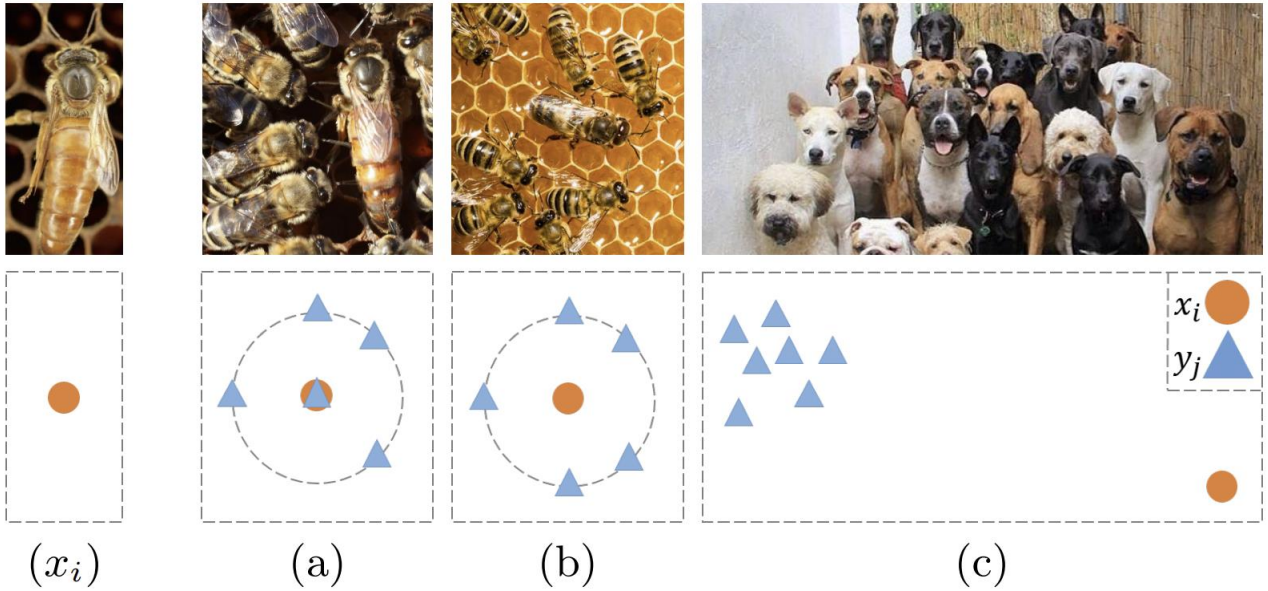
W – szerokość

2. Przekształcamy tę feature mapę w macierz F o wymiarach $C \times (H \cdot W)$

3. Obliczamy macierz G o wymiarach $C \times C$ reprezentującą korelację między kanałami.

$$G = F \cdot F^T$$

The Contextual loss



x, y – porównywane obrazy

$$L_{CX}(x, y, l) = -\log($$

Contextual Similarity

X, Y – tensory o

$$CX(X, Y) = \frac{1}{N}\sum_j \max_i CX_{ij},$$

gdzie

$$CX_{ij} = \frac{w_{ij}}{\sum_k w_{ik}}, \quad w_{ij} = \exp\left(\frac{1-\tilde{d}_{i,j}}{h}\right), \tilde{d}_{i,j} = \frac{d_{i,j}}{\min_k d_{i,k}+\varepsilon}$$

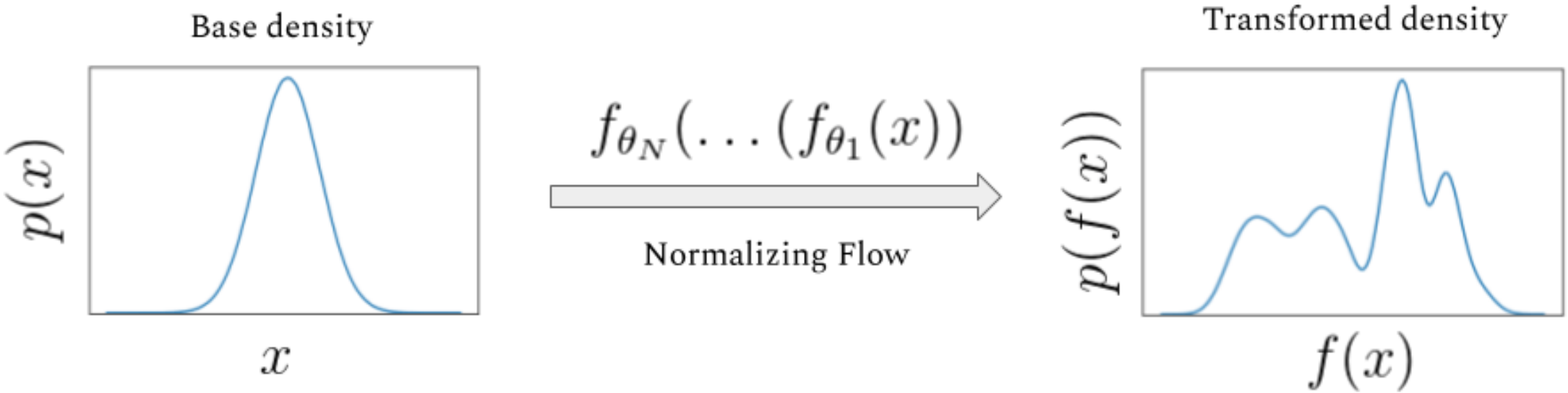
$d_{i,j}$ - odległość cosinusowa pomiędzy cechami x_i i y_j

Mechrez, Roey, Itamar Talmi, and Lihi Zelnik-Manor. "The contextual loss for image transformation with non-aligned data." *Proceedings of the European conference on computer vision (ECCV)*. 2018.

Halucynacje

- **Zniekształcone cechy**: Na wygenerowanych obrazach mogą pojawić się zdeformowane lub niewłaściwe cechy, takie jak błędne proporcje twarzy, nierealne kształty ciał czy zniekształcone obiekty.
- **Artefakty**: Nieprawidłowości takie jak ziarno, smugi, czy inne niepożądane tekstury, które są efektem ubocznym procesu generowania obrazów.
- **Nierealne elementy**: Wygenerowane obrazy mogą zawierać nierealne przedmioty, które nie istnieją w rzeczywistości, jak np. skrzydła w dziwnych miejscach, czy elementy, które wyglądają jakby były połączone w sposób nienaturalny.

Normalizing Flows (NF)



Normalizing Flows (NF)

Modelujemy docelowy rozkład danych $p_X(x)$ jako funkcję zmiennej z pochodzącej z rozkładu normalnego $p_Z(z) = N(0, I)$.

$$x = f(z),$$

gdzie f to funcja odwracalna i różniczkowalna.

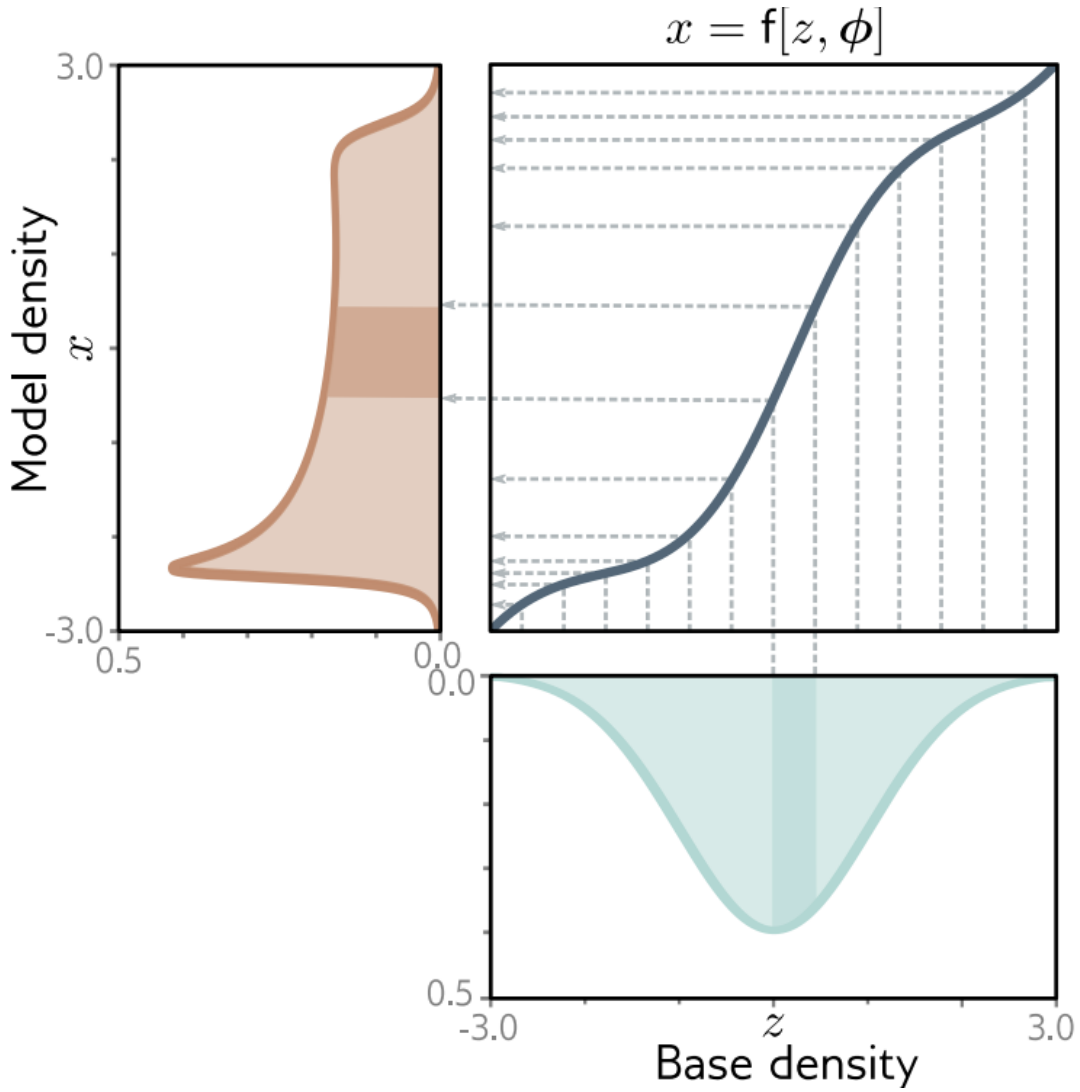
Normalizing Flows (NF)

- **Reguła zmiany zmiennych:**

Aby obliczyć gęstość prawdo
$p_X(x)$, używa się reguły zmiar

$$p_X(x) = p_Z(z) \left| det \frac{\partial}{} \right.$$

gdzie $\frac{\partial f^{-1}(x)}{\partial x}$ to macierz Jakob



Normalizing Flows (NF)

W praktyce przekształcenie f rozpatruje się często jako ciąg przekształceń $f_1, \dots f_K$. Wówczas

$$p_X(x) = p_Z(z) \prod_{i=1}^{K} \left| det \frac{\partial f_i^{-1}(x)}{\partial x} \right|$$

$$\log(p_X(x)) = \log(p_Z(z)) + \sum_{i=1}^{K} log \left| det \frac{\partial f_i^{-1}(x)}{\partial x} \right|$$

Normalizing Flows (NF)

W praktyce przekształcenie f rozpatruje się często jako ciąg przekształceń $f_1, \dots f_K$. Wówczas

$$p_X(x) = p_Z(z) \prod_{i=1}^{K} \left| det \frac{\partial f_i^{-1}(x)}{\partial x} \right|^{-1},$$

$$\log(p_X(x)) = \log(p_Z(z)) - \sum_{i=1}^{K} log \left| det \frac{\partial f_i^{-1}(x)}{\partial x} \right|$$

Normalizing Flows, a sieci neuronowe

$$\log(p_X(x)) = \log(p_Z(z)) - \sum_{i=1}^{K} log \left| det \frac{\partial f_i^{-1}(x)}{\partial x} \right|$$

Załóżmy, że f jest wielowarstwową siecią neuronową. Wówczas $f_1, \dots f_K$ reprezentują kolejne warstwy.

$$x = f(x, \theta) = f_K[f_{K-1}[\dots f_1(z, \theta_1) \dots]\theta_{K-1}]\theta_K],$$

gdzie $\theta_1, \dots, \theta_K$ to wagi kolejnych warstw.

Normalizing Flows, a sieci neuronowe

Załóżmy, że f jest wielowarstwową siecią neuronową. Wówczas $f_1, \dots f_K$ reprezentują kolejne warstwy.

$$x = f(x, \theta) = f_K[f_{K-1}[\dots f_1(z, \theta_1) \dots]\theta_{K-1}]\theta_K],$$

Wówczas inverse mapping (normalizing direction)

$$z = f^{-1}(x, \theta) = f_1^{-1}\left[f_2^{-1}\left[\dots f_K^{-1}(x, \theta_K) \dots\right]\theta_2\right]\theta_1\right],$$

Normalizing Flows, a sieci neuronowe

$$\frac{\partial f(z,\theta)}{\partial z} = \frac{\partial f_K(f_{K-1},\theta_K)}{\partial f_{K-1}} \cdot \frac{\partial f_{K-1}(f_{K-2},\theta_{K-1})}{\partial f_{K-2}} \cdot \ldots \cdot \frac{\partial f_1(z,\theta_1)}{\partial z}$$

$$\left|\frac{\partial f(z,\theta)}{\partial z}\right| = \left|\frac{\partial f_K(f_{K-1},\theta_K)}{\partial f_{K-1}}\right| \cdot \left|\frac{\partial f_{K-1}(f_{K-2},\theta_{K-1})}{\partial f_{K-2}}\right| \cdot \ldots \cdot \left|\frac{\partial f_1(z,\theta_1)}{\partial z}\right| \cdot$$

Jak dobrać warstw

Jak dobrać warstw

Linear flows

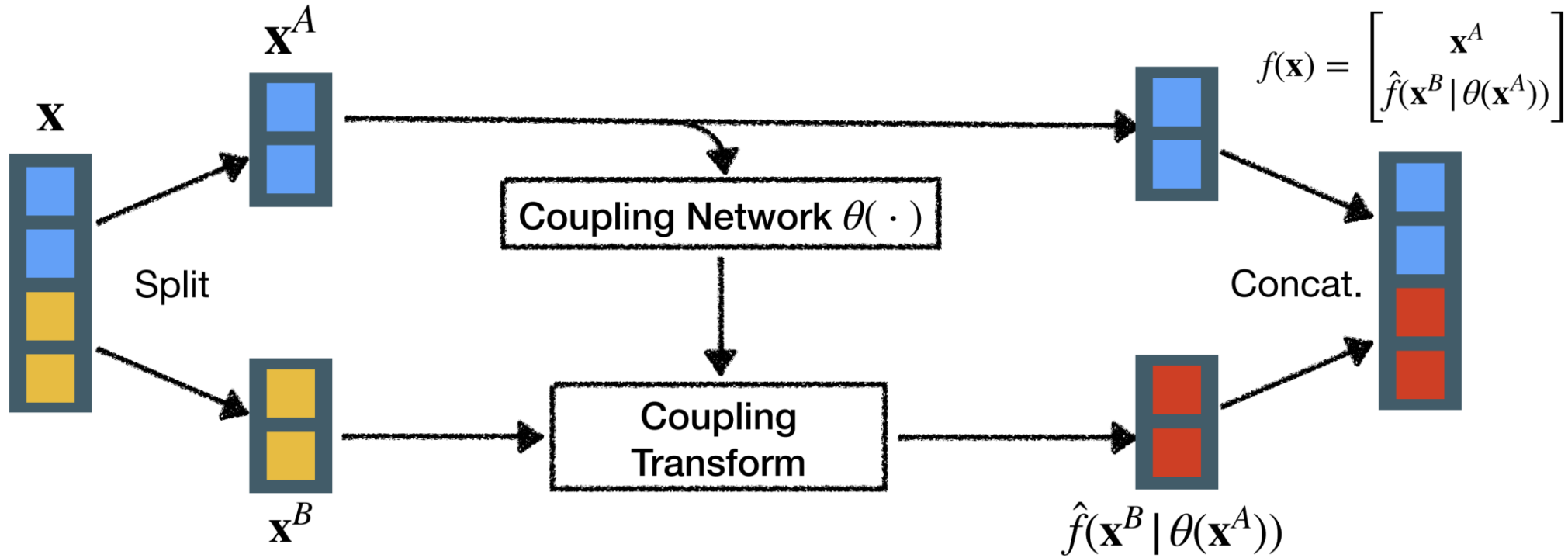
$$f(x, \theta) = Ax + b,$$

gdzie A to odwracalna macierz wag o wymiarze *d na d*.

Jak dobrać warstw

Coupling flow

$$f(x) = (x^{(A)}, \hat{f}(x^{(B)} | \theta(x^{(A)})))$$
$$f^{-1}(x) = (x^{(A)}, \hat{f}^{-1}(x^{(B)} | \theta(x^{(A)})))$$



Ilustracja: https://mbrubake.github.io/cvpr2021-nf_in_cv-tutorial/Introduction%20-%20CVPR2021.pdf

Jak dobrać warstw

Table 1: The three main components of our proposed flow, their reverses, and their log-determinants. Here, $\mathbf{x}$ signifies the input of the layer, and $\mathbf{y}$ signifies its output. Both $\mathbf{x}$ and $\mathbf{y}$ are tensors of shape $[h \times w \times c]$ with spatial dimensions (h, w) and channel dimension c. With (i, j) we denote spatial indices into tensors $\mathbf{x}$ and $\mathbf{y}$. The function $\texttt{NN}()$ is a nonlinear mapping, such as a (shallow) convolutional neural network like in ResNets (He et al., 2016) and RealNVP (Dinh et al., 2016).

Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i, j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i, j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$
Invertible 1×1 convolution. $\mathbf{W} : [c \times c]$. See Section 3.2.	$\forall i, j : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$\forall i, j : \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$	$h \cdot w \cdot \log \lvert\det(\mathbf{W})\rvert$ or $h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$ (see eq. (10))
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \texttt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \texttt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \texttt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \texttt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\texttt{sum}(\log(\lvert\mathbf{s}\rvert))$

https://paperswithcode.com/method/affine-coupling

Jak dobrać warstw

Affice autoregesive flow:

$$f(x) = \frac{x - \mu}{\sigma}$$

Operacja odwrotna:

$$f^{-1}(z) = \mu + \sigma z$$

Jak dobrać warstw

Affice autoregesive flow:

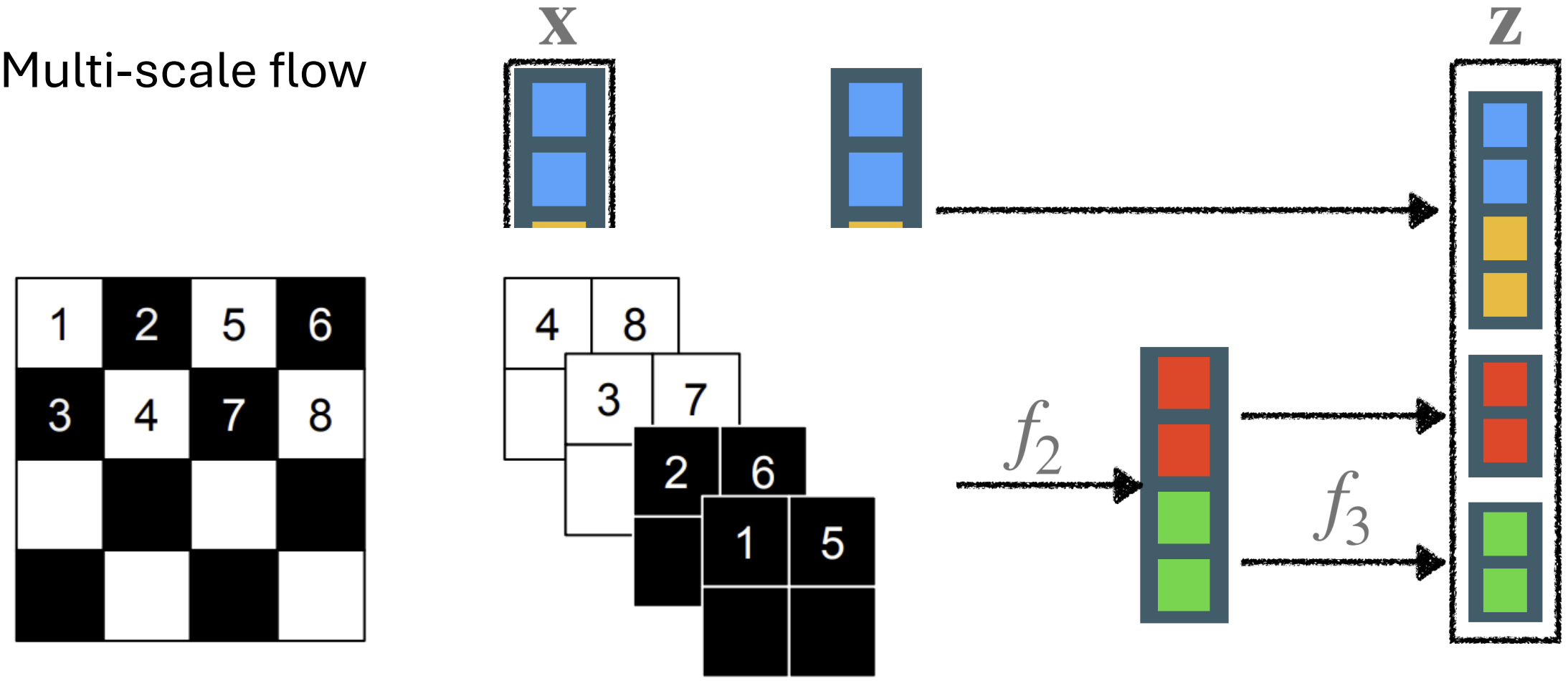
$$f_i(x) = \frac{x_i - \mu(x_{i-1}, \dots, x_1)}{\sigma(x_{i-1}, \dots, x_1)}$$

Operacja odwrotna:

$$f^{-1}(z) = \mu + \sigma x$$

Jak dobrać warstw

Multi-scale flow



Ilustracja: https://mbrubake.github.io/cvpr2021-nf_in_cv-tutorial/Introduction%20-%20CVPR2021.pdf

NICE

NICE: NON-LINEAR INDEPENDENT COMPONENTS ESTIMATION

Laurent Dinh David Krueger Yoshua Bengio*
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

ABSTRACT

We propose a deep learning framework for modeling complex high-dimensional densities called Non-linear Independent Component Estimation (NICE). It is based on the idea that a good representation is one in which the data has a distribution that is easy to model. For this purpose, a non-linear deterministic transformation of the data is learned that maps it to a latent space so as to make the transformed data conform to a factorized distribution, i.e., resulting in independent latent variables. We parametrize this transformation so that computing the determinant of the Jacobian and inverse Jacobian is trivial, yet we maintain the ability to learn complex non-linear transformations, via a composition of simple building blocks, each based on a deep neural network. The training criterion is simply the exact log-likelihood, which is tractable. Unbiased ancestral sampling is also easy. We show that this approach yields good generative models on four image datasets and can be used for inpainting.

RealNVP

Real-valued

Non-Volume

Preserving

DENSITY ESTIMATION USING REAL NVP

Laurent Dinh*
Montreal Institute for Learning Algorithms
University of Montreal
Montreal, QC H3T1J4

Jascha Sohl-Dickstein
Google Brain

Samy Bengio
Google Brain

ABSTRACT

Unsupervised learning of probabilistic models is a central yet challenging problem in machine learning. Specifically, designing models with tractable learning, sampling, inference and evaluation is crucial in solving this task. We extend the space of such models using real-valued non-volume preserving (real NVP) transformations, a set of powerful, stably invertible, and learnable transformations, resulting in an unsupervised learning algorithm with exact log-likelihood computation, exact and efficient sampling, exact and efficient inference of latent variables, and an interpretable latent space. We demonstrate its ability to model natural images on four datasets through sampling, log-likelihood evaluation, and latent variable manipulations.

Glow

Glow: Generative Flow with Invertible 1×1 Convolutions

Diederik P. Kingma[*†], **Prafulla Dhariwal**[*]
[*]OpenAI
[†]Google AI

Abstract

Flow-based generative models (Dinh et al., 2014) are conceptually attractive due to tractability of the exact log-likelihood, tractability of exact latent-variable inference, and parallelizability of both training and synthesis. In this paper we propose *Glow*, a simple type of generative flow using an invertible 1×1 convolution. Using our method we demonstrate a significant improvement in log-likelihood on standard benchmarks. Perhaps most strikingly, we demonstrate that a flow-based generative model optimized towards the plain log-likelihood objective is capable of efficient realistic-looking synthesis and manipulation of large images. The code for our model is available at `https://github.com/openai/glow`.

GLOW

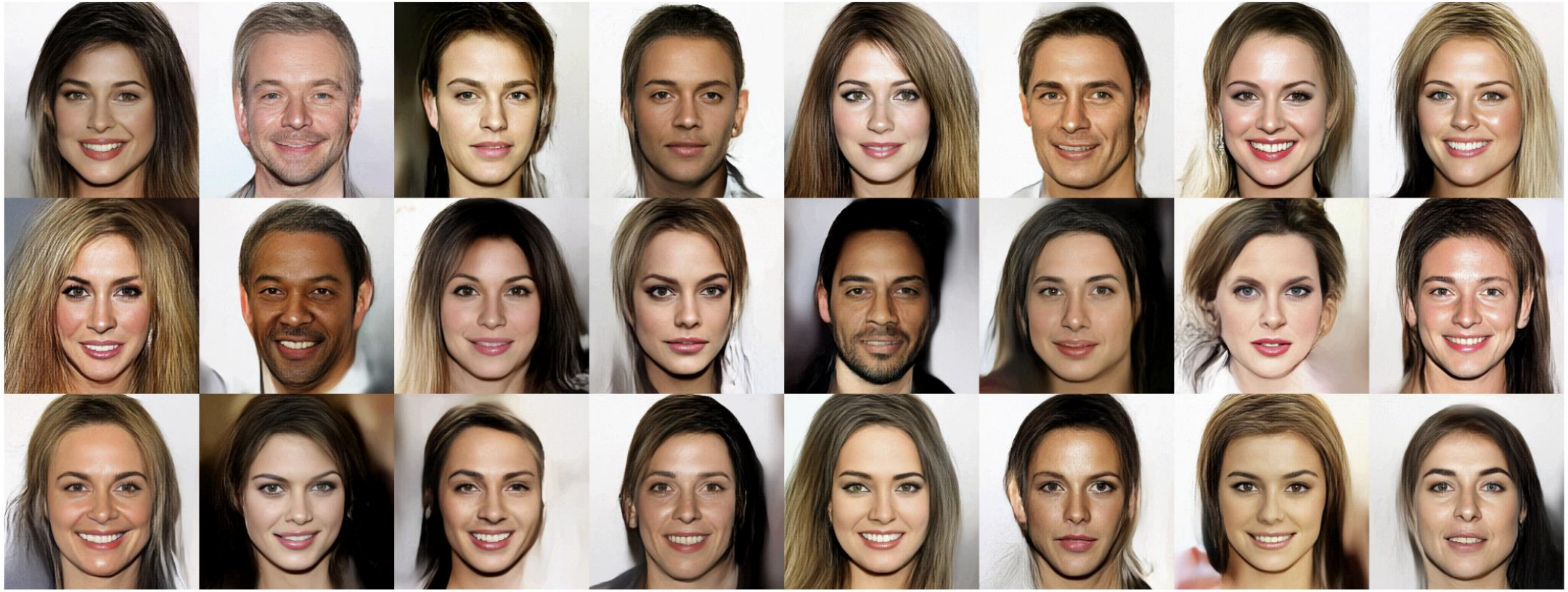


Figure 4: Random samples from the model, with temperature 0.7.

GLOW

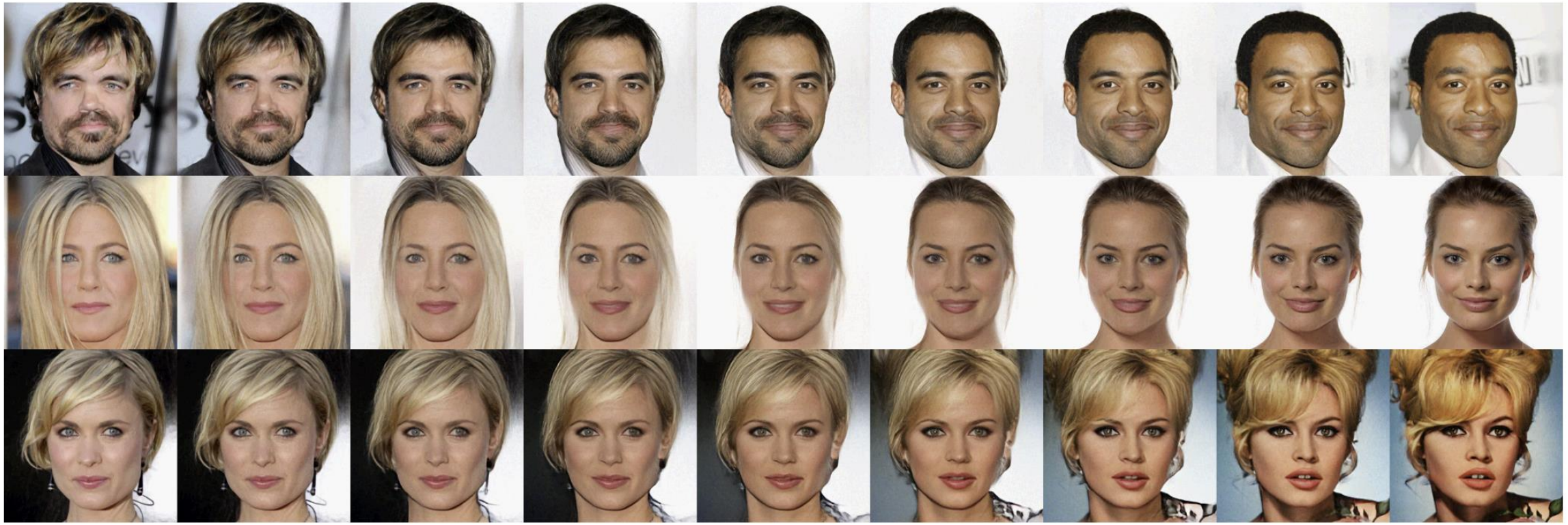
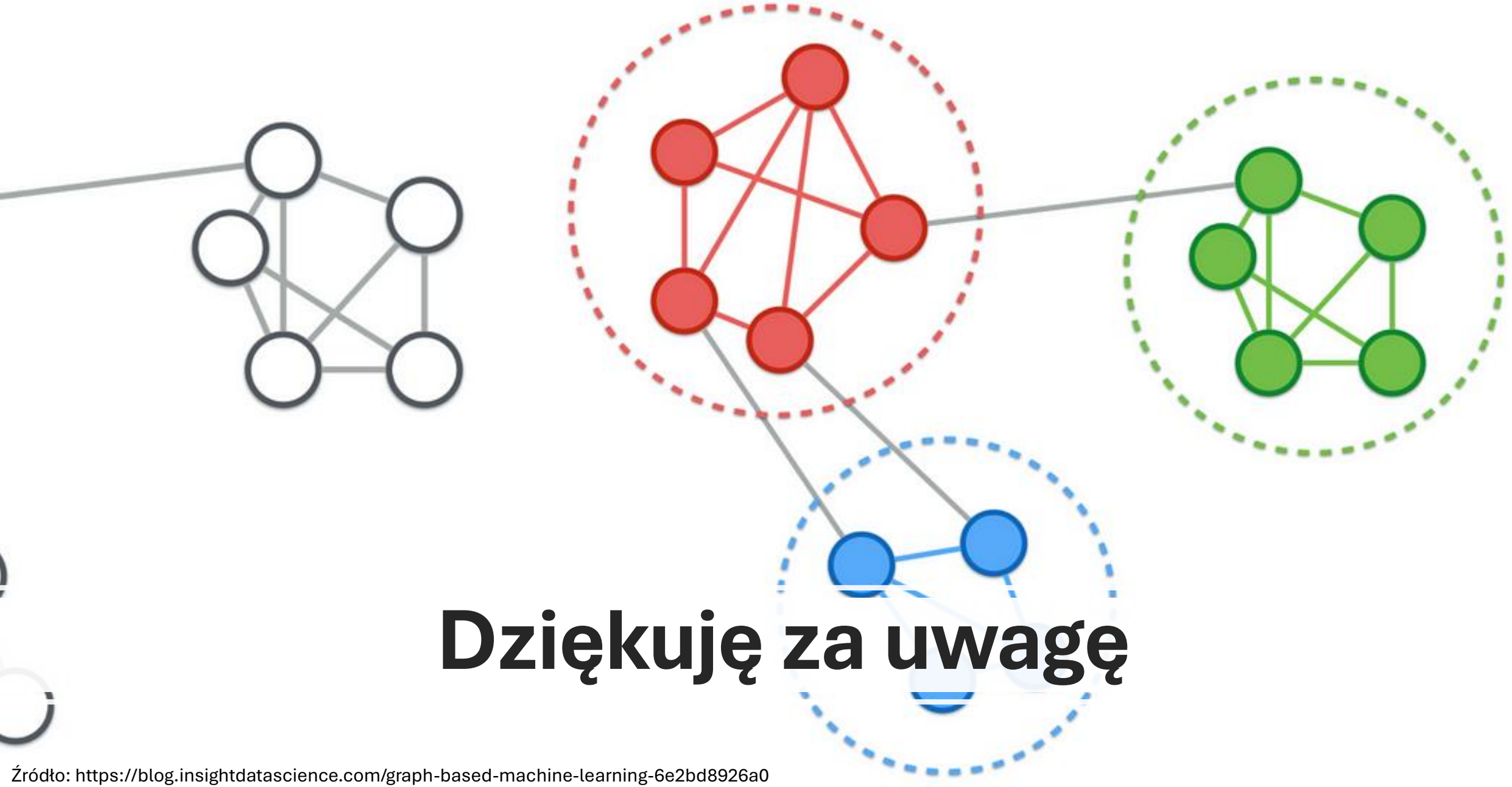


Figure 5: Linear interpolation in latent space between real images.

Porównanie metod

Kryterium	Normalizing Flows (NF)	Variational Autoencoders (VAE)	Generative Adversarial Networks (GAN)
Modelowanie rozkładu prawdopodobieństwa	Tak (dokładne modelowanie rozkładu)	Tak (rozważanie rozkładu latentnego)	Nie (brak jawnego modelu rozkładu)
Jakość generowanych próbek	Bardzo dobra, kontrolowana	Często gorsza (blurry), ale wystarczająca do wielu zadań	Bardzo dobra, szczególnie w zadaniach obrazowych
Obliczanie gęstości prawdopodobieństwa	Tak, dokładnie obliczane	Tak, ale tylko przy założeniu rozkładu normalnego w przestrzeni latentnej	Nie, nie obliczają dokładnej gęstości
Trudność trenowania	Umiarkowana (wymaga starannego projektowania transformacji)	Łatwiejsze, ale może generować mniej wyraźne próbki	Trudniejsze, problem z równowagą między generatorem a dyskryminatorem
Zastosowanie	Modelowanie gęstości, generowanie próbek, analiza anomalii	Generowanie próbek, inferencja bayesowska, modelowanie danych	Generowanie obrazów, wideo, muzyki, tekstów

Pytania / Komentarze / Uwagi



Dziękuję za uwagę

Źródło: https://blog.insightdatascience.com/graph-based-machine-learning-6e2bd8926a0