Introduction to Flow-based Generative Models

Ana Díaz Rivero

Machine Learning Journal Club 05/20/2019

Outline

1. Introduction to Generative Models
2. Normalizing Flows
3. NICE, RealNVP and Glow
4. An example: generating weak lensing maps with Glow

Introduction to Generative Models

What are generative models?

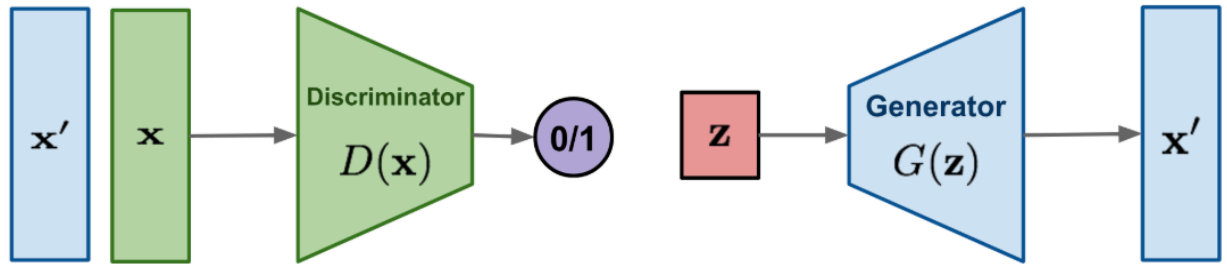
Generative models are a class of unsupervised ML techniques whose goal is to learn the underlying probability distribution $p_{\mathrm{X}}(\mathbf{x})$ of a given data set $\mathbf{x}$, such that new samples that are indistinguishable from the original data can be generated.

For example, a generative model trained on images of human faces can generate images that look like realistic human faces.

Examples of generative models

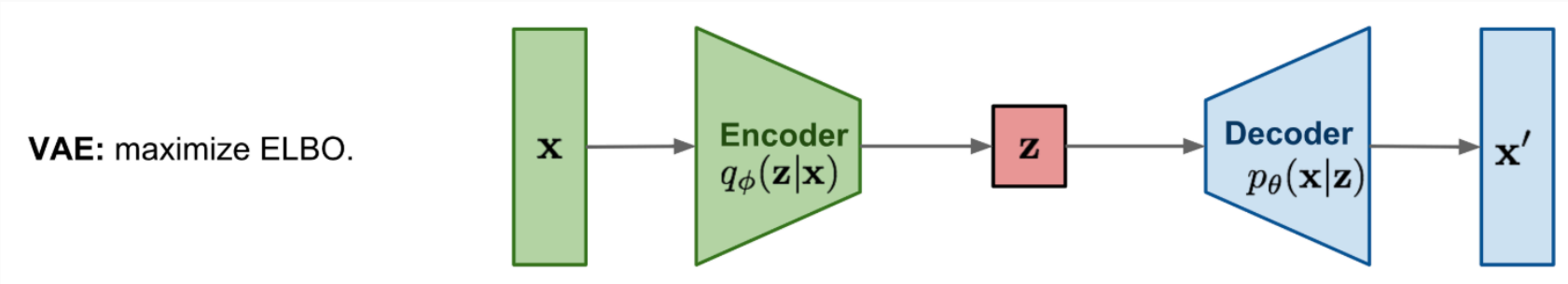
Generative Adversarial Networks (GANs, Goodfellow et al. 2014):

GAN: minimax the classification error loss.



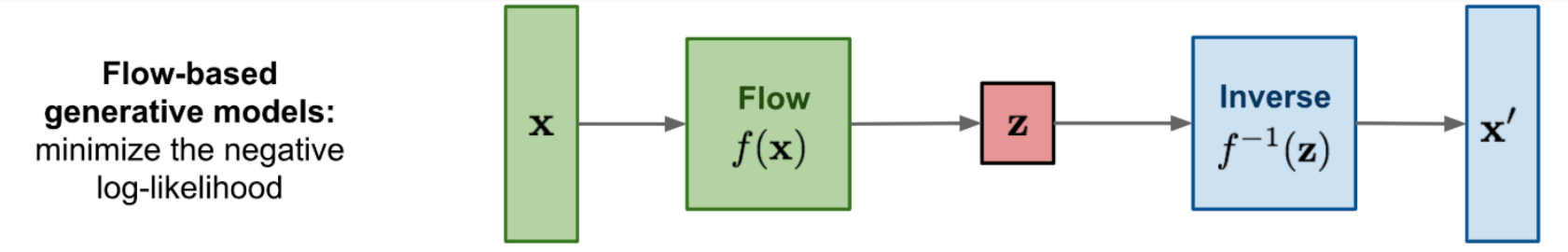
Examples of generative models

Variational Autoencoders (VAEs): optimize the log-likelihood of the data indirectly by maximizing the ELBO.



Examples of generative models

Flow-based models: explicitly optimize the exact log-likelihood.



Normalizing Flows

Change of variables theorem for probability distributions

Problem: Given a random variable $\mathbf{z}$ and its known probability density function $\mathbf{z} \sim \pi(\mathbf{z})$, we would like to construct a new random variable using a 1-1 invertible mapping function $\mathbf{x} = f(\mathbf{z})$. **How do we infer the distribution of the new variable x?**

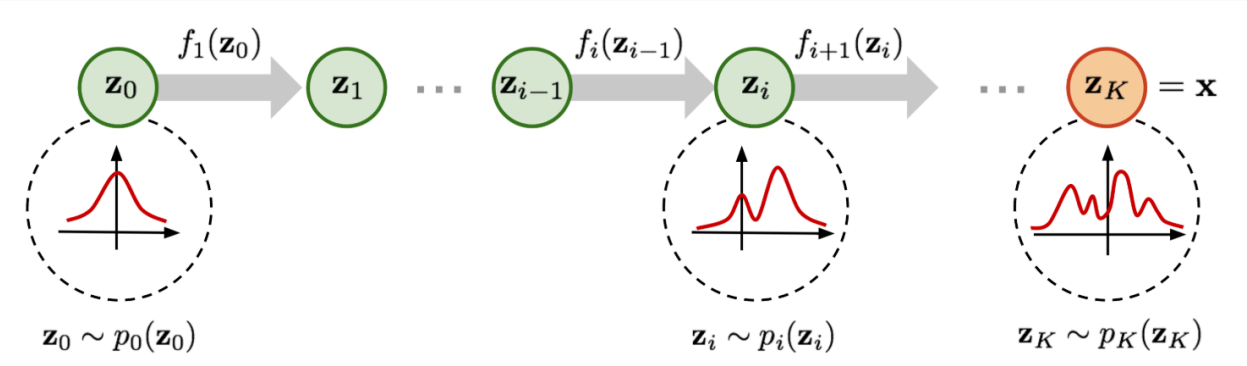
Change of variables theorem for probability distributions

$$\int p(\mathbf{x})d\mathbf{x} = \int \pi(\mathbf{z})d\mathbf{z} = 1 \tag{1}$$

$$\begin{aligned} p(\mathbf{x}) &= \pi(\mathbf{z})\left|\det \frac{d\mathbf{z}}{d\mathbf{x}}\right| \\ &= \pi(f^{-1}(\mathbf{x}))\left|\det \frac{df^{-1}}{d\mathbf{x}}\right| \qquad (2) \\ &= \pi(f^{-1}(\mathbf{x})) = \left|\det (f^{-1})'(\mathbf{x})\right| \end{aligned}$$

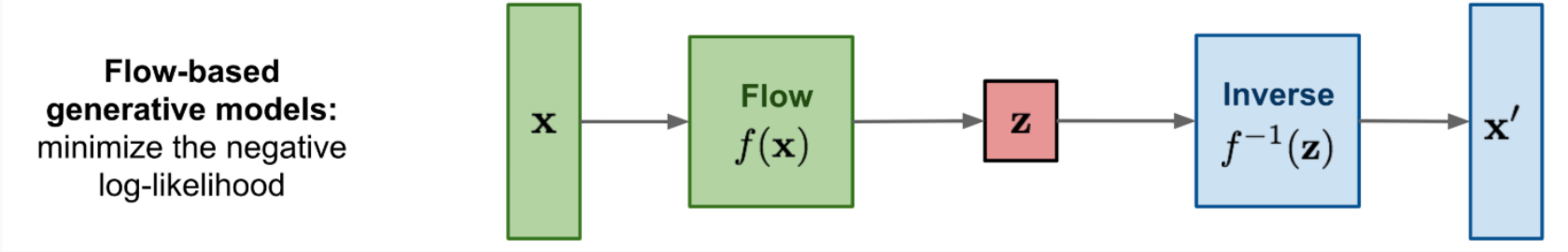
Normalizing flows

A normalizing flow transforms a simple probability density into an arbitrarily complex one by applying a sequence of transformations.



$$
\begin{aligned}
\mathbf{x} = \mathbf{z}_K &= f_K \circ f_{K-1} \circ \cdots \circ f_1(\mathbf{z}_0) \\
\log p(\mathbf{x}) = \log \pi_K(\mathbf{z}_K) &= \log \pi_{K-1}(\mathbf{z}_{K-1}) - \log \left| \det \frac{df_K}{d\mathbf{z}_{K-1}} \right| \\
&= \log \pi_{K-2}(\mathbf{z}_{K-2}) - \log \left| \det \frac{df_{K-1}}{d\mathbf{z}_{K-2}} \right| - \log \left| \det \frac{df_K}{d\mathbf{z}_{K-1}} \right| \\
&= \dots \\
&= \log \pi_0(\mathbf{z}_0) - \sum_{i=1}^{K} \log \left| \det \frac{df_i}{d\mathbf{z}_{i-1}} \right|
\end{aligned}
$$

Flow-based generative models



The generative process is defined as

$$\mathbf{z} \sim p_Z(\mathbf{z}) \tag{3}$$

$$\mathbf{x} = f^{-1}(\mathbf{z}). \tag{4}$$

The latent-variable inference is done by

$$\mathbf{z} = f(\mathbf{x}). \tag{5}$$

We have an exact form of the log-likelihood! So the loss function is simply the negative log-likelihood over the training set:

$$\mathcal{L}(\mathcal{D}) = -\frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \log p(\mathbf{x}) \tag{6}$$

Flow-based generative models

For these models to be useable, the transformation function must

1. be easily invertible
2. have an easy-to-compute Jacobian determinant

NICE, RealNVP and Glow

Non-linear Independent Component Estimation (NICE, Dinh et al. 2015)

By having the transformation split up the input dimensions into two parts the Jacobian is a triangular matrix, making the determinant easy to compute (and, in this case, trivial):

$$\begin{cases} \mathbf{y}_{1:d} & = \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} & = \mathbf{x}_{d+1:D} + m(\mathbf{x}_{1:d}) \end{cases} \Leftrightarrow \begin{cases} \mathbf{x}_{1:d} & = \mathbf{y}_{1:d} \\ \mathbf{x}_{d+1:D} & = \mathbf{y}_{d+1:D} - m(\mathbf{y}_{1:d}) \end{cases}$$

Real NVP (Dinh et al. 2017)

A generalization of NICE that includes a scaling in the transformation layers.

$$\begin{cases} \mathbf{y}_{1:d} & = \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} & = \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d}) \end{cases} \Leftrightarrow \begin{cases} \mathbf{x}_{1:d} & = \mathbf{y}_{1:d} \\ \mathbf{x}_{d+1:D} & = (\mathbf{y}_{d+1:D} - t(\mathbf{y}_{1:d})) \odot \exp(-s(\mathbf{y}_{1:d})) \end{cases}$$

$$\mathbf{J} = \begin{bmatrix} \mathbb{I}_d & \mathbf{0}_{d\times(D-d)} \\ \frac{\partial \mathbf{y}_{d+1:D}}{\partial \mathbf{x}_{1:d}} & \operatorname{diag}(\exp(s(\mathbf{x}_{1:d}))) \end{bmatrix}$$

Real NVP (Dinh et al. 2017)

Notice that

1. inverting the function does not require inverting s or t,
2. computing the Jacobian determinant is easy,

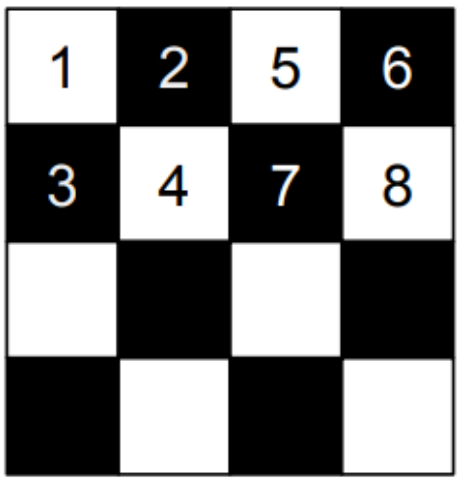
$$\det(\mathbf{J}) = \prod_{j=1}^{D-d} \exp(s(\mathbf{x}_{1:d}))_j = \exp(\sum_{j=1}^{D-d} s(\mathbf{x}_{1:d})_j)$$

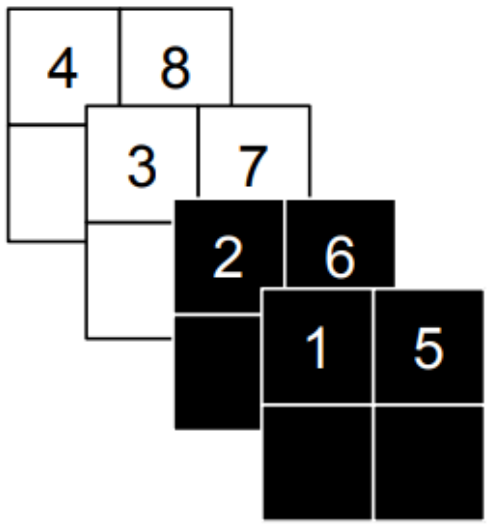
3. and, computing the determinant does not involve computing the Jacobian of s or t.

Thus, s and t can be arbitrarily complex - they can be modeled by neural networks.

Real NVP (Dinh et al. 2017)

To make sure that all the inputs have a chance to be altered, the model reverses the ordering of the data at each layer.





DEMO

Glow (Kingma and Dhariwal 2018):

The transformations are the same as in NICE and RealNVP, but changes the permutation operation by a 1×1 convolution.

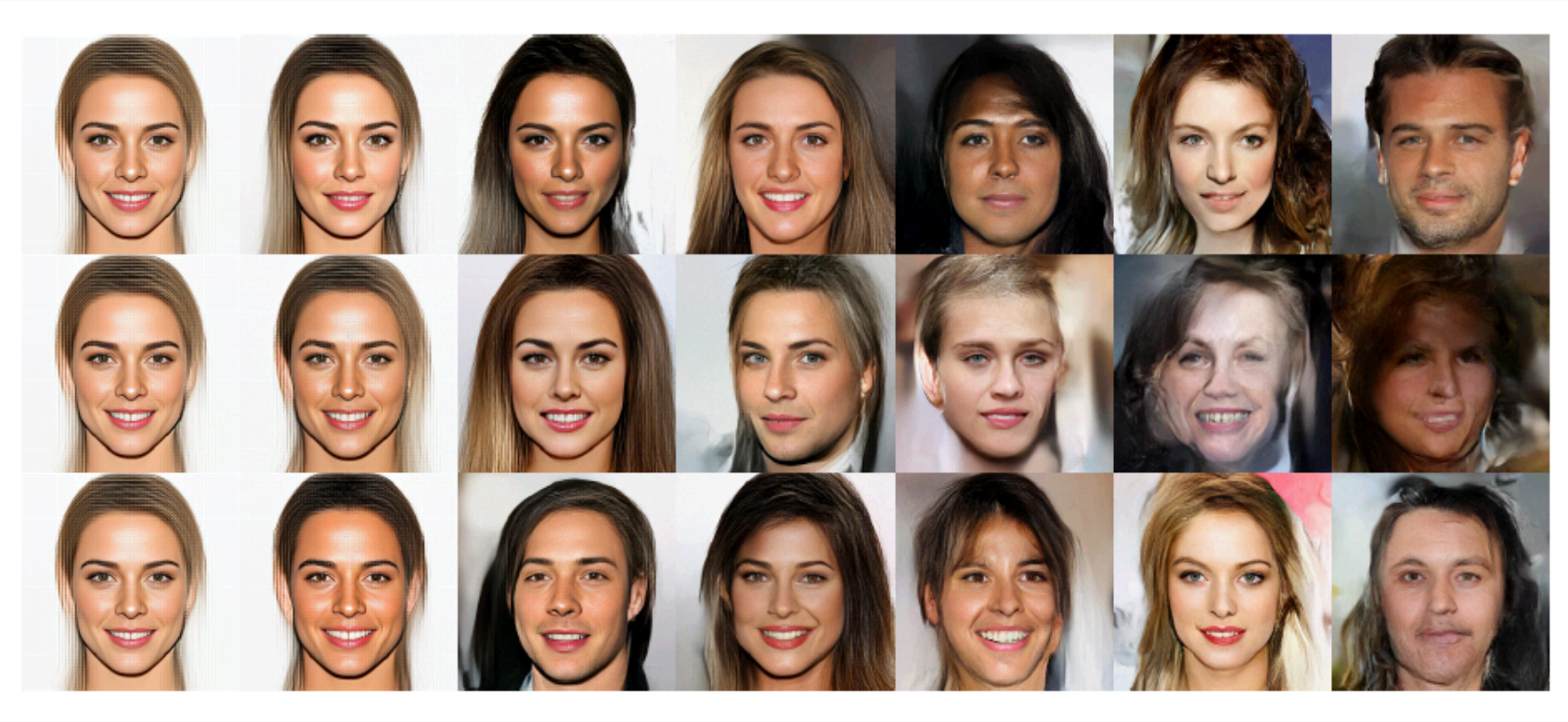
Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i, j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i, j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$
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)	$\mathbf{x}_a, \mathbf{x}_b = \texttt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \texttt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \texttt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \texttt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\texttt{sum}(\log(\lvert\mathbf{s}\rvert))$

Glow (Kingma and Dhariwal 2018):

Samples:

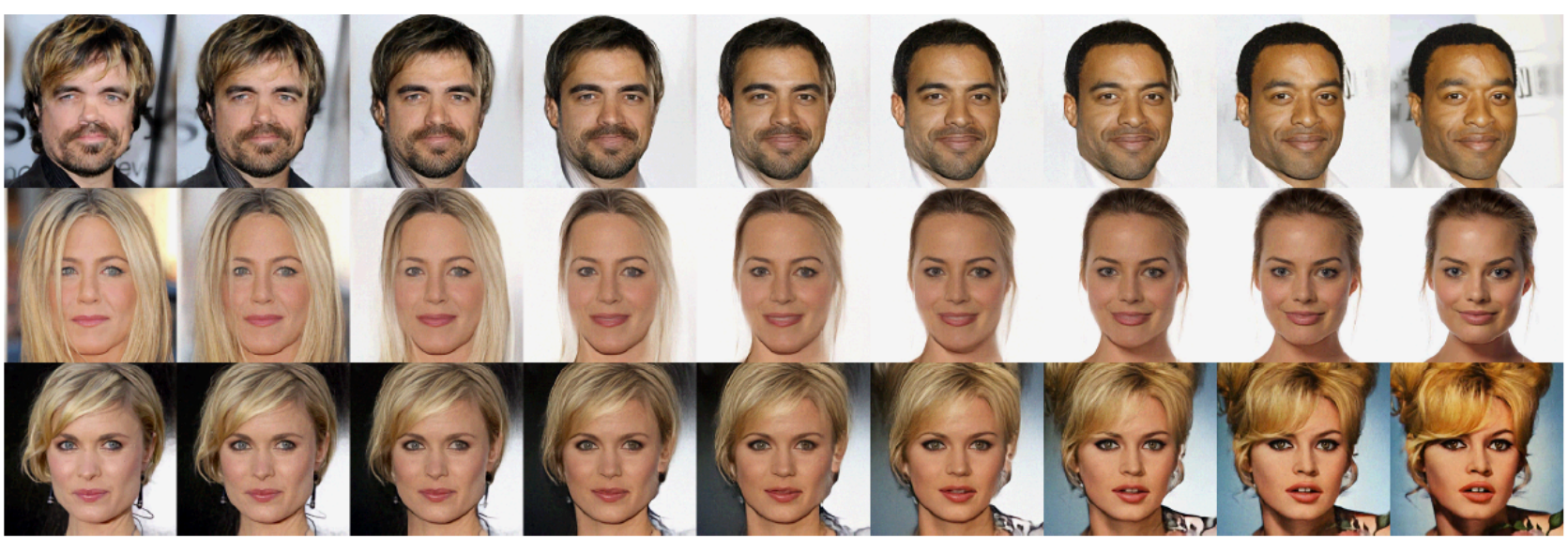
Glow (Kingma and Dhariwal 2018):

Samples as a function of temperature (T = 0,0.25,0.6,0.7,0.8,.9, 1):



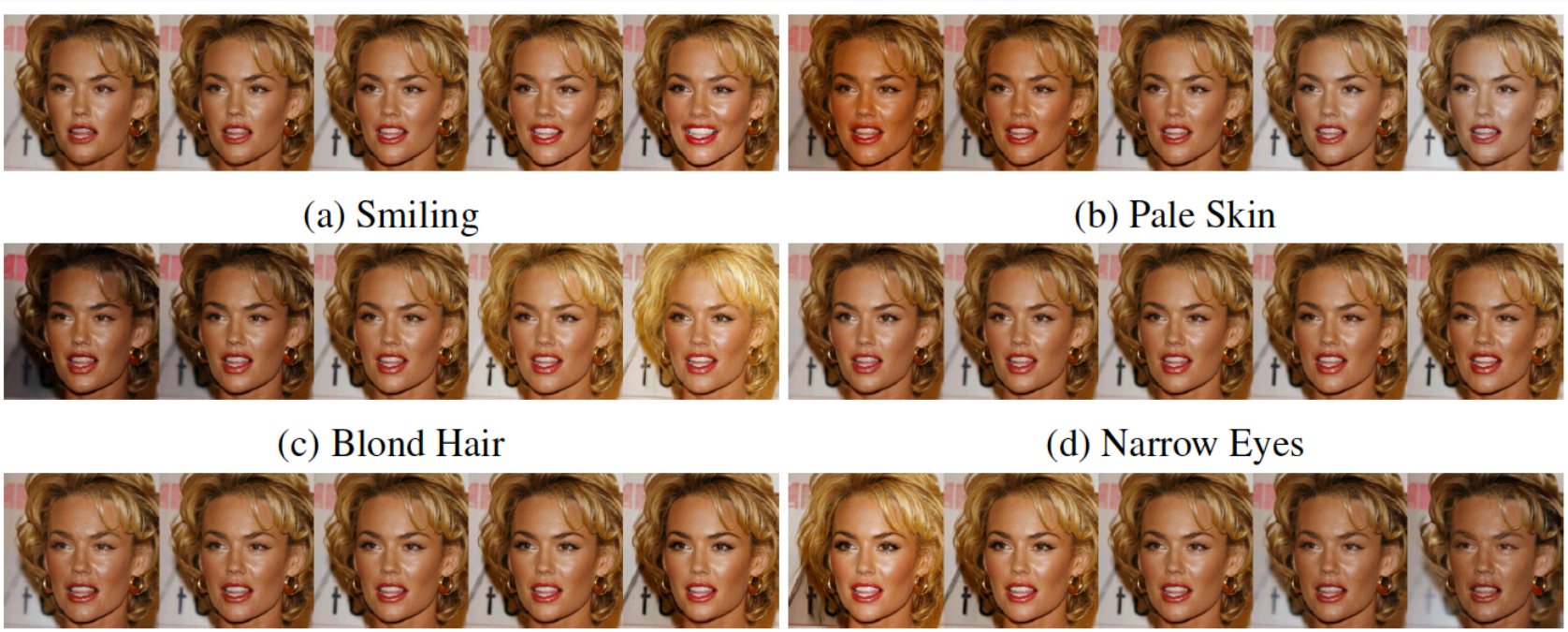
Glow (Kingma and Dhariwal 2018):

Interpolating in latent space:



Glow (Kingma and Dhariwal 2018):

Interpolating in latent space:



(a) Smiling (b) Pale Skin

(c) Blond Hair (d) Narrow Eyes

(e) Young (f) Male

An example: generating weak lensing maps with Glow