भारतीय प्रौद्योगिकी संस्थान दिल्ली
Indian Institute of Technology Delhi

COV877
Special Module on Visual Computing

Generative AI for Visual Content Creation: Image, Video, and 3D

StyleGAN, Normalizing Flows and VAE

Instructor:

Dr. Lokender Tiwari

Research Scientist

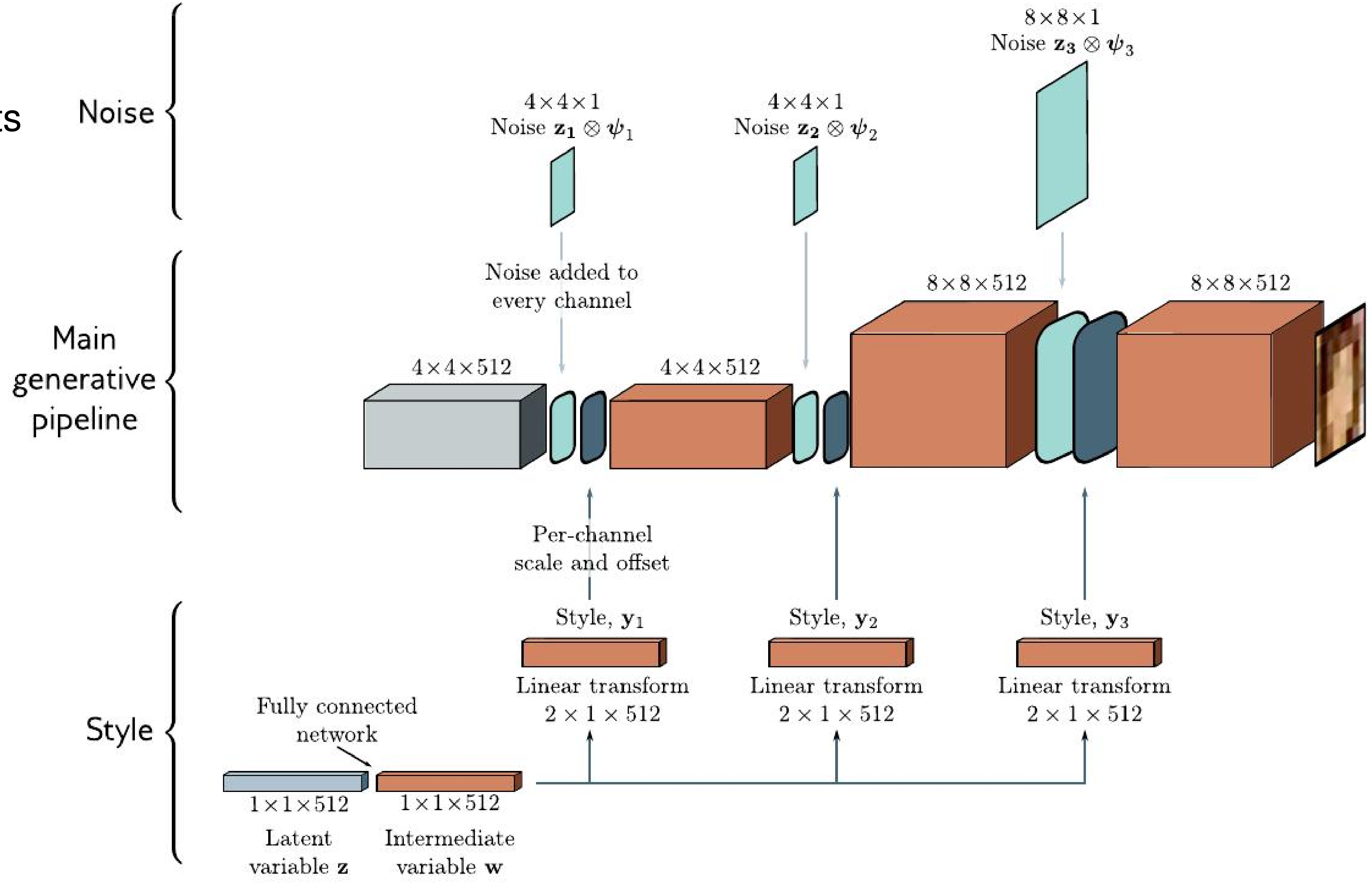
StyleGAN

Image Translation - StyleGAN

- StyleGAN controls the output image at different scales and separates style from noise

- latent variable injected to the inputs of the generator at various points

- to modify the current representation at these points in different ways.



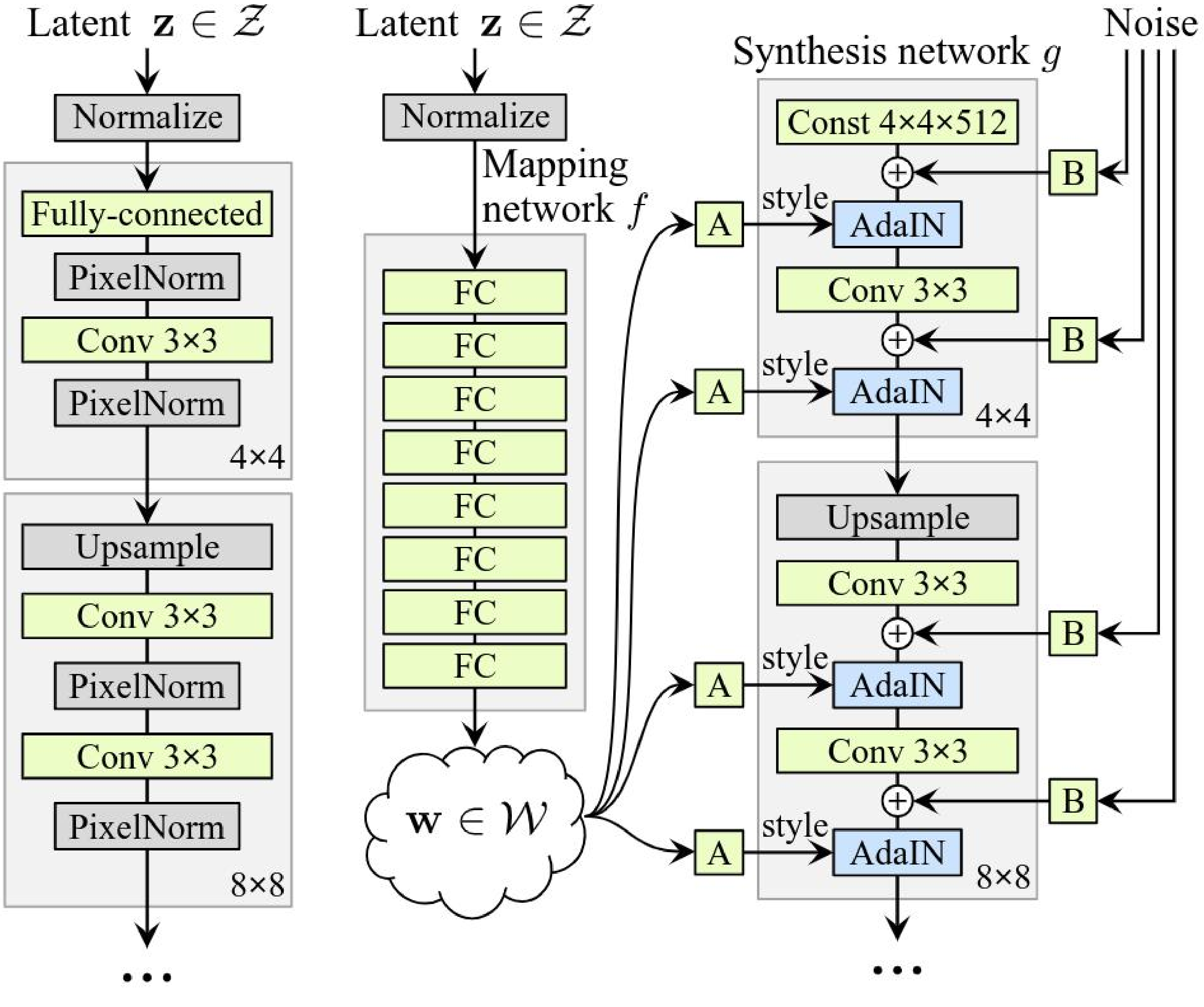
Karras, T., Laine, S., & Aila, T. (2019). A style based generator architecture for generative adversarial networks, CVPR 2019

Image Translation - StyleGAN

- Automatically learn, unsupervised separation of high-level attributes (e.g., pose and identity when trained on human faces) and
- Stochastic variation in the generated images (e.g., freckles, hair)
- Provides intuitive, scale-specific control of the synthesis

Key Philosophy

- control the strength of image features at different scales



(a) Traditional (b) Style-based generator

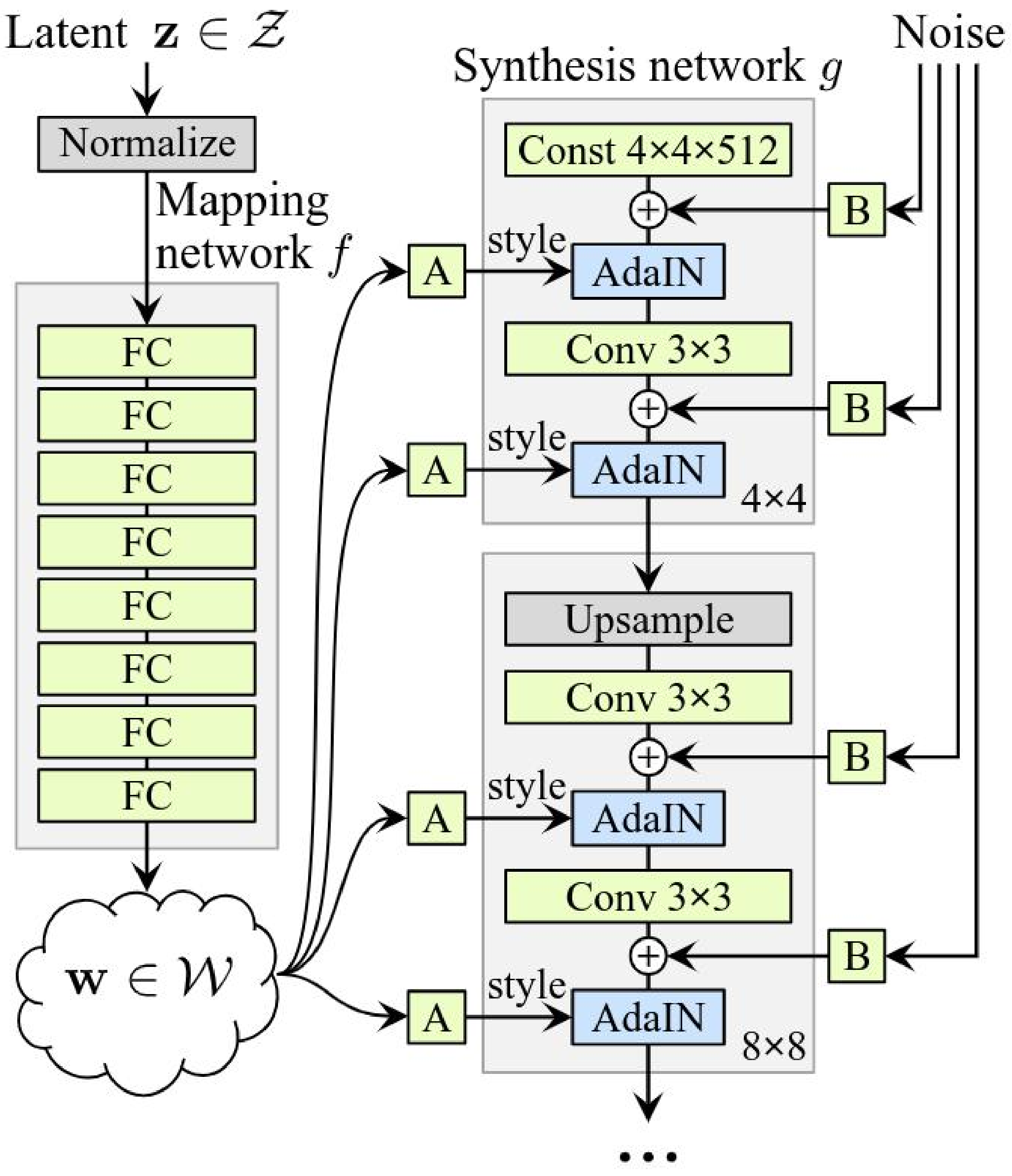
Karras, T., Laine, S., & Aila, T. (2019). A style based generator architecture for generative adversarial networks, CVPR 2019

Image Translation - StyleGAN

- Map the input to an intermediate latent space W
- Controls the generator through adaptive instance normalization (AdaIN) at each convolution layer
- Gaussian noise is added after each convolution

"A" learned affine transform
"B" applies learned per-channel scaling factors to the noise input

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i}$$



(b) Style-based generator

Karras, T., Laine, S., & Aila, T. (2019). A style based generator architecture for generative adversarial networks, CVPR 2019

Image Translation - StyleGAN

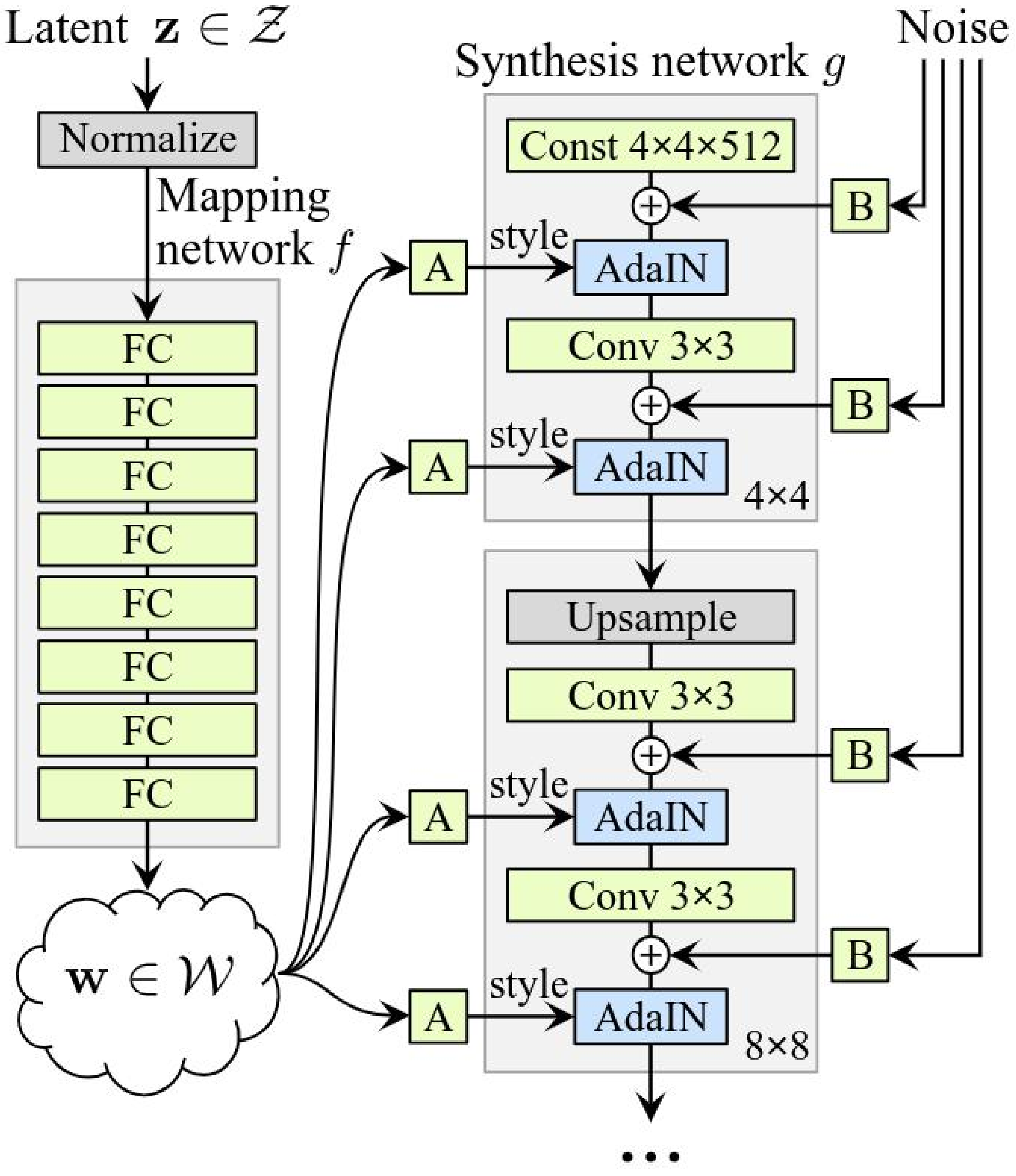
Properties of StyleGAN

- Mapping network and affine transformations focus on drawing samples for each style from a learned distribution
- Synthesis network generates a novel image based on a collection of styles.
- The effects of each style are localized in the network, i.e., modifying a specific subset of the styles can be expected to affect only certain aspects of the image.

Reason ?

AdaIN

- It modifies the relative importance of features for the subsequent convolution operation, (not depending on original statistics)
- Each style controls only one convolution before being overridden by the next AdaIN operation.



(b) Style-based generator

Karras, T., Laine, S., & Aila, T. (2019). A style based generator architecture for generative adversarial networks, CVPR 2019

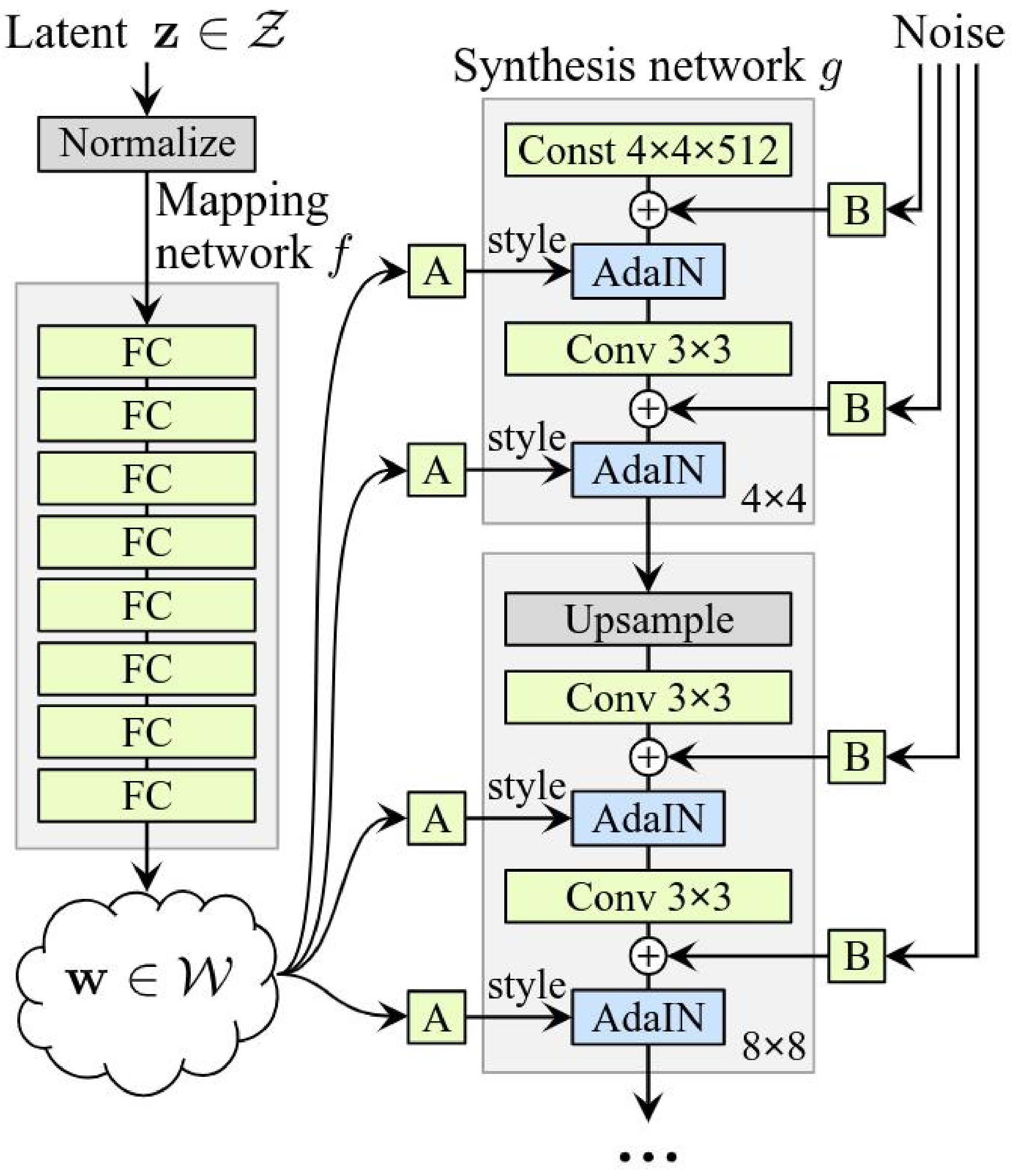
Image Translation - StyleGAN

Another reason - Style Mixing

- Run two latent codes z1, z2 through the mapping network,
- the corresponding w1, w2 control the styles so that w1 applies before the crossover point and w2 after it.

- Coarse [4x4–8x8]
- Middle [16x16–32x32]
- Fine [64x64–1024x1024]

- Improves the localization considerably, (improved FIDs) in scenarios where multiple latents are mixed at test time



(b) Style-based generator

Image Translation - StyleGAN

Another reason - Style Mixing

- Run two latent codes z1, z2 through the mapping network,
- the corresponding w1, w2 control the styles so that w1 applies before the crossover point and w2 after it.

to prevents network from assuming that adjacent styles are correlated.

- Coarse [4x4–8x8]
- Middle [16x16–32x32]
- Fine [64x64–1024x1024]

- Improves the localization considerably, (improved FIDs) in scenarios where multiple latents are mixed at test time

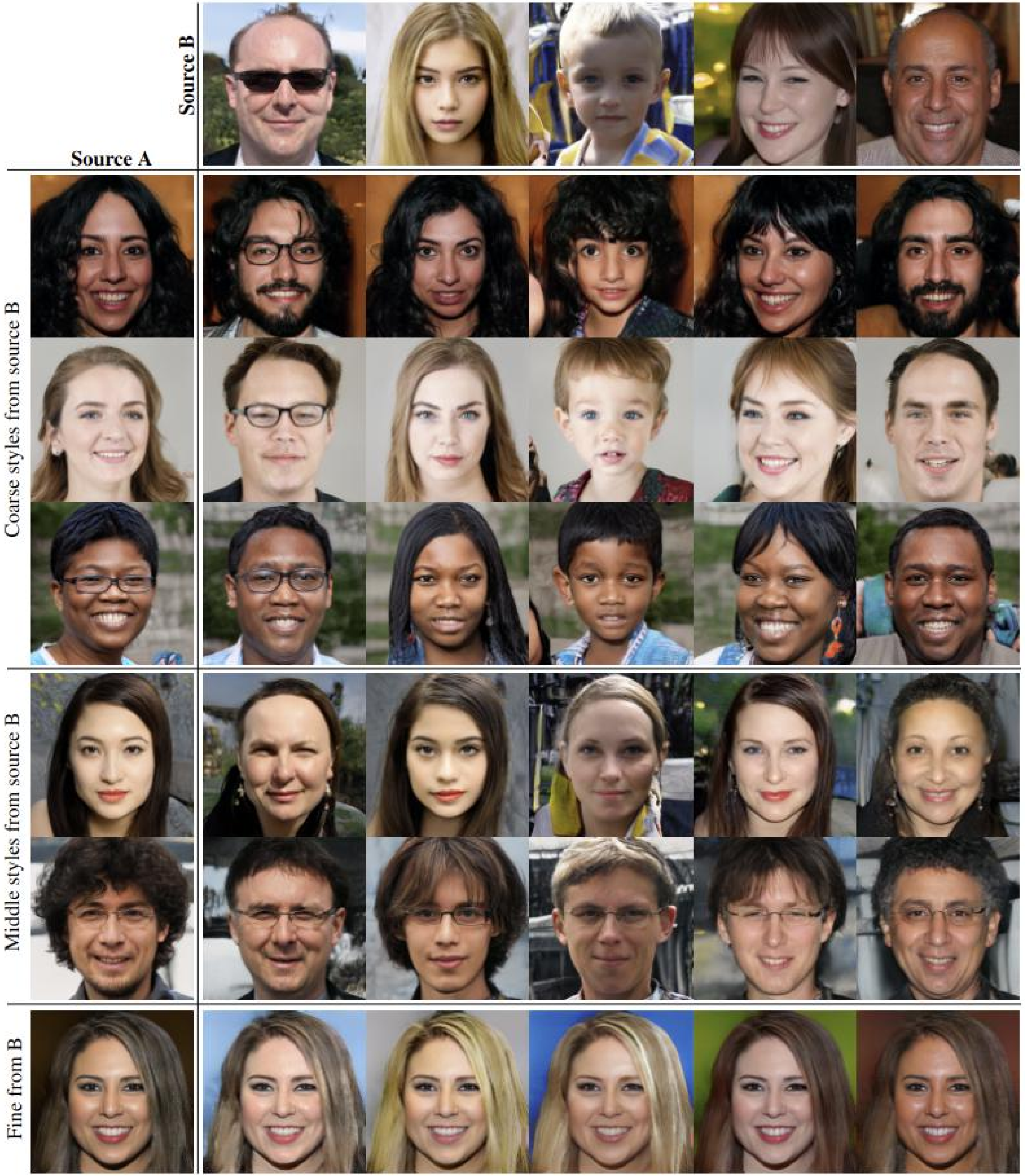


Image Translation - StyleGAN

Stochastic Variations

- Many features are stochastic in nature such as the exact placement of hairs, stubble, freckles, or skin pores
- Adding noise helps in maintaining stochasticity

(a) Noise is applied to all layers.
(b) No noise.
(c) Noise in fine layers only (64^2 – 1024^2).
(d) Noise in coarse layers only (4^2 – 32^2)

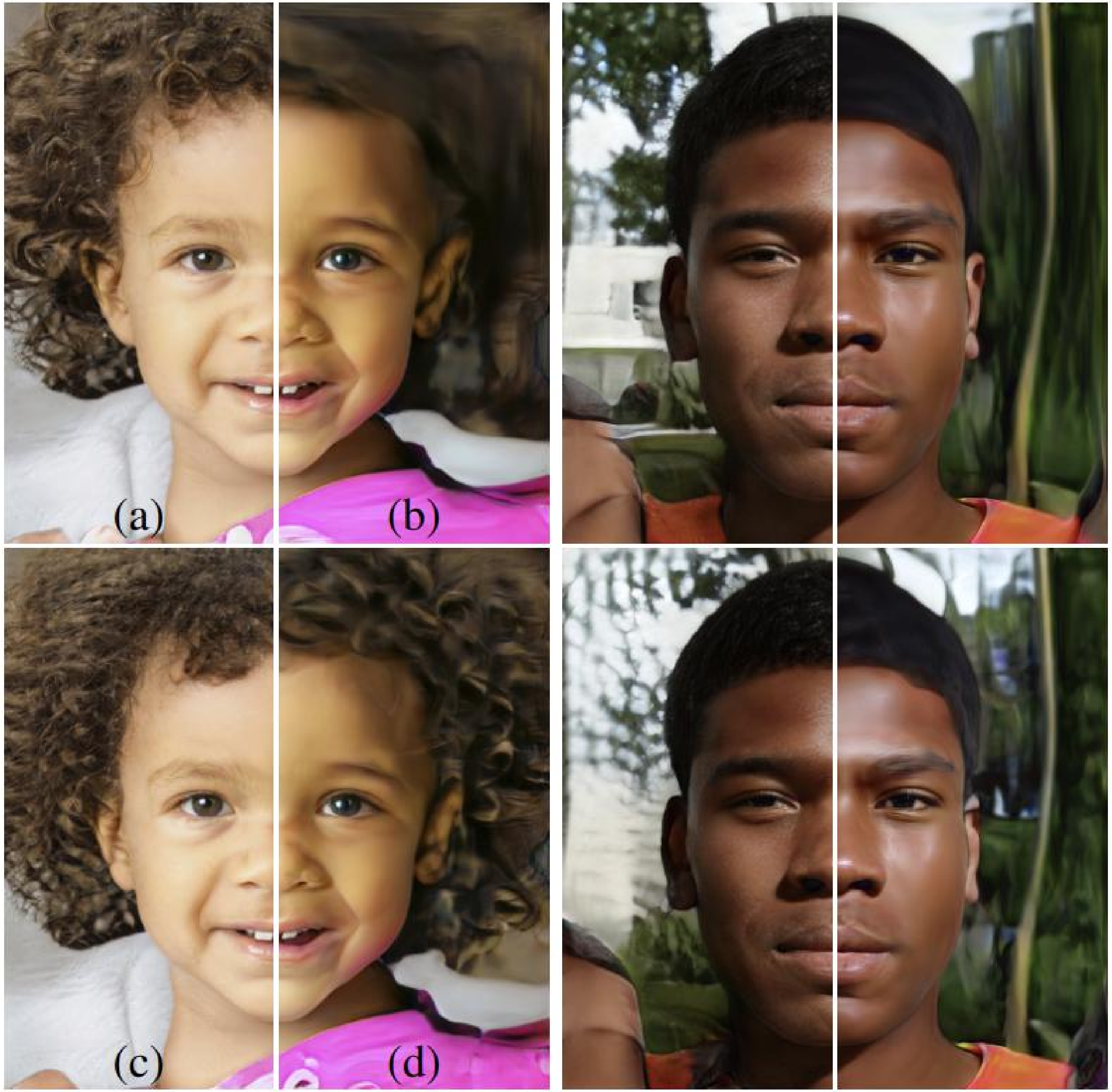


Image Translation - StyleGAN (Disentanglement Analysis)

- Common Definition : latent space that consists of linear subspaces, each of which controls one factor of variation
  - But StyleGAN doesn't explicitly learn factor of variations

- Perceptual Path Length and Linear Separability

Perceptually-based pairwise image distance : a weighted difference between two VGG16 embeddings

Entangled Space : Features that are absent in either endpoint may appear in the middle of a linear interpolation path

$$l_{\mathcal{Z}} = \mathbb{E}\Big[\frac{1}{\epsilon^2} d\big(G(\mathrm{slerp}(\mathbf{z}_1, \mathbf{z}_2;\, t)),\; G(\mathrm{slerp}(\mathbf{z}_1, \mathbf{z}_2;\, t+\epsilon))\big)\Big]$$

$\mathbf{z}_1, \mathbf{z}_2 \sim P(\mathbf{z}), t \sim U(0,1), G$ generator

$$\epsilon = 10^{-4}$$

$$l_{\mathcal{W}} = \mathbb{E}\Big[\frac{1}{\epsilon^2} d\big(g(\mathrm{lerp}(f(\mathbf{z}_1), f(\mathbf{z}_2);\, t)),\; g(\mathrm{lerp}(f(\mathbf{z}_1), f(\mathbf{z}_2);\, t+\epsilon))\big)\Big]$$

Less curved latent space should result in perceptually smoother transition than a highly curved latent space

Image Translation - StyleGAN (Disentanglement Analysis)

Method		Path length full	Path length end	Separa-bility
B Traditional generator	$\mathcal{Z}$	412.0	415.3	10.78
D Style-based generator	$\mathcal{W}$	446.2	376.6	3.61
E + Add noise inputs	$\mathcal{W}$	**200.5**	**160.6**	3.54
+ Mixing 50%	$\mathcal{W}$	231.5	182.1	**3.51**
F + Mixing 90%	$\mathcal{W}$	234.0	195.9	3.79

A low value suggests consistent latent space directions for the corresponding factor(s) of variation.

final separability score as $exp(\sum_i H(Y_i|X_i))$, where i enumerates the 40 attributes.

- If a latent space is sufficiently disentangled, it should be possible to find direction vectors that consistently correspond to individual factors of variation.
- Measuring how well the latent-space points can be separated into two distinct sets via a linear hyperplane,
  - ....each set corresponds to a specific binary attribute of the image.
- Train an auxilliary classifiers on 40 attributes from Celeba-HQ dataset
  generate 200,000 images with $\mathbf{z} \sim P(\mathbf{z})$
- Retain the half 100k latent-space vectors
- Fit a Linear SVM
- compute the conditional entropy H(Y |X) where X = classes predicted by the SVM and Y = classes predicted by the pre-trained classifier.

Normalizing Flows

Normalizing Flows - 1D Toy Example

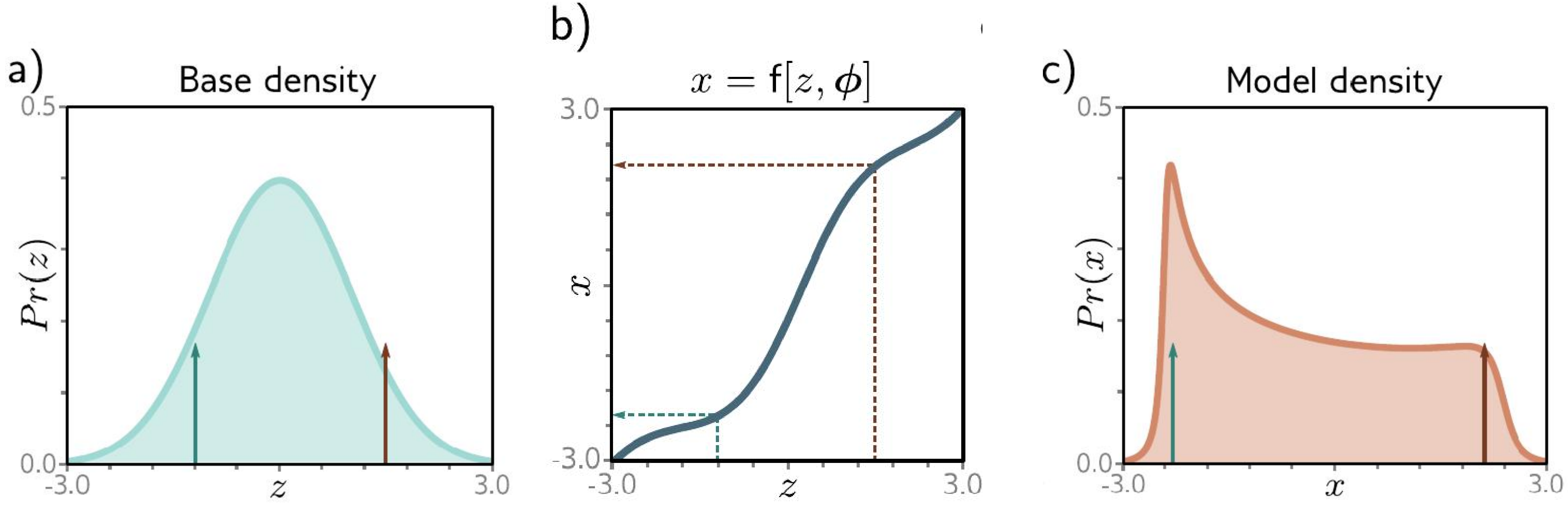
GAN

- Can generate new samples
- Evaluating the probability that the generated sample belongs to the same dataset isn’t straightforward

Normalizing Flows

- Probabilistic generative model
- Learns probability model by transformaing a simple distribution to a complex

Normalizing Flows - 1D Toy Example



Tractable base distribution

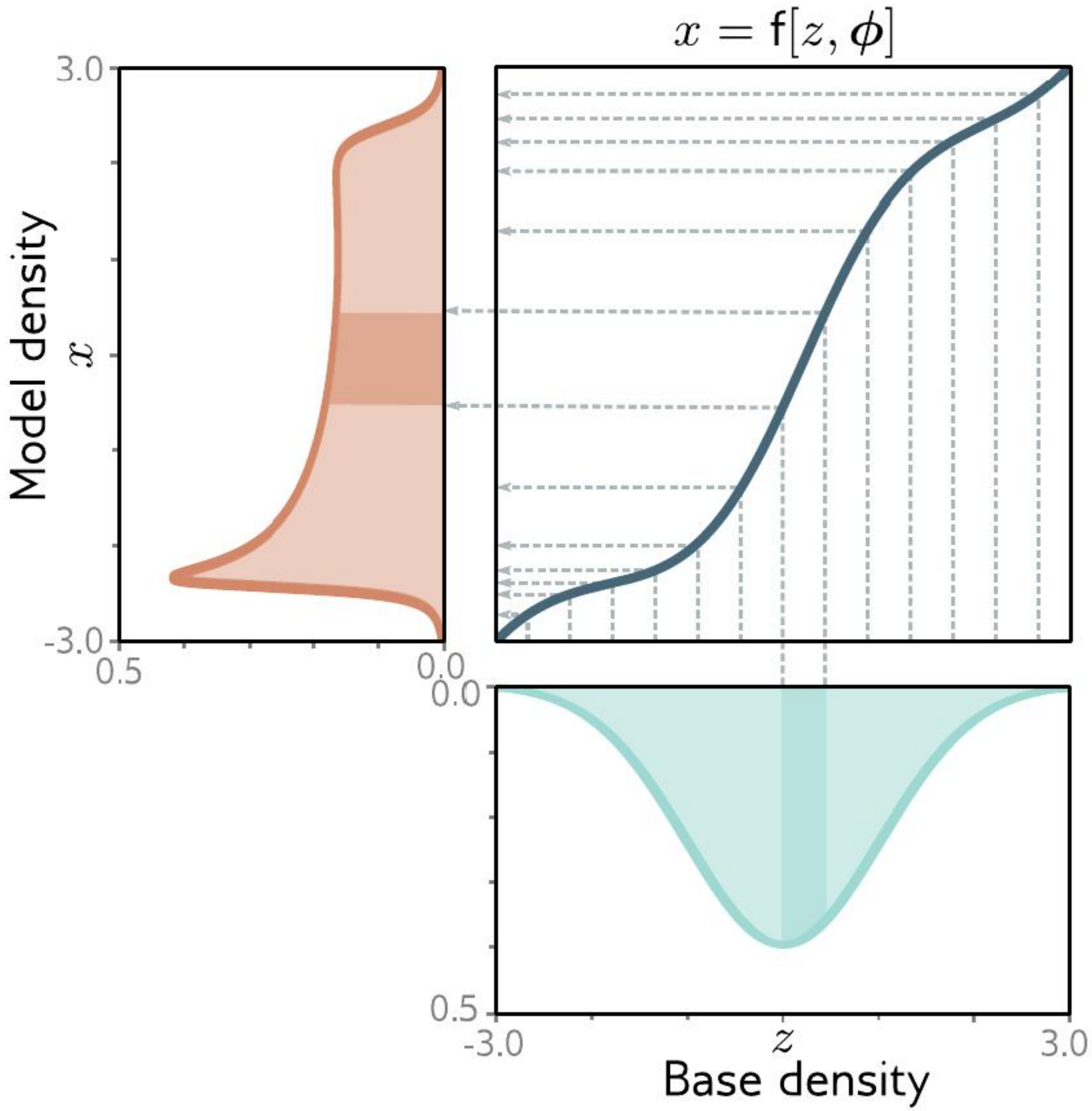
Normalizing Flows - 1D Toy Example

Probability of a data point *x* under transformed distribution

- The probability density will **decrease** in areas that are **stretched** by the function
  - since the area under the new distribution remains one
- The degree to which a function *f[z, ϕ]* stretches or compresses its input depends on the magnitude of its gradient

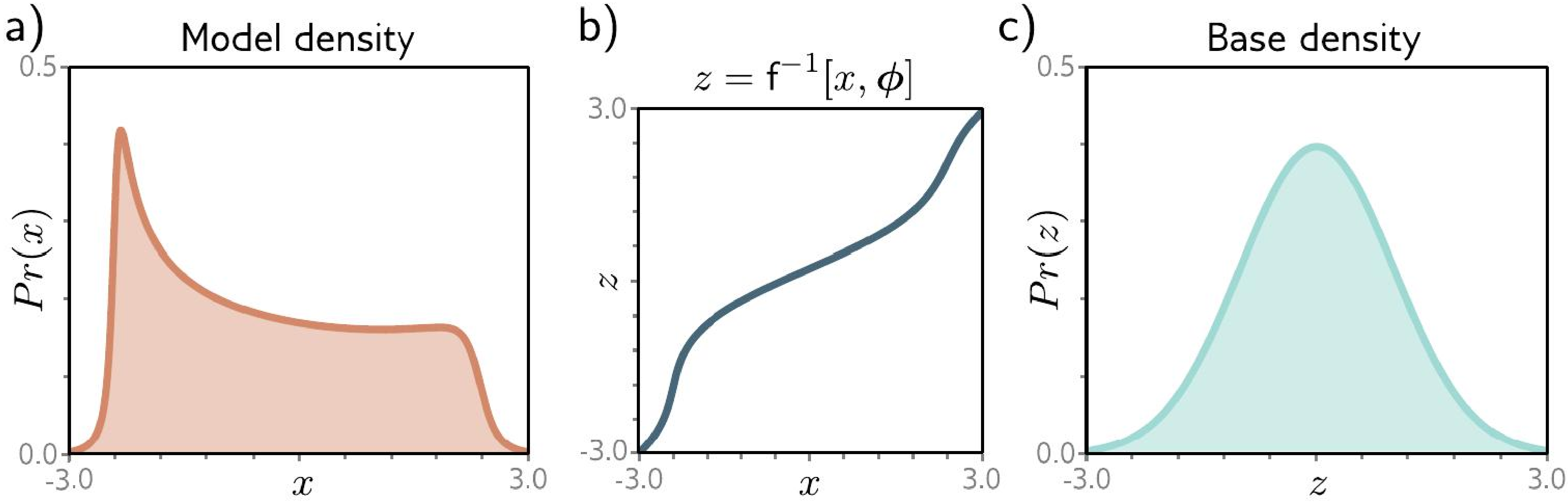
Magnitude of the derivative of the function

$$Pr(x|\phi) = \boxed{\left|\frac{\partial \mathrm{f}[z,\phi]}{\partial z}\right|^{-1}} \cdot Pr(z)$$



Normalizing Flows - 1D Toy Example

- The forward mapping (base density to model density) is called **generative direction** $\mathbf{x} = \mathbf{f}[\mathbf{z}, \phi]$
- The inverse mapping (model density to base density) is called **normalizing direction** $\mathbf{z} = \mathbf{f}^{-1}[\mathbf{x}, \phi]$
  - base density is the standard normal distribution



- To draw samples we need the forward mapping,
- To measure the likelihood, we need to compute the inverse $z = f^{-1}[x, \phi]$.

Normalizing Flows - Training Objective

Find parameters ϕ that maximize the likelihood of the training data or equivalently minimize the negative log-likelihood:

$$
\begin{aligned}
\hat{\phi} &= \underset{\phi}{\operatorname{argmax}}\left[\prod_{i=1}^{I} Pr(x_i|\phi)\right] \\
&= \underset{\phi}{\operatorname{argmin}}\left[\sum_{i=1}^{I} -\log\Big[Pr(x_i|\phi)\Big]\right] \\
&= \underset{\phi}{\operatorname{argmin}}\left[\sum_{i=1}^{I} \log\left[\left|\frac{\partial \mathrm{f}[z_i,\phi]}{\partial z_i}\right|\right] - \log\Big[Pr(z_i)\Big]\right]
\end{aligned}
$$

$$
Pr(x|\phi) = \left|\frac{\partial \mathrm{f}[z,\phi]}{\partial z}\right|^{-1} \cdot Pr(z)
$$

Normalizing Flows - General Scenario

- $Pr(\mathbf{z})$ multivariate base density
- $Pr(\mathbf{x})$ multivariate model density

Deep Neural Network

$$\mathbf{x} = \mathbf{f}[\mathbf{z}, \emptyset]$$

$$\mathbf{z} \in \mathrm{R}^{\mathrm{D}}$$

$$\mathbf{x} \in \mathrm{R}^{\mathrm{D}}$$

- How to get a new sample ?

$$\mathbf{z}^* \sim Pr(\mathbf{z})$$

$$\mathbf{x}^* = \mathbf{f}[\mathbf{z}^*, \emptyset]$$

$$Pr(\mathbf{x}|\phi) = \boxed{\left|\frac{\partial \mathbf{f}[\mathbf{z}, \phi]}{\partial \mathbf{z}}\right|^{-1}} \cdot Pr(\mathbf{z})$$

***D* × *D* Jacobian Matrix**

Forward Mapping

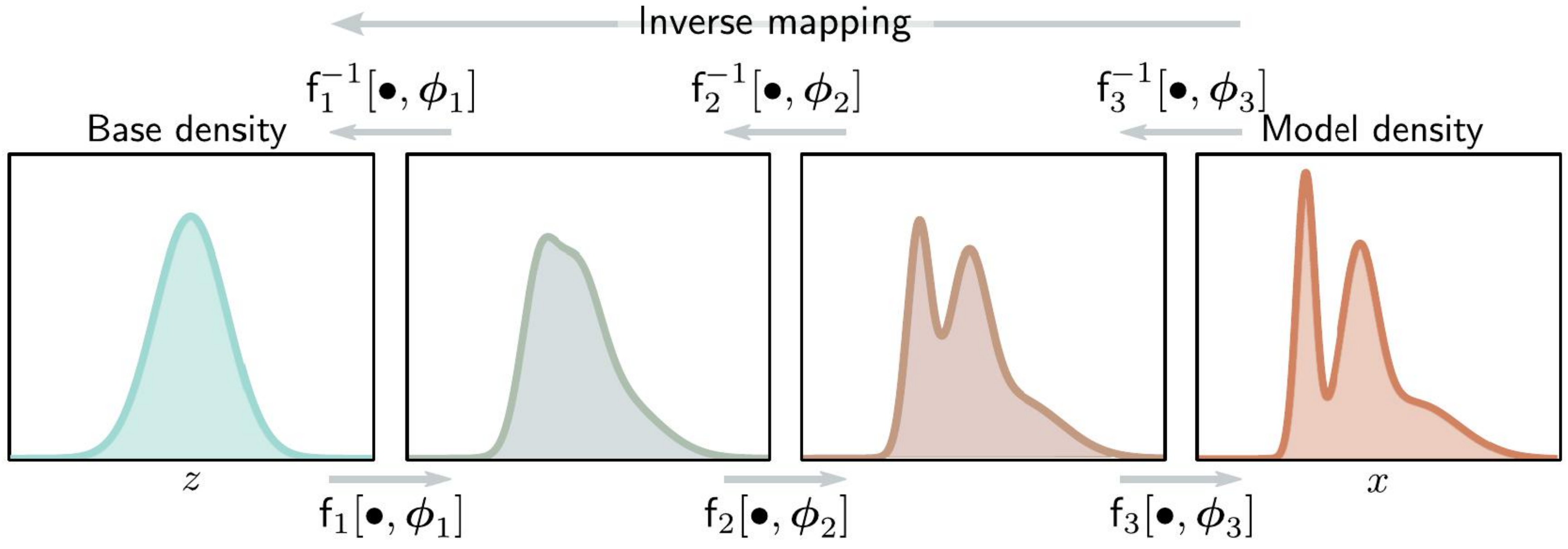
$$\mathbf{x} = \mathbf{f}[\mathbf{z}, \phi] = \mathbf{f}_K\left[\mathbf{f}_{K-1}\left[\ldots \mathbf{f}_2\left[\mathbf{f}_1[\mathbf{z}, \phi_1], \phi_2\right], \ldots \phi_{K-1}\right], \phi_K\right]$$

Inverse Mapping / Normalizing Direction

$$\mathbf{z} = \mathbf{f}^{-1}[\mathbf{x}, \phi] = \mathbf{f}_1^{-1}\left[\mathbf{f}_2^{-1}\left[\ldots \mathbf{f}_{K-1}^{-1}\left[\mathbf{f}_K^{-1}[\mathbf{x}, \phi_K], \phi_{K-1}\right], \ldots \phi_2\right], \phi_1\right]$$

Gradually **move/flow** data density towards **normal** distribution $Pr(\mathbf{z})$

Normalizing Flows - General Scenario



Forward Mapping

$$\mathbf{x} = \mathbf{f}[\mathbf{z}, \phi] = \mathbf{f}_K\left[\mathbf{f}_{K-1}\left[\ldots \mathbf{f}_2\left[\mathbf{f}_1[\mathbf{z}, \phi_1], \phi_2\right], \ldots \phi_{K-1}\right], \phi_K\right]$$

Inverse Mapping / Normalizing Direction

$$\mathbf{z} = \mathbf{f}^{-1}[\mathbf{x}, \phi] = \mathbf{f}_1^{-1}\left[\mathbf{f}_2^{-1}\left[\ldots \mathbf{f}_{K-1}^{-1}\left[\mathbf{f}_K^{-1}[\mathbf{x}, \phi_K], \phi_{K-1}\right], \ldots \phi_2\right], \phi_1\right]$$

Gradually **move/flow** data density towards **normal** distribution $Pr(\mathbf{z})$

Normalizing Flows - General Scenario (Training)

- Dataset $\{\mathbf{x}_i\}$
- **Training Objective** : Maximize the probabiliy of each data sample $\mathbf{x}_i$

$$\hat{\phi} = \underset{\phi}{\operatorname{argmax}} \left[\prod_{i=1}^{I} \boxed{Pr(\mathbf{z}_i) \cdot \left| \frac{\partial \mathbf{f}[\mathbf{z}_i, \phi]}{\partial \mathbf{z}_i} \right|^{-1}} \right]$$

$$= \underset{\phi}{\operatorname{argmin}} \left[\sum_{i=1}^{I} \log \left[\left| \frac{\partial \mathbf{f}[\mathbf{z}_i, \phi]}{\partial \mathbf{z}_i} \right| \right] - \log \left[Pr(\mathbf{z}_i) \right] \right]$$

Negative log-likelihood

$$\frac{\partial \mathbf{f}[\mathbf{z}, \phi]}{\partial \mathbf{z}} = \frac{\partial \mathbf{f}_K[\mathbf{f}_{K-1}, \phi_K]}{\partial \mathbf{f}_{K-1}} \cdot \frac{\partial \mathbf{f}_{K-1}[\mathbf{f}_{K-2}, \phi_{K-1}]}{\partial \mathbf{f}_{K-2}} \cdots \frac{\partial \mathbf{f}_2[\mathbf{f}_1, \phi_2]}{\partial \mathbf{f}_1} \cdot \frac{\partial \mathbf{f}_1[\mathbf{z}, \phi_1]}{\partial \mathbf{z}}$$

$$\left| \frac{\partial \mathbf{f}[\mathbf{z}, \phi]}{\partial \mathbf{z}} \right| = \left| \frac{\partial \mathbf{f}_K[\mathbf{f}_{K-1}, \phi_K]}{\partial \mathbf{f}_{K-1}} \right| \cdot \left| \frac{\partial \mathbf{f}_{K-1}[\mathbf{f}_{K-2}, \phi_{K-1}]}{\partial \mathbf{f}_{K-2}} \right| \cdots \left| \frac{\partial \mathbf{f}_2[\mathbf{f}_1, \phi_2]}{\partial \mathbf{f}_1} \right| \cdot \left| \frac{\partial \mathbf{f}_1[\mathbf{z}, \phi_1]}{\partial \mathbf{z}} \right|$$

Normalizing Flows - Invertible Layers

Linear Flows : $\mathbf{f}[\mathbf{h}] = \boldsymbol{\beta} + \boxed{\boldsymbol{\omega}}\mathbf{h}$

Invertible Matrix

- expensive, not sufficiently expressive
- normal to normal mapping (**input is normally distributed**)
- difficult to map to arbitrary density using linear flows alone

Elementwise/Nonlinear Flow: $\mathbf{f}[\mathbf{h}] = \left[\mathrm{f}[h_1, \boldsymbol{\phi}], \boxed{\mathrm{f}[h_2, \boldsymbol{\phi}]}, \ldots \mathrm{f}[h_D, \boldsymbol{\phi}]\right]^T$

pointwise nonlinear function

- could be fixed nonlinearity (leaky ReLU)
- parametric one-to-one mapping

$$\left|\boxed{\frac{\partial \mathbf{f}[\mathbf{h}]}{\partial \mathbf{h}}}\right| = \prod_{d=1}^{D} \left|\frac{\partial \mathrm{f}[h_d]}{\partial h_d}\right|$$

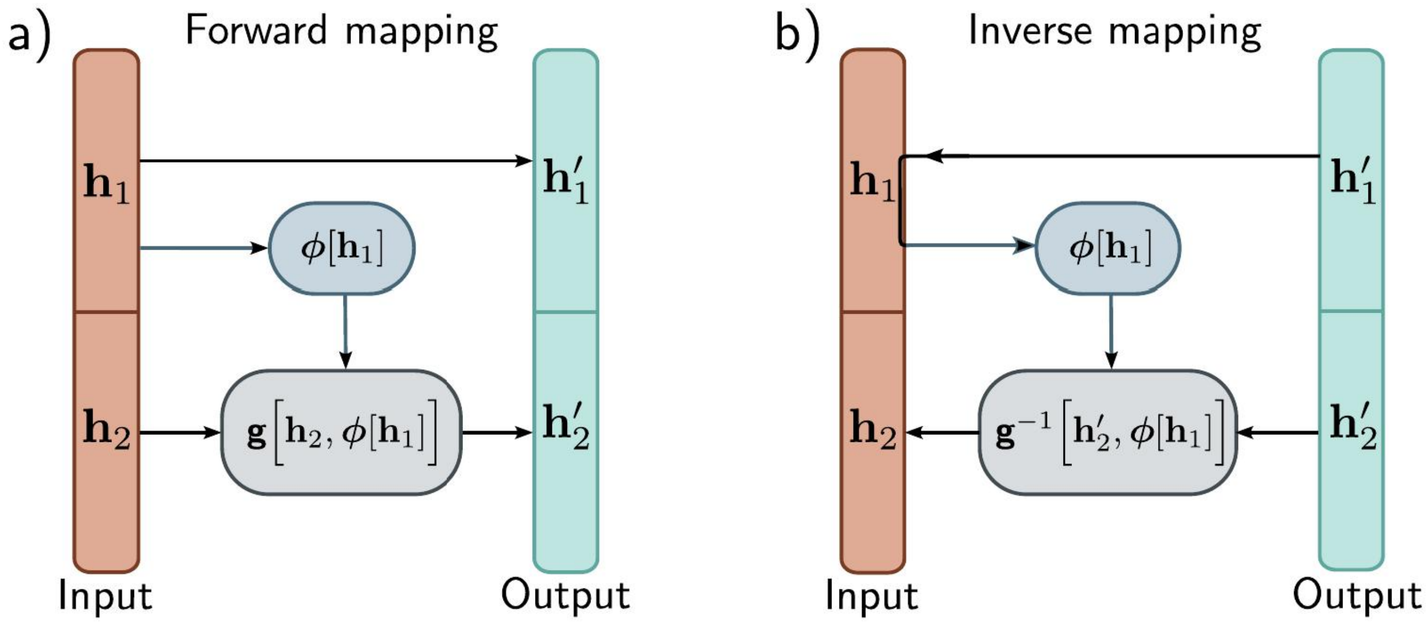
jacobian is a diagonal

(since d^{th} input only affects d^{th} output)

Normalizing Flows - Invertible Layers

- Elementwise flows are nonlinear but don't mix input dimensions,
- Can't capture correlations between variables.

- Elementwise flows alternated with linear flows can be used to model more complex transformations



Coupling Flows :

$$\mathbf{h} = [\mathbf{h}_1^T, \mathbf{h}_2^T]^T$$

$$\begin{aligned} \mathbf{h}'_1 &= \mathbf{h}_1 \\ \mathbf{h}'_2 &= \mathbf{g}\Big[\mathbf{h}_2, \phi[\mathbf{h}_1]\Big] \end{aligned}$$

$$\begin{aligned} \mathbf{h}_1 &= \mathbf{h}'_1 \\ \mathbf{h}_2 &= \mathbf{g}^{-1}\Big[\mathbf{h}'_2, \phi[\mathbf{h}_1]\Big] \end{aligned}$$

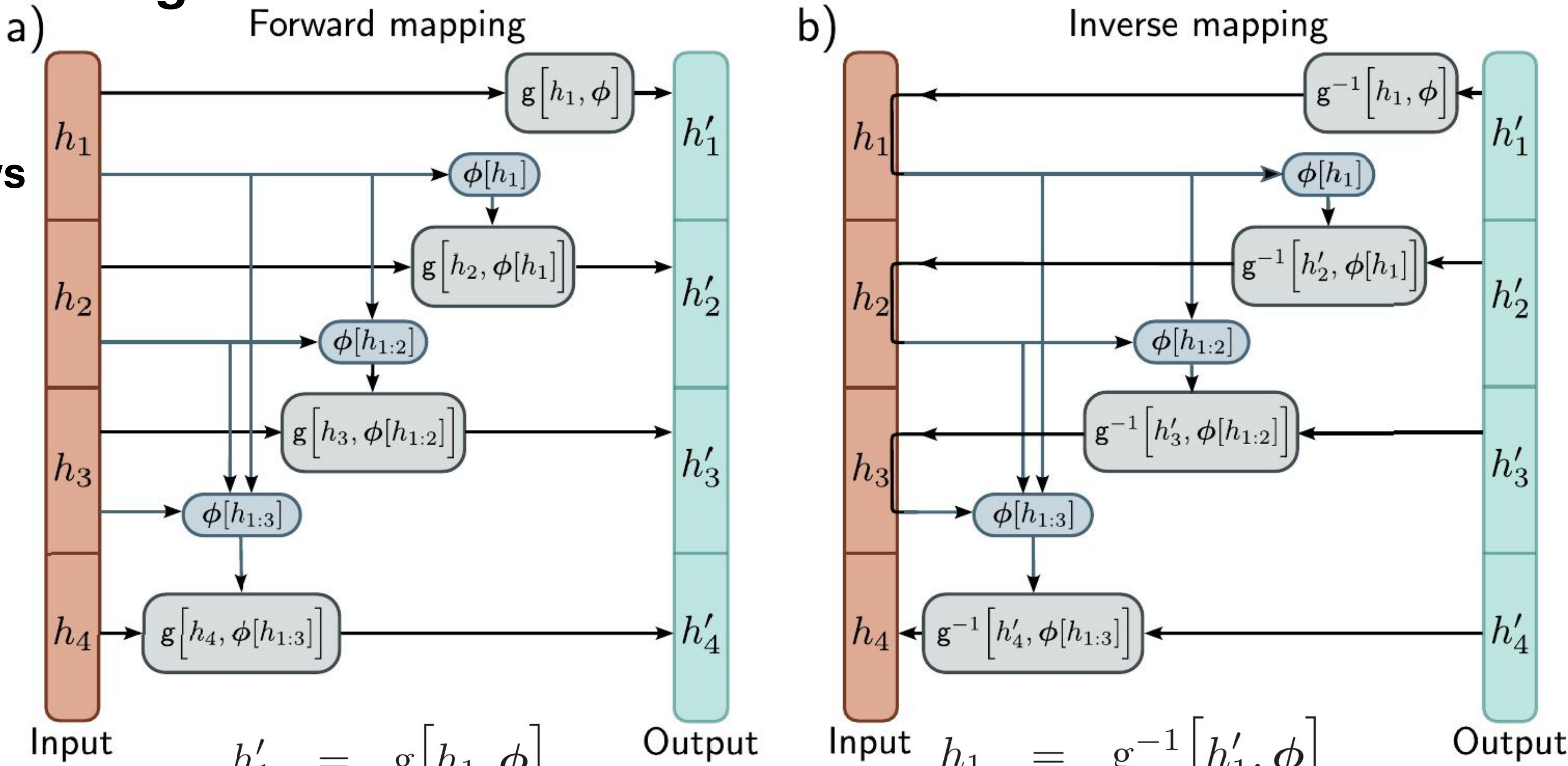
- To make a more general transformation,
  - the elements of ***h*** are randomly shuffled using permutation matrices between layers,
  - every variable is ultimately transformed by every other

e.g., images, the channels are divided and permuted between layers using 1×1 convolutions

Normalizing Flows - Autoregressive Flows

Autoregressive flows are a **generalization of coupling flows** that treat each input dimension as a separate “block

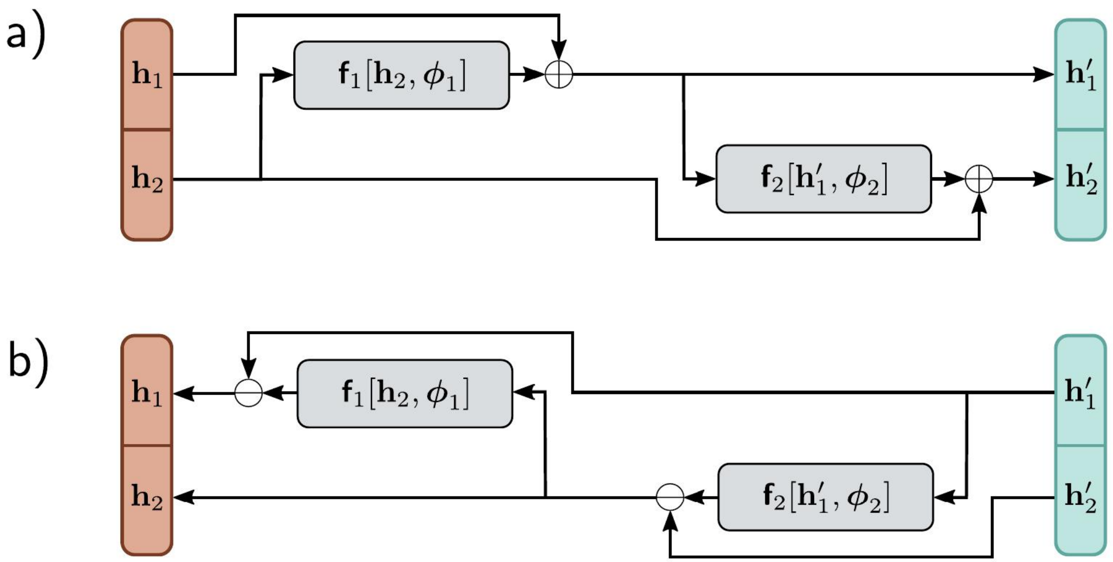
$$h'_d = \mathrm{g}\Big[h_d, \boldsymbol{\phi}[\mathbf{h}_{1:d-1}]\Big]$$



$$\begin{aligned} h'_1 &= \mathrm{g}\Big[h_1, \boldsymbol{\phi}\Big] \\ h'_2 &= \mathrm{g}\Big[h_2, \boldsymbol{\phi}[h_1]\Big] \\ h'_3 &= \mathrm{g}\Big[h_3, \boldsymbol{\phi}[h_{1:2}]\Big] \\ h'_4 &= \mathrm{g}\Big[h_4, \boldsymbol{\phi}[h_{1:3}]\Big] \end{aligned}$$

$$\begin{aligned} h_1 &= \mathrm{g}^{-1}\Big[h'_1, \boldsymbol{\phi}\Big] \\ h_2 &= \mathrm{g}^{-1}\Big[h'_2, \boldsymbol{\phi}[h_1]\Big] \\ h_3 &= \mathrm{g}^{-1}\Big[h'_3, \boldsymbol{\phi}[h_{1:2}]\Big] \\ h_4 &= \mathrm{g}^{-1}\Big[h'_4, \boldsymbol{\phi}[h_{1:3}]\Big] \end{aligned}$$

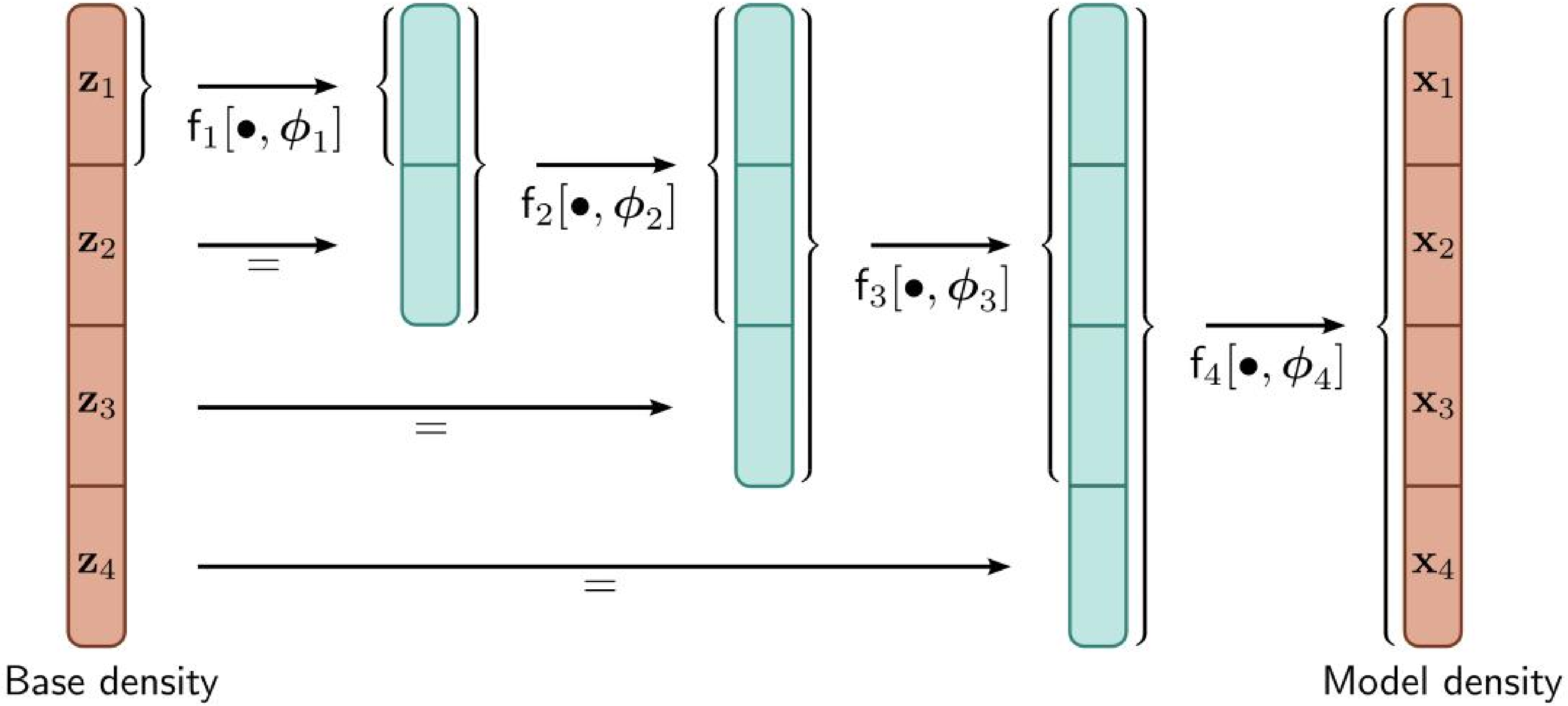
Normalizing Flows - Residual Flows



$$
\begin{aligned}
\mathbf{h}'_1 &= \mathbf{h}_1 + \mathbf{f}_1[\mathbf{h}_2, \phi_1] \\
\mathbf{h}'_2 &= \mathbf{h}_2 + \mathbf{f}_2[\mathbf{h}'_1, \phi_2],
\end{aligned}
$$

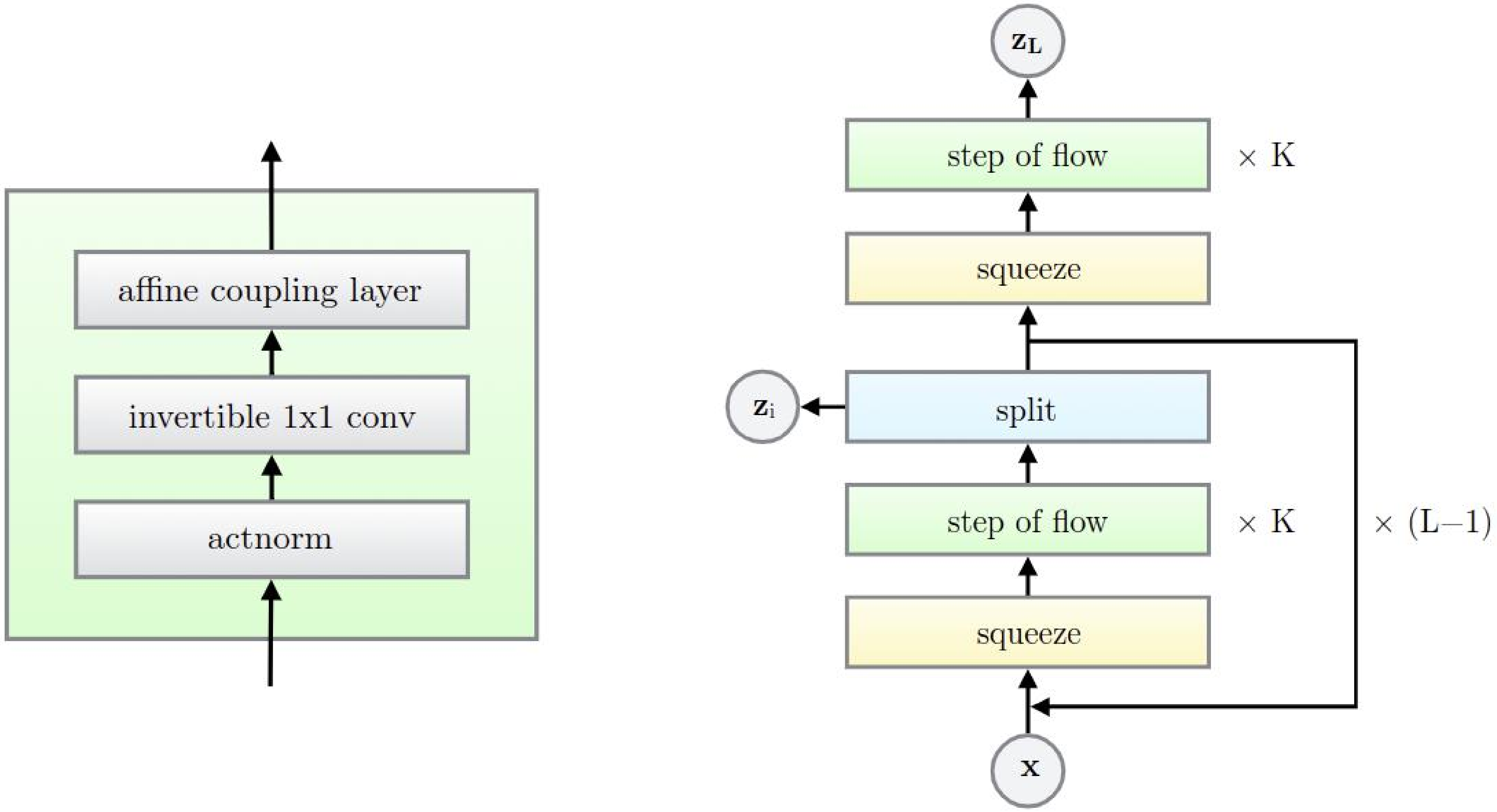
$$
\begin{aligned}
\mathbf{h}_2 &= \mathbf{h}'_2 - \mathbf{f}_2[\mathbf{h}'_1, \phi_2] \\
\mathbf{h}_1 &= \mathbf{h}'_1 - \mathbf{f}_1[\mathbf{h}_2, \phi_1]
\end{aligned}
$$

Normalizing Flows - Multi-Scale Flows



Normalizing Flows - Glow

Affine coupling
+
Multi-scale



(a) One step of our flow.

(b) Multi-scale architecture (Dinh et al., 2016).

Normalizing Flows - Glow

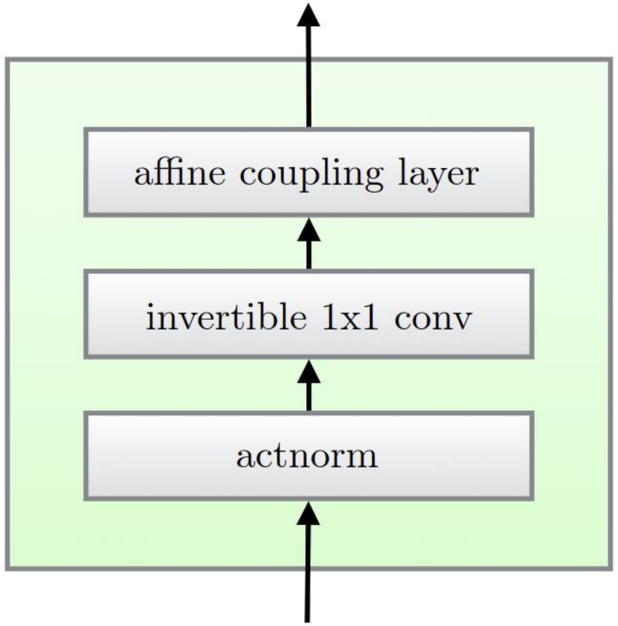
Table 1: The three main components of our proposed flow, their reverses, and their log-determinants. Here, $\mathbf{x}$ signifies the input of the layer, and $\mathbf{y}$ signifies its output. Both $\mathbf{x}$ and $\mathbf{y}$ are tensors of shape $[h \times w \times c]$ with spatial dimensions (h, w) and channel dimension c. With (i, j) we denote spatial indices into tensors $\mathbf{x}$ and $\mathbf{y}$. The function $\mathtt{NN}()$ 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.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i,j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \mathtt{sum}(\log \lvert\mathbf{s}\rvert)$
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 $h \cdot w \cdot \mathtt{sum}(\log \lvert\mathbf{s}\rvert)$ (see eq. (10))
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \mathtt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \mathtt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \mathtt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \mathtt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\mathtt{sum}(\log(\lvert\mathbf{s}\rvert))$

Normalizing Flows - Glow

actnorm layer

- performs an affine transformation of the activations using a scale and bias parameter per channel, similar to batch normalization.
- These parameters are initialized such that the post-actnorm activations per-channel have **zero mean** and **unit variance** given an initial minibatch of data.
- This is a form of data dependent initialization



Normalizing Flows - Glow

Invertible 1x1 convolution

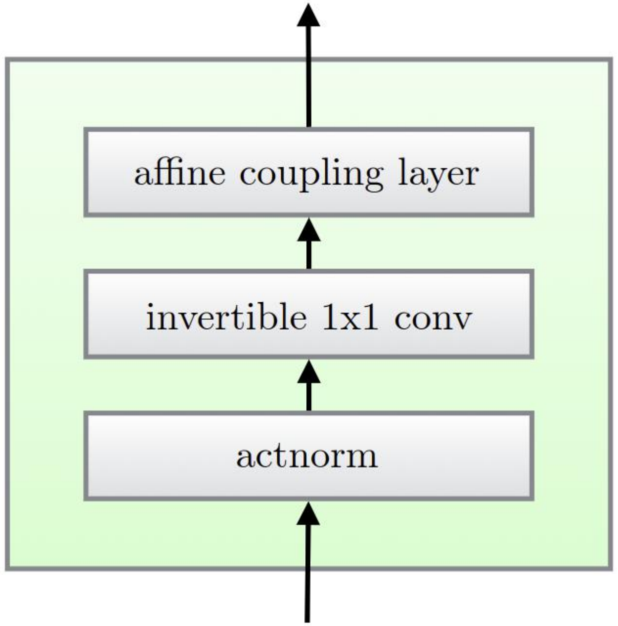
- Invertible 1 × 1 convolution, where the weight matrix is initialized as a random rotation matrix.
- 1 × 1 convolution with equal number of input and output channels is a generalization of a permutation operation

The log-determinant of an invertible 1 × 1 convolution of a *h* × *w* × *c* tensor ***h*** with c × c weight matrix **W** is

$$\log \left| \det \left(\frac{d\, \mathtt{conv2D}(\mathbf{h}; \mathbf{W})}{d\, \mathbf{h}} \right) \right| = h \cdot w \cdot \log |\det(\mathbf{W})|$$

$$\mathbf{W} = \mathbf{PL}(\mathbf{U} + \mathrm{diag}(\mathbf{s}))$$

$$\log |\det(\mathbf{W})| = \mathtt{sum}(\log |\mathbf{s}|)$$



Normalizing Flows - Glow

Invertible 1x1 convolution

- Invertible 1×1 convolution, where the weight matrix is initialized as a random rotation matrix.
- 1×1 convolution with equal number of input and output channels is a generalization of a permutation operation

The log-determinant of an invertible 1×1 convolution of a $h \times w \times c$ tensor $\boldsymbol{h}$ with $c \times c$ weight matrix $\mathbf{W}$ is

$$\log \left| \det \left(\frac{d\, \texttt{conv2D}(\mathbf{h}; \mathbf{W})}{d\, \mathbf{h}} \right) \right| = h \cdot w \cdot \log |\det(\mathbf{W})|$$

$$\mathbf{W} = \mathbf{PL}(\mathbf{U} + \mathrm{diag}(\mathbf{s}))$$

$$\log |\det(\mathbf{W})| = \texttt{sum}(\log |\mathbf{s}|)$$

```python
class Invertible1x1Conv(nn.Module):

    def __init__(self, dim):
        super().__init__()
        self.dim = dim
        Q = torch.nn.init.orthogonal_(torch.randn(dim, dim))
        P, L, U = torch.lu_unpack(*Q.lu())
        self.P = P
        self.L = nn.Parameter(L)
        self.S = nn.Parameter(U.diag())
        self.U = nn.Parameter(torch.triu(U, diagonal=1))

    def _assemble_W(self):
        """ assemble W from its pieces (P, L, U, S) """
        L = torch.tril(self.L, diagonal=-1) + torch.diag(torch.ones(self.dim))
        U = torch.triu(self.U, diagonal=1)
        W = self.P @ L @ (U + torch.diag(self.S))
        return W

    def forward(self, x):
        W = self._assemble_W()
        z = x @ W
        log_det = torch.sum(torch.log(torch.abs(self.S)))
        return z, log_det

    def backward(self, z):
        W = self._assemble_W()
        W_inv = torch.inverse(W)
        x = z @ W_inv
        log_det = -torch.sum(torch.log(torch.abs(self.S)))
        return x, log_det
```

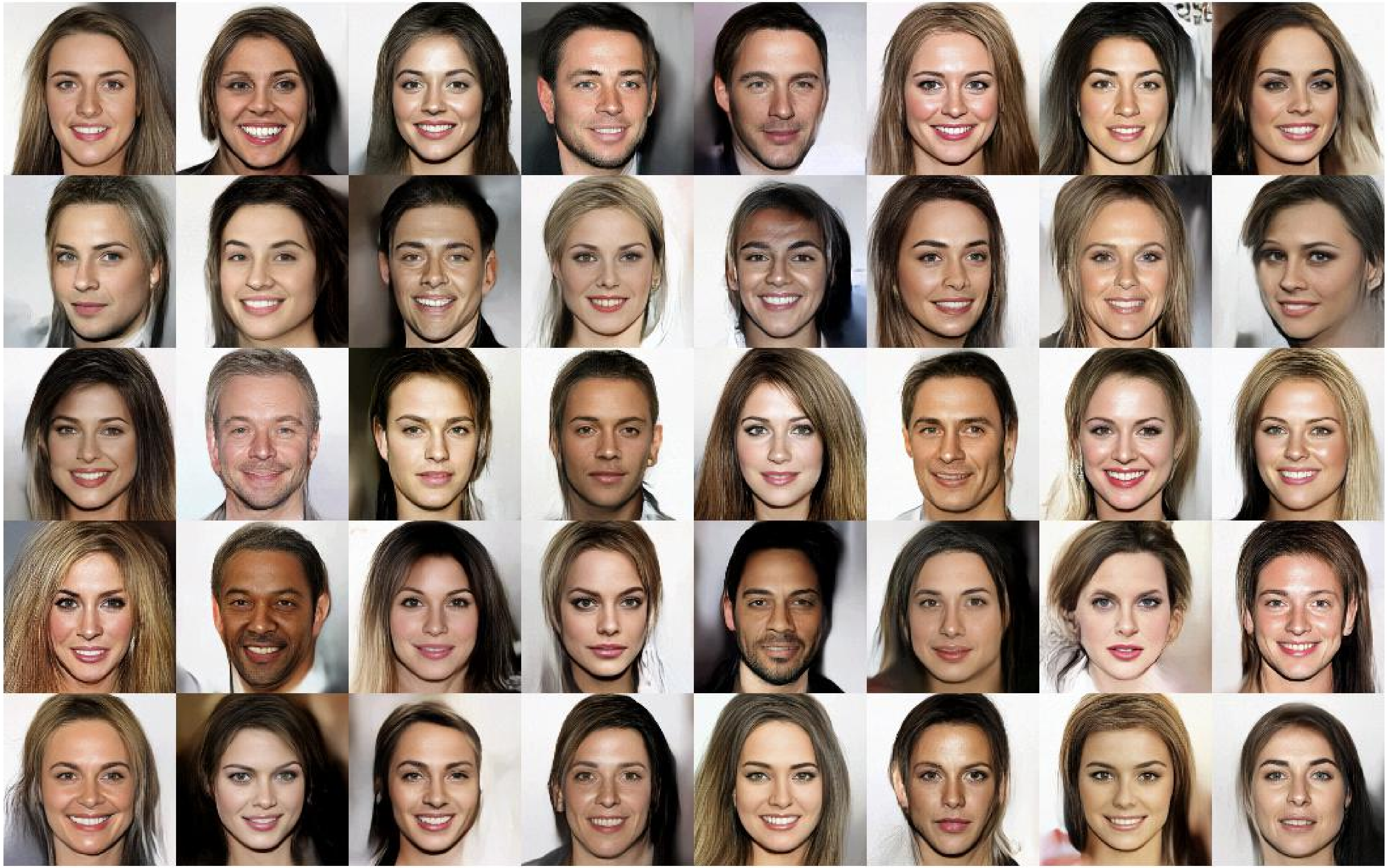
Normalizing Flows - Glow

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)	$\mathbf{x}_a, \mathbf{x}_b = \texttt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \texttt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \texttt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t}) / \mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \texttt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\texttt{sum}(\log(\lvert\mathbf{s}\rvert))$

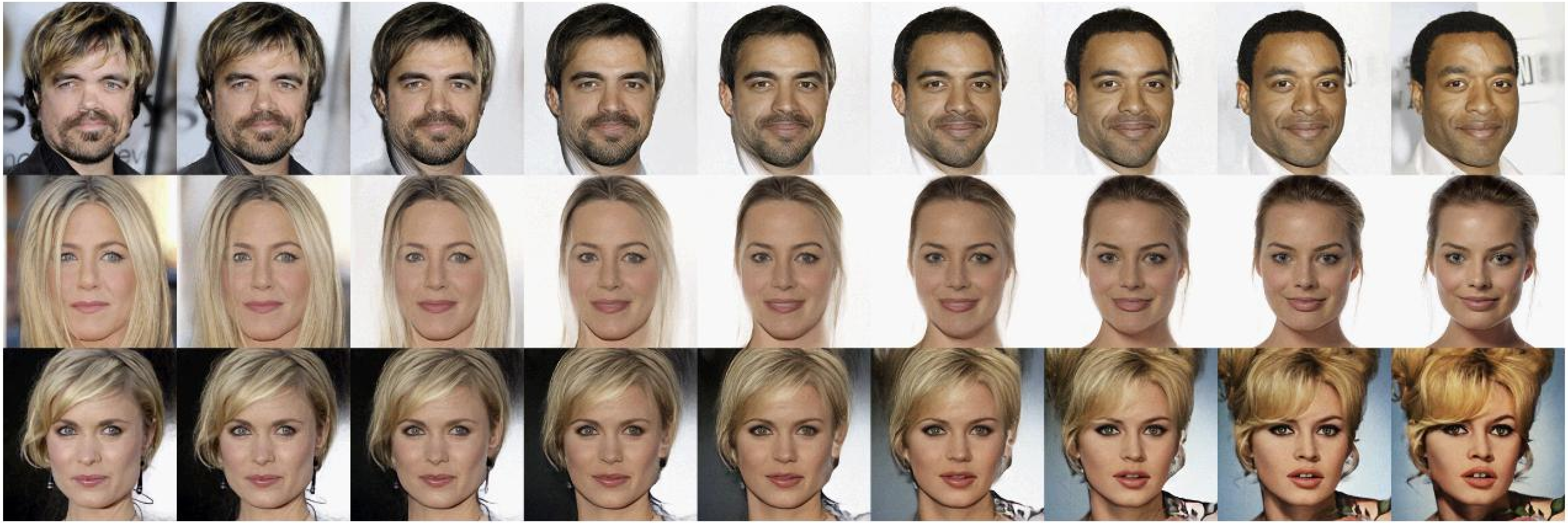
Normalizing Flows - Glow

Random Results



Normalizing Flows - Glow

Linear interpolation in latent space between real images



Normalizing Flows - Glow

Semantic Manipulation

(a) Smiling

(b) Pale Skin

(c) Blond Hair

(d) Narrow Eyes

(e) Young

(f) Male