African Institute for Mathematical Sciences — Cameroon
Structured Master's, Data Science Stream · Computer Vision

Learning to Create

Week 3 · Day 1: Generative Models, Autoencoders and Variational Autoencoders

Yusuf Brima, Ph.D.

Introduction to Computer Vision · AIMS Cameroon, 2026–2027

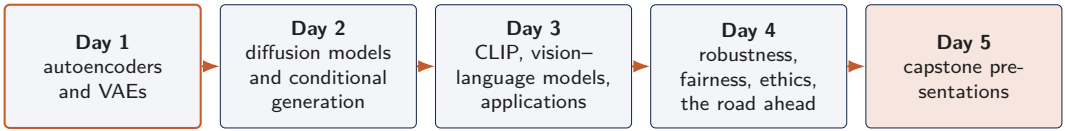
Today

1. **Generative modelling**: from recognising images to creating them
2. **Autoencoders**: compress, then reconstruct
3. **What the code space looks like**, and why sampling from it fails
4. **Variational autoencoders**: a code space you can sample from
5. **Uses**: denoising, anomaly detection, compression, and the engine room of Stable Diffusion
6. **A glimpse of GANs**

By the end of today you should be able to

- explain the difference between discriminative and generative models;
- build and train an autoencoder, and read its latent space;
- explain the VAE loss: reconstruction + KL, and the reparameterisation trick;
- use reconstruction error for anomaly detection;
- place AEs, VAEs, GANs and diffusion models side by side.

Week 3 in one picture



- Weeks 1–2: machines that *understand* images: classify, detect, segment, describe.
- Week 3: machines that *create* images, connect them to language, and the responsibilities that come with both.

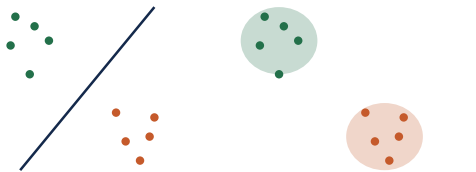
Practitioner's Note

Capstone work runs alongside all week. Lectures stay at two hours; consultations and tutorials happen outside them.

Part 1

Generative modelling

Two kinds of models



discriminative: draw the boundary
$p(\text{label} \mid \text{image})$

generative: describe where data live
$p(\text{image})$ or $p(\text{image} \mid \text{label})$

- Weeks 1–2 were **discriminative**: given an image, which label?
- A **generative** model learns what the data themselves look like, well enough to produce *new* examples.

Intuition

A discriminative model can tell a cassava leaf from a bean leaf. A generative model could *draw* a cassava leaf that never existed, which requires knowing far more about what cassava leaves look like.

Why would we want to generate images?

- **Restoration**: denoise, deblur, inpaint, super-resolve (Week 1's problems, solved by learning).
- **Anomaly detection**: learn what "normal" looks like; flag what the model cannot reproduce (industrial defects, unusual scans).
- **Data augmentation**: more training examples for rare classes.
- **Compression**: describe an image with a few numbers.
- **Simulation**: images for robotics and planning.
- **Representation learning**: a model that can generate has learned useful features (MAE, Week 2).
- **Creative tools**: design, illustration, education.

Practitioner's Note

Generated images used as *training data* for medical or safety-critical systems must be validated carefully: a model can learn the generator's mistakes.

Why generating images is hard

How many possible images?

A 28×28 grey image with 256 levels per pixel: 256^{784} possible images, a number with 1888 digits.

A 512×512 colour photo: $256^{786\,432}$.

Choose pixels at random and you get noise, every single time.

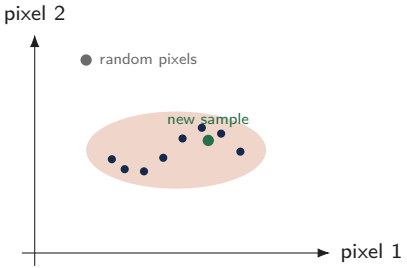
The manifold hypothesis

Real images occupy a tiny, smooth, low-dimensional region (a "manifold") inside the enormous space of all pixel arrays.

Intuition

A generative model must find this thin region and learn to move around inside it: changing the pose of a shoe or the shade of a leaf, without ever falling off into noise.

What it means to learn a distribution



- Every image is a point in a space with one axis per pixel (784 axes for Fashion-MNIST; drawn here with 2).
- Real images (navy) gather in a small region; the model learns its shape (orange).
- **Generating** = drawing a new point from that region (green), not copying a training image.

Intuition

Week 2 drew a boundary *between* regions. Today we learn the shape *of* a region well enough to sample new members from it.

A family of generative models

Family	Idea	Where in this course
Autoencoders	compress to a code, decode back	today
Variational autoencoders	a code space you can sample from	today; inside Stable Diffusion (Day 2)
GANs	a generator fools a critic	briefly today
Diffusion models	learn to remove noise, step by step	Day 2
Autoregressive models	generate one patch or token at a time	Day 3 (vision–language models)

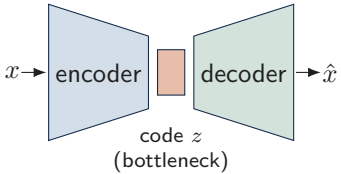
Intuition

All of them answer the same question differently: how do we turn simple random numbers into a realistic image? Today's answer: learn a *decoder* from a small code to an image, and learn what codes to feed it.

Part 2

Autoencoders

The autoencoder



- **Encoder**: image $\rightarrow$ a short code z (e.g. 16 numbers).
- **Decoder**: code $\rightarrow$ an image $\hat{x}$.
- Train both together so that $\hat{x} \approx x$.
- No labels needed: the image is its own target.

Reconstruction loss

$\mathcal{L}(x) = \|x - \hat{x}\|^2$ or pixel-wise cross-entropy

Intuition

The bottleneck forces a summary. To rebuild a shoe from 16 numbers, the network must learn what matters about shoes, and forget the rest.

Reconstruction loss, worked

Four pixels

Original $x = (0, 1, 1, 0)$, reconstruction $\hat{x} = (0.1, 0.8, 0.9, 0.3)$.

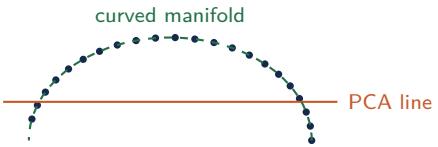
Squared error: $0.01 + 0.04 + 0.01 + 0.09 = 0.15$.

Binary cross-entropy, pixel by pixel $(-[x \log \hat{x} + (1 - x) \log(1 - \hat{x})])$:
$0.105 + 0.223 + 0.105 + 0.357 = 0.79$.

- Squared error suits any pixel values; cross-entropy suits pixels in $[0, 1]$ treated as probabilities of "ink".
- Both punish the last pixel (should be 0, predicted 0.3) the most.
- Our models use cross-entropy summed over the 784 pixels: that is the "reconstruction loss per image" on the next slides.

The simplest autoencoder is PCA

- A *linear* encoder and decoder with squared-error loss learn the same subspace as **principal component analysis**.
- PCA keeps the k directions in which the data vary most and projects onto them.
- Non-linear (deep) autoencoders can follow *curved* manifolds that PCA must flatten.



Compression ratio

A 28×28 image has 784 numbers; a 16-number code is $784/16 = 49$ times smaller. A 2-number code is 392 times smaller, small enough to plot.

Our convolutional autoencoder

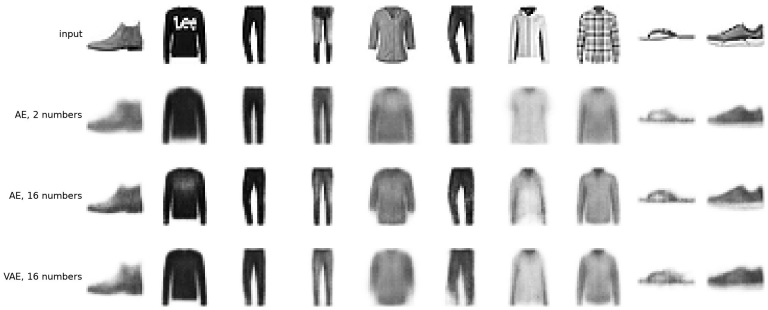


- Encoder: Week 2's CNN, ending in z instead of class scores.
- Decoder: a mirror image, with transposed convolutions (Week 2, Day 4) to grow back to 28×28.
- Trained on 12 000 Fashion-MNIST images, 10 epochs, about 124 seconds on a laptop CPU.

Intuition

The decoder is a generator in waiting: give it a code and it draws an image. The question for the rest of today is: *which codes* should we give it?

How good are the reconstructions?

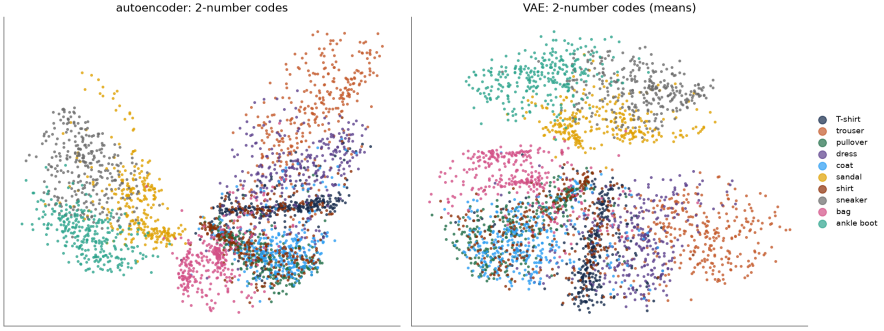


With **2 numbers**, the network keeps only the broad category and shape (reconstruction loss 256 per image). With **16 numbers**, some detail returns: the seams of the jeans, the sandal's straps, a faint trace of the sweater's logo (loss 219).

Intuition

The bottleneck size is a dial between *summary* and *detail*. A good code keeps what distinguishes items and drops what does not, like a skilled sketch artist.

The code space of an autoencoder



Left: 3000 test images, each placed at its 2-number code by the plain autoencoder. Similar items cluster together, although the network never saw a label.

Right: the VAE's codes (explained shortly) fill a more compact region, closer to the round prior, with fewer empty gaps between classes.

What the code numbers mean

- No single code number has a name: the network chooses its own coordinates.
- Yet nearby codes give similar images, and directions often carry meaning: one direction may separate footwear from clothing, another tops from bottoms.
- Classes cluster although the autoencoder never saw a label: the structure comes from the images alone.

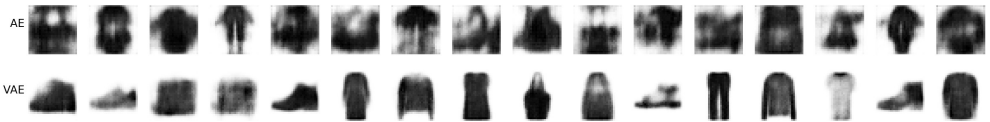
Intuition

This is unsupervised learning of a representation: the same idea that made DINO's features separate objects without labels (Week 2, Day 5). A code good enough to rebuild an image must capture what kind of thing it is.

Practitioner's Note

A quick way to explore a new, unlabelled image collection: train an autoencoder, plot the codes in 2-D (or project with PCA), and look at what lies in each cluster.

Why we cannot simply sample from an autoencoder



Pick 16 random codes from a standard normal distribution and decode them. The plain **autoencoder** (top) produces smudges: its codes live in an irregular region with gaps, and random points mostly land in the gaps. The **VAE** (bottom) produces recognisable garments.

Intuition

An autoencoder learns a map *from* images *to* codes, but nothing forces the codes into a shape we know. Its code space is a city with no street plan: you cannot find a house by picking a random address.

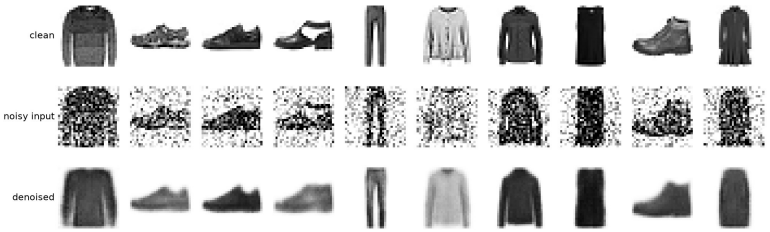
Variants: undercomplete, overcomplete, sparse

- **Undercomplete** (code smaller than input): the bottleneck forces compression. Our models.
- **Overcomplete** (code larger than input): could learn to copy pixels through unchanged, learning nothing. Needs a different constraint:
  - **sparse** codes (most entries zero);
  - **denoising**: corrupt the input, reconstruct the clean image;
  - **masking**: hide patches, reconstruct them (MAE, Week 2, Day 5).

Intuition

The goal is never the reconstruction itself; it is what the network must learn to achieve it. Any constraint that makes copying impossible forces understanding.

Denoising autoencoders

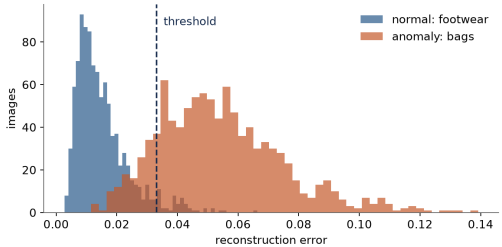


Trained with heavy Gaussian noise added to every input, and the clean image as the target. On new test images it removes the noise and keeps the garment's shape.

Intuition

Compare with Week 1, Day 3: a Gaussian blur also removes noise, but blurs the edges. A learned denoiser knows what clothes look like, so it keeps the edges that belong to clothes. Tomorrow's diffusion models are denoisers taken to the extreme.

Anomaly detection with reconstruction error



- Train an autoencoder *only on normal items*: here, footwear.
- Show it something new (bags). It reconstructs them badly: their reconstruction error is high.

Our result

Separating bags from footwear by error alone: area under the ROC curve 0.98. With the threshold set so 5% of normal items are flagged, 86.6% of bags are caught.

Practitioner's Note

Industrial inspection: train on the many good parts; flag what the model cannot reproduce. No defect labels needed.

Choosing the anomaly threshold, worked

A production line

10 000 good parts and 20 defective parts per day. The threshold flags 5% of good parts and catches 90% of defects.

Flagged good parts: $0.05 \times 10\,000 = 500$.
Caught defects: $0.9 \times 20 = 18$; missed: 2.

Of 518 flagged parts, only 18 are defective: precision 3.5%.

Intuition

When anomalies are rare, even a small false-alarm rate swamps the true alarms. An inspector who must check 500 parts a day to find 18 will soon stop checking.

Practitioner's Note

Set the threshold from the cost of each error and the inspection capacity, and report precision at the real defect rate, not just AUC.

Part 3

Variational autoencoders

The VAE idea: make the code space a known shape

- Choose a simple **prior** for codes: $z \sim \mathcal{N}(0, I)$, a round cloud around the origin.
- Train the encoder so that the codes of real images, taken together, fill that cloud, with no big gaps.
- Then *any* point drawn from $\mathcal{N}(0, I)$ decodes to a plausible image.

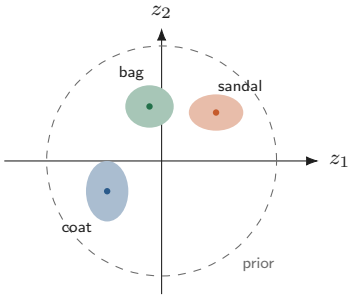
Intuition

Give the city a street plan: every address in the round region has a house, and neighbouring addresses have similar houses. Now "pick a random address" finds a house.

Generating with a trained VAE

$$z \sim \mathcal{N}(0, I), \qquad x = \text{decoder}(z)$$

A probabilistic encoder

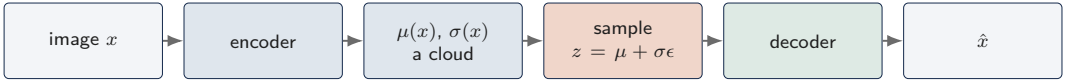


- The encoder outputs not one code but a small **cloud**: a mean $\mu(x)$ and a spread $\sigma(x)$ for each code dimension.
- During training we sample a code from that cloud and decode it.
- Neighbouring codes must therefore decode to similar images: the code space becomes smooth.

Intuition

Instead of pinning each image to one point, we smear it over a small area. The smears of different images overlap and fill the space, so there are no empty gaps between them.

The VAE, step by step



Training (every image, every step)

1. Encode x to μ and σ.
2. Sample one code from the cloud.
3. Decode; measure reconstruction error.
4. Add the KL penalty for μ and σ.
5. Backpropagate through everything.

Generating (after training)

1. Throw the encoder away.
2. Sample $z \sim \mathcal{N}(0, I)$.
3. Decode.

Intuition

The encoder is a training tool: it teaches the decoder which codes mean what. The generator is the decoder alone.

The VAE loss: two forces

Loss for one image

$$\mathcal{L}(x) = \underbrace{\mathbb{E}_{z\sim q(z|x)}\big[-\log p(x \mid z)\big]}_{\text{reconstruct well}} + \underbrace{\mathrm{KL}\big(q(z \mid x)\,\|\,\mathcal{N}(0, I)\big)}_{\text{stay close to the prior}}$$

- **Reconstruction**: wants each image's cloud to be tight and far from the others (easy to decode precisely).
- **KL term**: wants every cloud to look like the prior: centred near 0, spread about 1.

Intuition

A tug of war. Reconstruction pulls codes apart for precision; KL pulls them together for a smooth, gap-free space. The balance gives a code space that is both informative and easy to sample.

Our VAE with 16 numbers: reconstruction loss 234 per image (vs. 219 for the plain autoencoder) and KL 12. The KL term costs some sharpness.

The KL term for Gaussians, worked

KL to a standard normal, one dimension

$$\mathrm{KL} = \tfrac{1}{2}\left(\mu^2 + \sigma^2 - 1 - \log \sigma^2\right)$$

Summed over the code dimensions.

Three encoder outputs

$\mu = 0$, $\sigma = 1$: $\frac{1}{2}(0+1-1-0) = 0$. Exactly the prior.

$\mu = 2$, $\sigma = 1$: $\frac{1}{2}(4) = 2$. Penalised for drifting away.

$\mu = 0$, $\sigma = 0.1$: $\frac{1}{2}(0.01 - 1 + 4.61) = 1.81$. Penalised for being too precise.

Intuition

The KL term has two separate complaints: "you moved away from the centre" (μ^2) and "your cloud is too narrow" ($-\log \sigma^2$). Together they stop the encoder from hiding images in private, pinpoint codes.

Where the loss comes from: the evidence lower bound

We would like to maximise $\log p(x) = \log \int p(x \mid z)\, p(z)\, dz$, the probability the model gives to real images. The integral over all codes is intractable, so we introduce the encoder's distribution $q(z \mid x)$ and rewrite it exactly as

$$\log p(x) = \underbrace{\mathbb{E}_q\big[\log p(x \mid z)\big] - \mathrm{KL}\big(q(z \mid x) \,\|\, p(z)\big)}_{\text{ELBO}} + \mathrm{KL}\big(q(z \mid x) \,\|\, p(z \mid x)\big).$$

The last term is ≥ 0, so the ELBO is a **lower bound** on $\log p(x)$. Maximising the ELBO (minimising our loss) pushes $\log p(x)$ up.

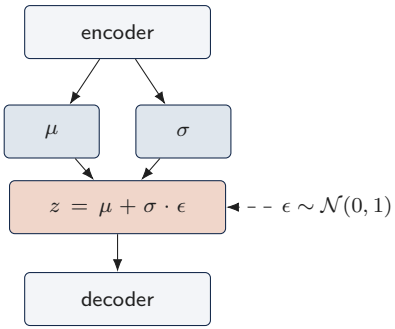
Intuition

We cannot measure how likely an image is directly, so we maximise a quantity that is guaranteed to be smaller: lift the floor and the ceiling must rise.

Practitioner's Note

You do not need to reproduce this derivation in a quiz; you need to know what each term of the loss does and why the KL term is there.

The reparameterisation trick

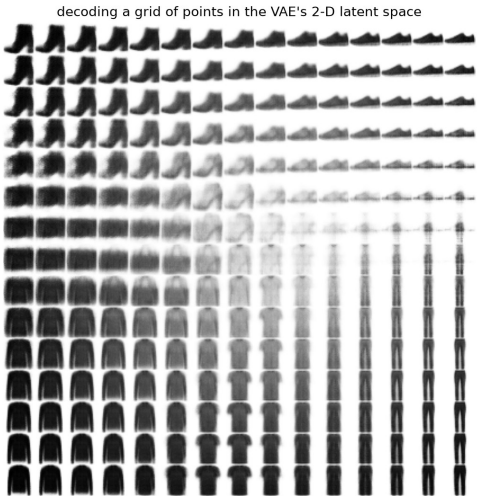


- Problem: "draw a random sample" has no gradient; backpropagation cannot pass through a dice roll.
- Trick: draw the randomness ϵ *outside* the network, then compute $z = \mu + \sigma\epsilon$.
- Now z is a smooth function of μ and σ, so gradients flow back to the encoder.

One sample

$\mu = 0.5$, $\sigma = 0.2$, $\epsilon = -1.5$: $z = 0.5 - 0.3 = 0.2$.
$\partial z/\partial \mu = 1, \quad \partial z/\partial \sigma = \epsilon = -1.5.$

Walking through a VAE's code space



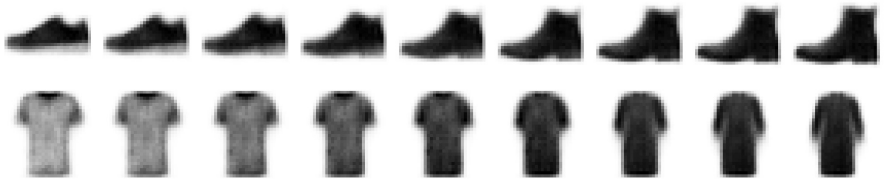
We decode a regular 15×15 grid of 2-number codes, from -2.2 to 2.2 in each direction.

- Almost every point gives a recognisable garment; only a pale band where footwear meets clothing shows where the decoder is unsure.
- Moving across the grid morphs smoothly: ankle boots into sneakers, pullovers into T-shirts and trousers.
- With only 2 numbers, everything is blurry; the model can keep only the gist.

Intuition

This picture is the manifold of Fashion-MNIST, flattened onto a sheet. The VAE learned the street plan.

Interpolation: morphing between two images



Encode two test images, then decode codes along the straight line between them: a sneaker gradually becomes an ankle boot (top); a T-shirt becomes a dress (bottom). Every step looks like a plausible item.

Intuition

Averaging two *images* pixel by pixel gives a ghostly double exposure. Averaging their *codes* gives a new, coherent object. Meaning lives in the code space, not in the pixels.

Arithmetic in code space

- If directions in code space carry meaning, we can do arithmetic with them.
- Average the codes of many images *with* a property and many *without*; their difference is a "property direction".
- Add it to any code to add the property.

Adding "boot height"

v = mean code(ankle boots) − mean code(sneakers).
For a new sneaker with code z: decode $z + 0.5v$, $z + v$: the sneaker grows a higher ankle, step by step.

Intuition

The same trick, at scale, edits faces (add glasses, change age) in GAN and diffusion latent spaces. It works because the model has organised its codes by meaning, without being told to.

Practitioner's Note

Directions found this way also carry the biases of the data: a "professional" direction learned from biased photos can change skin tone or gender. Inspect what a direction really changes.

Why are VAE images blurry?

- The decoder is trained with a pixel-wise loss. When it is unsure (is the strap here or 2 pixels to the right?), the loss is lowest for the *average* of the possibilities: a blur.
- The KL term deliberately limits how much information the code carries, so the decoder often must guess.
- Small decoders and tiny codes make it worse.

The averaging effect

A stripe is equally likely at column 10 or 12 (value 1, background 0). Predict 1 at column 10: half the time the error is 2 pixels of size 1. Predict 0.5 at both: always 2 pixels of error 0.5. Squared loss: 1 vs. 0.5. The blur wins.

Intuition

Sharper generators need a loss that rewards *realism*, not just closeness: a critic (GANs) or iterative refinement (diffusion, tomorrow).

β-VAE: turning the dial

β-VAE loss

$$\mathcal{L} = \text{reconstruction} + \beta \cdot \mathrm{KL}$$

- $\beta < 1$: sharper reconstructions, a less regular code space (closer to a plain AE).
- $\beta > 1$: smoother, more organised codes; blurrier images.
- Large β sometimes yields **disentangled** codes: one number controls size, another rotation, another thickness.

Intuition

β sets the exchange rate between detail and order. Choose it for the job: a compressor wants detail; a generator for exploration wants order.

Practitioner's Note

If your VAE ignores its code entirely ("posterior collapse"; KL near 0, samples all alike), lower β at the start of training and increase it gradually (KL warm-up).

Conditional VAEs: asking for a class

- Give both encoder and decoder the label y as an extra input.
- To generate: choose y ("sandal"), sample $z \sim \mathcal{N}(0, I)$, decode (z, y).
- The code z then captures what the label does not: style, thickness, pose.

Intuition

The label says *what*; the code says *how*. Keep z fixed and change y, and you get "the same style" of sandal, sneaker and boot.

Practitioner's Note

Conditioning is the basis of all controllable generation: on a class today, on text tomorrow ("a photo of a bean leaf with rust").

Generated images as training data

- A conditional generator can produce extra examples of a rare class: an uncommon disease, a rare defect.
- Helps most when real data are scarce and the generator is good; can hurt when samples are unrealistic or lack variety.
- Always evaluate on *real* test images only.

A fair experiment

Train classifier A on real data only; classifier B on real + generated data. Test both on the same real held-out set. Repeat over seeds. Keep the generated data only if B is reliably better, especially on the rare class.

Practitioner's Note

Generated data cannot add information the generator never had: a generator trained on 30 images of a rare lesion cannot invent the variety of the 31st.

The VAE inside Stable Diffusion

- Diffusion on full-resolution pixels is very expensive (Day 2).
- Latent diffusion first trains a VAE-like autoencoder that compresses an image to a small **latent** grid, and runs diffusion in that latent space.
- Its decoder turns the final latent back into pixels.

How much smaller?

Stable Diffusion: $512 \times 512 \times 3 = 786\,432$ numbers $\rightarrow$ a $64 \times 64 \times 4 = 16\,384$-number latent: 48 times fewer.

Intuition

The autoencoder handles "what pixels look like" (textures, edges); diffusion handles "what scenes look like" in a compact space. Today's model is the bottom layer of tomorrow's.

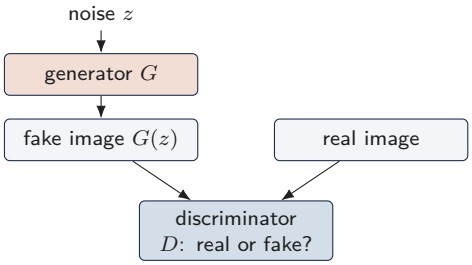
Practitioner's Note

VQ-VAE: codes chosen from a learned dictionary of vectors, i.e. discrete "visual words" (Week 1, Day 5, again). Used in many image and video generators.

Part 4

A glimpse of GANs

Generative adversarial networks (2014)



- Two networks play a game. The **discriminator** learns to tell real from generated images. The **generator** learns to fool it.
- The generator never sees a real image directly; it learns only from the discriminator's judgement.

The minimax game

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

Intuition

A forger and an inspector training each other. The loss is "does it look real?", not "is it close pixel by pixel?", so GAN images are sharp, unlike VAE images.

What GANs achieved, and why diffusion took over

Achievements

- Photorealistic faces at 1024×1024 (StyleGAN, 2019).
- Image-to-image translation: maps to satellite images, sketches to photos, day to night.
- Super-resolution with realistic texture.

Problems

- **Unstable training**: two networks chasing each other can oscillate.
- **Mode collapse**: the generator finds a few images that fool D and produces only those.
- Hard to condition on rich text.

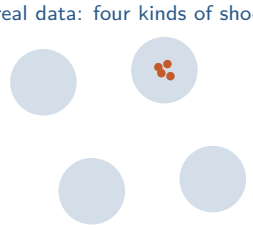
Intuition

Diffusion models (tomorrow) train with a simple, stable regression loss and cover the variety of the data well. Since about 2021 they have replaced GANs for most generation tasks; GANs survive where speed matters (one forward pass).

Practitioner's Note

"Deepfakes" began with GAN-based face swapping. Day 4 discusses detection, consent and misuse.

Mode collapse, intuitively



- The generator finds one kind of image the discriminator accepts and keeps producing it.
- Each sample looks perfect; the *collection* is wrong.
- Detect it by looking at many samples together and by recall-type metrics (FID, precision/recall).

Intuition

A student who learns one essay that always gets a pass and submits it for every question. The marker (discriminator) eventually catches on, the student switches to another essay, and training oscillates.

Four generative families, side by side

	Autoencoder	VAE	GAN	Diffusion
Trains with	reconstruction	reconstruction + KL	adversarial game	denoising (regression)
Can sample?	not reliably	yes	yes	yes
Image quality	blurry	blurry	sharp	sharp
Diversity	n/a	good	mode collapse risk	good
Training	easy	easy	unstable	stable, slow
Generation speed	fast	fast	fast	slow (many steps)
Typical use	compression, anomalies	latent spaces, priors	real-time, translation	text-to-image, editing

Intuition

Modern systems combine them: Stable Diffusion = an autoencoder for pixels + diffusion in its latent space + a text encoder from CLIP.

How do we know a generator is good?

- **Look at samples**: necessary, never sufficient.
- **Fréchet Inception Distance (FID)**: pass real and generated images through a pre-trained network (Inception), fit a Gaussian to each set of features, and measure the distance between the two Gaussians. Lower is better.
- **Precision and recall for generators**: are samples realistic (precision) and do they cover the variety of real data (recall)?

Intuition

FID compares *populations*, not single images: a generator that produces one perfect shoe over and over has realistic samples but poor variety, and FID notices.

Practitioner's Note

FID depends on the number of samples and the feature network; compare FIDs only within the same setup. For your applications, the best test is usually *task-based*: does training on generated data help a downstream model?

Do generators copy their training data?

- A generator that reproduces training images exactly would score well on realism and be useless (and possibly infringe copyright or privacy).
- Large text-to-image models have been shown to reproduce some training images almost exactly, especially ones duplicated many times in their data (Carlini et al., 2023).
- Check: for each generated sample, find its nearest training image; look at the closest pairs.

Practitioner's Note

For medical or personal data, memorisation is a privacy risk: a generator trained on patient scans could leak a real patient's scan. Deduplicate training data and test for memorisation before sharing a generator.

Intuition

The goal is to learn the *distribution*, not the *dataset*: new members of the family, not photocopies of old ones.

Part 5

Autoencoders in practice

Where autoencoders are used today

Use	How	Example
Anomaly detection	train on normal; flag high reconstruction error	defects on a production line; unusual scans
Denoising	corrupted input, clean target	low-dose X-ray, low-light photos
Compression	short code, learned decoder	learned image codecs; latent diffusion
Pre-training	masked reconstruction	MAE (Week 2, Day 5)
Visualisation	2-D codes	exploring a new image collection

Practitioner's Note

Anomaly detection with autoencoders has a known weakness: a powerful autoencoder can reconstruct anomalies too. Check on real anomalies, and consider comparing features from a pre-trained network instead of raw pixels.

Choosing a bottleneck size

- **Too small**: reconstructions lose detail that matters (our 2-number codes lose collars and straps).
- **Too large**: the autoencoder may copy too much, including noise and anomalies.
- Rule of thumb: increase the size until reconstructions keep the details your *task* needs; check on a validation set.

Our numbers

Reconstruction loss per image: 256 with 2 numbers, 219 with 16. Sixteen numbers for a 784-pixel image (a 49-fold compression) already keep most visible detail of Fashion-MNIST.

Intuition

The right size depends on how varied the images are: centred garments on a plain background need few numbers; street scenes need many.

A VAE in PyTorch

```
class VAE(nn.Module):
    def __init__(self, z=16):
        super().__init__()
        self.enc = ConvEncoder(out=2 * z)   # mu
     and log-variance
        self.dec = ConvDecoder(z)
        self.z = z
    def forward(self, x):
        h = self.enc(x)
        mu, logvar = h[:, :self.z], h[:, self.z:]
        eps = torch.randn_like(mu)
        z = mu + eps * (0.5 * logvar).exp()   #
     reparameterisation
        return torch.sigmoid(self.dec(z)), mu,
     logvar
xr, mu, logvar = vae(x)
rec = F.binary_cross_entropy(xr, x, reduction="sum"
     ) / len(x)
kl = (-0.5 * (1 + logvar - mu**2 - logvar.exp()).
     sum(1)).mean()
loss = rec + beta * kl
```

- Predicting $\log \sigma^2$ instead of σ keeps the spread positive without constraints.
- The KL line is the formula from three slides ago, summed over dimensions.
- Everything else is Week 2's training loop.

Common pitfalls when training autoencoders and VAEs

Symptom	Likely cause and fix
Outputs are grey blobs	decoder output not passed through a sigmoid, or pixels not scaled to $[0, 1]$
Every sample looks the same	posterior collapse: KL too strong early; warm up β from 0
Sharp reconstructions, noisy samples	KL too weak (β too small): the code space has gaps
Loss explodes	log-variance unbounded; clamp it or lower the learning rate
Anomalies reconstructed well	bottleneck too large; model too powerful; train fewer epochs or compare features instead

Practitioner's Note

Always look at three things during training: reconstructions of validation images, samples from the prior, and the two loss terms separately.

Reading the two loss terms during training

- **Reconstruction** should fall steadily: the decoder learns to draw.
- **KL** usually rises at first (codes spread out to carry information), then settles.
- KL near zero for every image means the code carries no information: posterior collapse.

Our 16-number VAE after 10 epochs

Reconstruction 234 and KL 12 per image. The KL is the "price" paid for a smooth code space: about 12 nats of information about each image flow through the code.

Intuition

KL measured in nats is literally an information budget: how much the code tells the decoder about this particular image beyond what the prior already says. A small budget forces the code to keep only the essentials.

Autoencoder or VAE? Choosing for a task

Task	Choose	Why
Denoising, restoration	(denoising) AE	reconstruction quality matters most
Anomaly detection	AE or VAE	both work; VAE gives a likelihood-like score
Generating new samples	VAE (or diffusion)	needs a code space you can sample
Smooth interpolation, editing	VAE	no gaps between codes
Compression for a generator	VAE-style autoencoder	as in latent diffusion

Intuition

Ask whether you will ever decode a code that did not come from a real image. If yes, you need a VAE (or another true generative model); if no, a plain autoencoder is simpler and sharper.

Part 6

Wrapping up

Key takeaways

1. Generative models learn what data look like; real images lie on a thin manifold in pixel space.
2. **Autoencoders** compress to a code and reconstruct; useful for denoising, anomaly detection and compression.
3. Their code space has gaps, so random codes decode to noise.
4. **VAEs** encode to a cloud, sample with the reparameterisation trick, and add a KL term that keeps codes close to $\mathcal{N}(0, I)$: now you can sample and interpolate.
5. VAE images are blurry; GANs are sharp but unstable; diffusion (tomorrow) is sharp and stable.
6. The autoencoder lives on inside Stable Diffusion.

Before the next lecture

Practise (tutorial with the tutors)

- Train today's autoencoder and VAE; plot the 2-D code spaces.
- Change β; watch reconstructions and samples.
- Build an anomaly detector for a class of your choice.

Capstone: teams confirmed today; data downloaded and inspected; book a consultation slot.

Next lecture: diffusion models

- Destroying images with noise, step by step
- Learning to undo it
- A diffusion model trained on a laptop
- Guidance: asking for what you want
- Text-to-image: Stable Diffusion

Credits

Fashion-MNIST: Zalando Research (MIT licence). All models were trained for this lecture on a laptop CPU with `Week3/tools/make_week3_figures.py`.