Glow: Generative Flow with Invertible 1×1 Convolutions

Kingma & Dhariwal, OpenAI (2018)

Figure 1: samples from the model

COMP 690 - Deep Generative Models
Presented by Yuvraj Jain
Sep 10, 2026

Let's travel back to 2018

- GANs owned the "realistic images" narrative
  - Progressive GAN months earlier, StyleGAN months later
- Likelihood-based methods: principled but seen as blurry
- Three families, each with one fatal flaw:
  - Autoregressive: best likelihoods, sequential sampling
  - VAEs: parallel, but optimizes a lower bound
  - Flows: exact likelihood and parallel sampling, but niche
- Glow is the paper that briefly made flows a contender

However… Glow aged like milk

Never closed the gap on sample quality, and diffusion models took over. But, the invertible 1x1 conv survived as a general “learned channel mixing” trick. Modern flow matching models report FID and do much better

What are the merits of flow-based models?

- Exact latent-variable inference and exact log-likelihood
- Efficient inference and synthesis, both parallelizable
- Useful latent space for downstream tasks *
- Significant potential for memory savings *

** claimed in the intro, not demonstrated in the paper*

Background: the objective and the setup

- Maximize likelihood = minimize negative log-likelihood (eq. 1)

$$\mathcal{L}(\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} -\log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$$

- Generative process: $z \sim N(0, I)$, $x = g(z)$
- g is invertible, so latent inference is exact: $z = f(x) = g^{-1}(x)$
- g is a chain of simple invertible steps: $f = f_1 \circ f_2 \circ ... \circ fK$

Background: change of variables

Given some random variable $z \sim \pi(z)$ and a invertible mapping $x = f(z)$ (i.e., $z = f^{-1}(x) = g(x)$).

The multivariate version takes the following form:

$$p(\mathbf{x}) = \pi(\mathbf{z}) \left| \det \frac{d\mathbf{z}}{d\mathbf{x}} \right| = \pi(g(\mathbf{x})) \left| \det \frac{dg}{d\mathbf{x}} \right|$$

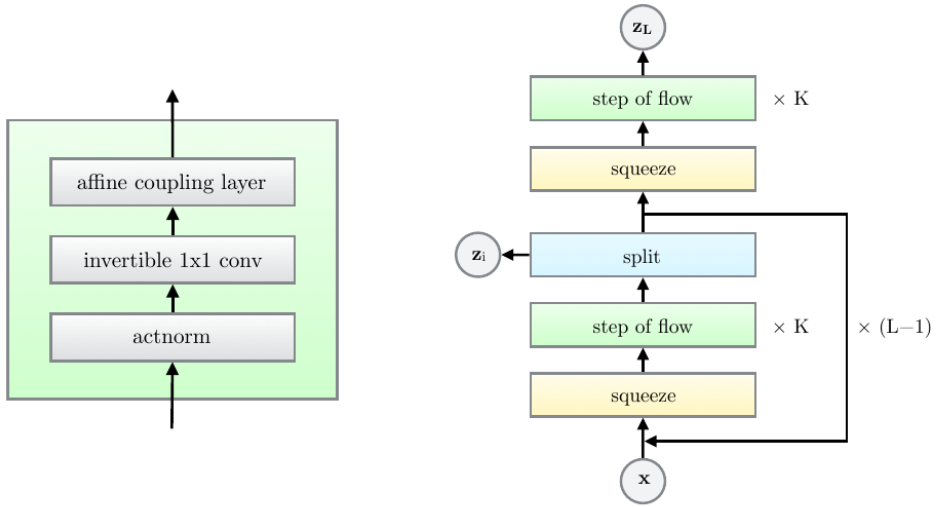
where $\det \frac{dg}{d\mathbf{x}}$ is the *Jacobian determinant* of g.

- Term 1: how likely is the latent this image maps to
- Term 2: correction for how much the map stretched or squeezed space

Triangular Jacobian → det = product of the diagonal

\Every layer in the paper is designed to keep that determinant cheap

Proposed generative flow



(a) One step of our flow.

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

Figure 2: one step of flow (left), multi-scale architecture (right)

No bottleneck: squeeze trades resolution for channels, split factors out half the latents

Step 1: Actnorm

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

Table 1, row 1

- One scale and one bias per channel, shared across all h · w spatial positions
- Elementwise per channel, so the reverse just divides s back out
- Replaces batchnorm, whose noise grows as per-GPU batch size shrinks
  - at 256×256 memory forces minibatch size 1 per GPU
  - batchnorm is not a bijection: output depends on the rest of the batch
- Data-dependent init on one batch: s = 1/std, b = −mean/std, per channel
- After init, ordinary trainable parameters, no further dependence on data
- Jacobian is diagonal, so log-det is h · w · sum(log|s|), derived next

Where h · w · sum(log|s|) comes from

Setup flatten the tensor: $N = h\,w\,c$ scalar coordinates, indexed by (i, j, k)

$$y_{i,j,k} \;=\; s_k\, x_{i,j,k} + b_k \qquad\qquad \mathbf{s}, \mathbf{b} \in \mathbb{R}^c$$

1 one Jacobian entry (b_k differentiates away: shifts do not change volume)

$$\frac{\partial y_{i,j,k}}{\partial x_{i',j',k'}} = \begin{cases} s_k & (i,j,k) = (i',j',k') \\ 0 & \text{otherwise} \end{cases}$$

2 each output uses exactly one input, so J is diagonal, $N \times N$

3 determinant of a diagonal matrix is the product of its entries

$$\det J \;=\; \prod_{i=1}^{h} \prod_{j=1}^{w} \prod_{k=1}^{c} s_k \;=\; \prod_{k=1}^{c} s_k^{hw}$$

s_k carries no i, j: the same scale is reused at all hw spatial positions

4 take logs

$$\log|\det J| \;=\; \sum_{k=1}^{c} hw \log|s_k| \;=\; h \cdot w \cdot \mathtt{sum}(\log|\mathbf{s}|)$$

The same h · w factor reappears in eq. 9: one matrix W shared across all spatial positions

Step 2: Invertible 1×1 convolution

Description	Function	Reverse Function	Log-determinant
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))

Table 1, row 2

- The paper's one genuinely new component
- RealNVP reversed the channel ordering; Glow learns a c × c matrix W instead
  - A 1×1 conv with equal in/out channels generalizes any permutation
- Initialized as a random rotation, so log-det starts at 0
- Not triangular: det(W) costs $O(c^3)$, but c is the channel count, not the dimension
- LU parameterization W = PL(U + diag(s)) drops it to O(c) (eq. 10)
- Cost: 0.2% more parameters, roughly 7% wallclock

Where h · w · log|det W| comes from

Setup $\mathbf{x}_{i,j} \in \mathbb{R}^c$ is the channel vector at one pixel; the same W acts at all hw pixels

$$\mathbf{y}_{i,j} = W\mathbf{x}_{i,j} \qquad \Longleftrightarrow \qquad y_{i,j,k} = \sum_{k'=1}^{c} W_{kk'}\, x_{i,j,k'}$$

1 one Jacobian entry (dense within a pixel, zero across pixels)

$$\frac{\partial y_{i,j,k}}{\partial x_{i',j',k'}} = \begin{cases} W_{kk'} & (i,j) = (i',j') \\ 0 & \text{otherwise} \end{cases}$$

2 group the coordinates by pixel: J is block diagonal, hw identical copies of W

3 determinant of a block diagonal matrix is the product of its blocks

$$\det J = \prod_{i=1}^{h}\prod_{j=1}^{w} \det W = (\det W)^{hw} \qquad \Longrightarrow \qquad \log|\det J| = h \cdot w \cdot \log|\det W|$$

Same hw factor as actnorm, same reason. Only the block changed: $\mathrm{diag}(\mathbf{s}) \to$ dense W, so eq. 8 no longer applies and $\det W$ costs $O(c^3)$.

- LU form W = PL(U + diag(s)) makes log|det W| = sum(log|s|), O(c) not O(c³)
  - eq. 10; P is sampled once and frozen, and they report no real wallclock gain

Step 3: Affine coupling layer

Description	Function	Reverse Function	Log-determinant
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \texttt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \texttt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \texttt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \texttt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\texttt{sum}(\log(\lvert\mathbf{s}\rvert))$

Table 1, row 3

- Split channels in half; transform x_a using a network applied to x_b only
- x_b passes through untouched, which forces a zero block in the Jacobian
- Triangular → log-det = sum(log|s|); the NN block sits off the diagonal
  - So NN() can be arbitrarily deep and costs nothing in the determinant
- Zero-init the last conv of each NN() → each layer starts as the identity
- Inherited from NICE / RealNVP, not new here

Where sum(log|s|) comes from

Setup split the channels; only the a half is transformed, and only the b half is looked at

$$(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{x}_b) \qquad \mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t} \qquad \mathbf{y}_b = \mathbf{x}_b$$

1 Jacobian in blocks, coordinates ordered a then b

$$J = \begin{bmatrix} \frac{\partial \mathbf{y}_a}{\partial \mathbf{x}_a} & \frac{\partial \mathbf{y}_a}{\partial \mathbf{x}_b} \\ \frac{\partial \mathbf{y}_b}{\partial \mathbf{x}_a} & \frac{\partial \mathbf{y}_b}{\partial \mathbf{x}_b} \end{bmatrix} = \begin{bmatrix} \operatorname{diag}(\mathbf{s}) & A \\ 0 & I \end{bmatrix}$$

lower left is 0 because $\mathbf{y}_b$ never sees $\mathbf{x}_a$; A holds every derivative of $\mathtt{NN}$

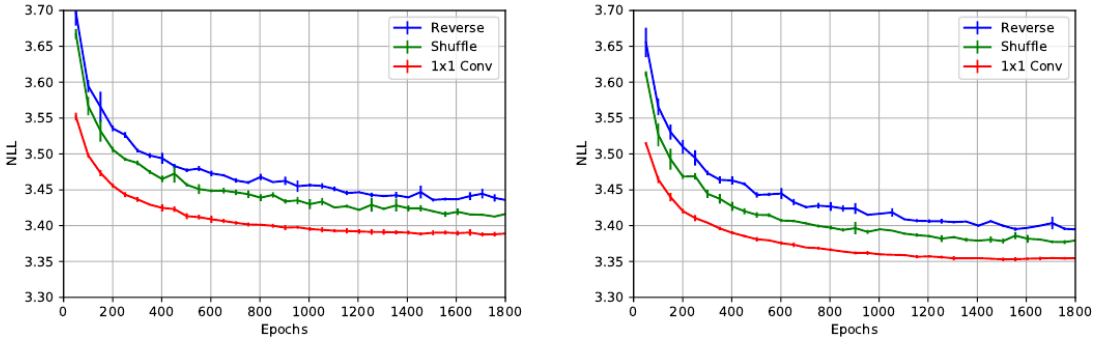
2 block triangular, so the determinant is the product of the diagonal blocks

$$\det J = \det\big(\operatorname{diag}(\mathbf{s})\big) \cdot \det(I) = \prod \mathbf{s} \qquad \Longrightarrow \qquad \log|\det J| = \mathtt{sum}(\log|\mathbf{s}|)$$

A never enters: $\mathtt{NN}$ can be arbitrarily deep and costs nothing in the determinant. And no hw factor here, because $\mathtt{NN}$ emits a different $\mathbf{s}$ at every position: nothing is shared, so nothing is repeated.

- Additive coupling (NICE) is the special case s = 1, log-det exactly 0
  - the affine version adds the scale, so coupling finally contributes to the likelihood

Does the 1×1 convolution actually help?



(a) Additive coupling. (b) Affine coupling.

Figure 3: reverse vs. shuffle vs. 1×1 conv, additive (left) and affine (right) coupling

- CIFAR-10, K = 32, L = 3, mean and std over three seeds, all else held constant
- 1×1 conv reaches lower NLL and converges faster in both settings
- The cleanest part of the paper: one variable changed, seeds reported

Comparison with RealNVP

Model	CIFAR-10	ImageNet 32x32	ImageNet 64x64	LSUN (bedroom)	LSUN (tower)	LSUN (church outdoor)
RealNVP	3.49	4.28	3.98	2.72	2.81	3.08
Glow	**3.35**	**4.09**	**3.81**	**2.38**	**2.46**	**2.67**

Table 2:Negative Log Likelihood in bits per dimension, lower is better

- Improves on RealNVP across every dataset tested
- Same preprocessing as RealNVP, so the comparison is fair
- But RealNVP is the only baseline in the table
  - No autoregressive models, no GANs, no perceptual metric anywhere
- Largest gains are on LSUN, roughly 0.3 to 0.4 bits/dim

The latent space is the real payoff

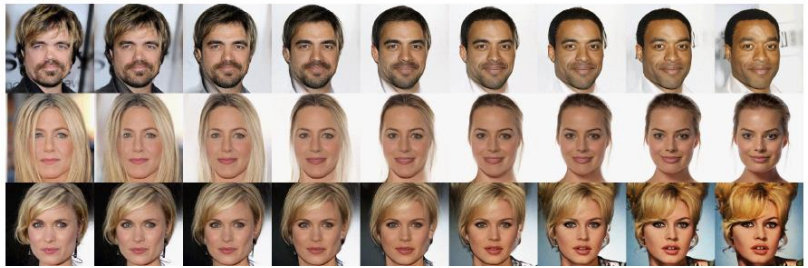


Figure 5: linear interpolation between real images

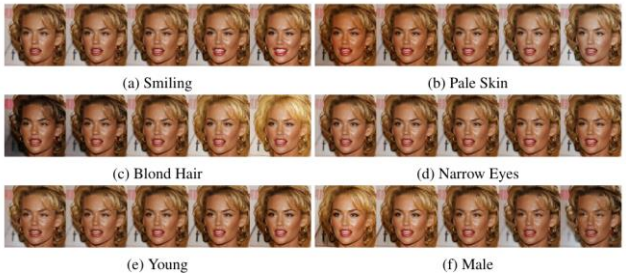


Figure 6: attribute manipulation

- Encode two real photos, interpolate the latents, decode: every midpoint is a plausible face
- Attribute direction = mean(z with label) − mean(z without), computed after training
- No labels used during training; one vector subtraction per attribute
- Only possible because inference is exact, which is the argument against GANs

Strengths and weaknesses

Strengths

- Clean controlled ablation for the new component
- First likelihood model with convincing high-res samples
- Exact inference makes editing and interpolation trivial
- Code released; the paper is short and readable

Weaknesses

- RealNVP is the only quantitative baseline
- No FID, no human eval; sample quality is asserted visually
- Enormous compute, never reported in GPU-hours
- 5-bit images and temperature 0.7 quietly do a lot of work
- Actnorm is argued for but never ablated
- "Downstream tasks" never demonstrated

Open questions and what survived

- Is likelihood the right target? Bits/dim and perceptual quality decouple
- Temperature < 1 means sampling from a distribution you did not train
  - If T = 0.7 is needed for good samples, is the learned density the problem?
- The cost of invertibility: every layer constrained by its determinant
- OOD detection, the most concrete promise, failed a year later (Nalisnick et al., 2019)
- What survived: the 1×1 conv as a general learned channel-mixing trick
- Flow matching kept the transport framing and dropped the invertibility constraint
  - SD3, Flux and others are flow models; they report FID, not bits/dim

Questions

- Why is 1x1 convolution used instead of any other dimension?

  What they're really doing is channel mixing, calling it a 1x1 convolution is just putting it in terms people in Computer Vision are already familiar with.

- Why does the 1×1 convolution help so much when it's still just a linear operation? Is it mainly because the model gets more flexibility in mixing the channels, or does it also make training easier?

  Training is still easy because invertible. Permutation matrix is just a special case of W here in Glow.

- How does Glow actually compare with GANs from the same time period using something like FID? The samples look really good, but I was curious about how they compare quantitatively?

  Glow does not report FID… that is a big weakness