Image generation

Deep learning - BMDC 2025-2026

Gheorghe Cosmin Silaghi

Universitatea Babeș-Bolyai

October 21, 2025

Image generation

What is

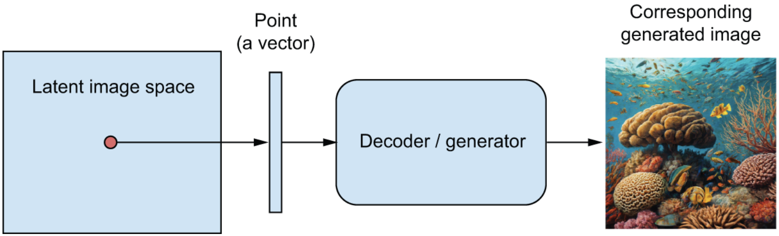
- learning latent visual spaces and sampling from them to create entirely new pictures, interpolated from real ones
- techniques for image generation are not specific to images, but in practice, the most interesting results were obtained on images

Two main techniques

- variational autoencoders (VAEs)
- diffusion models

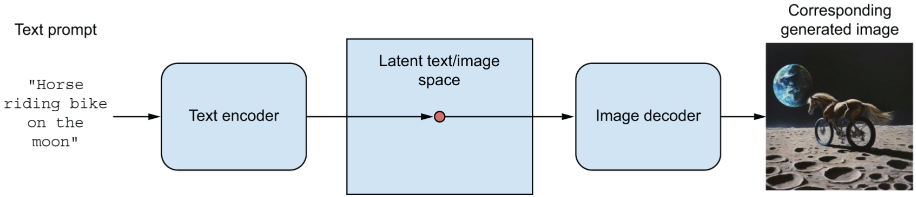
Sampling from latent spaces of images

- key idea: to develop a low-dimensional latent space of representations where any point can be mapped to a valid image: an image that looks like a real thing
- generator (or decoder): the module capable of realizing this mapping. Takes as input a latent point and outputs an image (grid of pixels)
- once the latent space has been learned, you can sample points from it, and by mapping tghem back to the image space, generate images that have never been before
- those generated images are in-between training images



Language-guided image generation

- *image-to-text models*: map a space of prompts in natural language to the latent space - generates pictures that correspond to a text description
- interpolating between many training images in the latent space enables that such models to generate infinite combinations of visual concepts


Text prompt
"Horse riding bike on the moon"
Text encoder
Latent text/image space
Image decoder
Corresponding generated image

Challenges

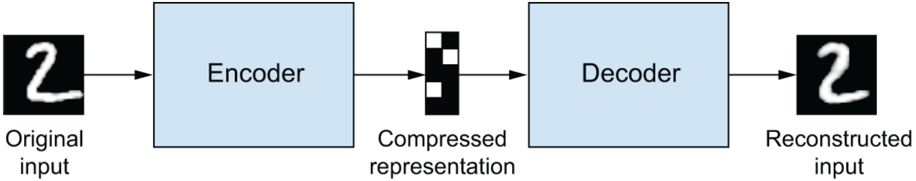
- coherence of the generated images: the latent space doesn't encode a consistent model of the physical world, so we might occasionally see hands with extra fingers, incoherent lighting, garbled objects etc.

strategies for learning the latent space

- diffusion models
- variational autoencoders (VAEs)
- generative adversarial networks (GANs) - gradually fallen out of fashion

Variational autoencoders

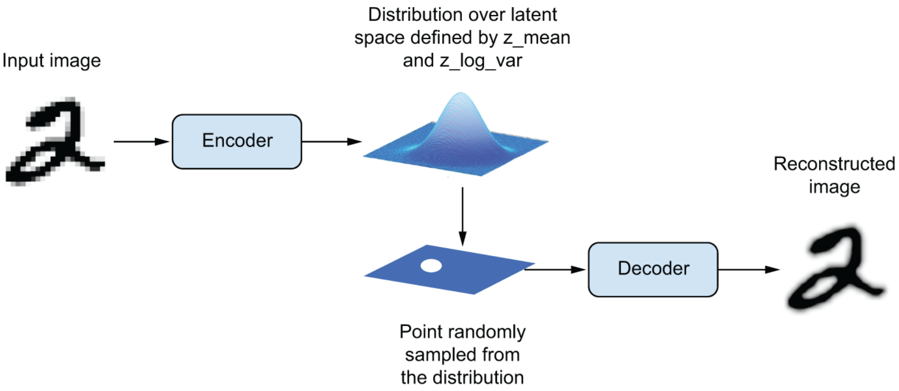
- autoencoder: a type of network that aims to encode an input to a low dimensional latent space and then decode it back
- VAEs remain an important tool in deep learning toolbox, especially when interpretability, control over the latent space and data reconstruction capabilities are crucial
- a classical autoencoder takes an image, maps it to a latent vector space via an encoder module, then decodes it back to an output with the same dimension as the original image
- it is trained having the target the same images as the input images - the autoencoder learns to reconstruct the original inputs
- by imposing various constraints on the code, you get the autoencoder to learn more or less interesting latent representations of the data



VAEs

- a VAE, instead of compressing its input image into a fixed code in the latent space, turns the image into the parameters of a statistical distribution: a mean and a variance.
- they assume that the input image was generated by a statistical process and the randomness of this process should be taken into account during encoding and decoding.
- the VAE uses the mean and the variance parameters to randomly sample one element of the distribution and ecodes that element back to the original input
- stochasticity of the process improves robustness and forces the latent space to encode meaningful representation everywhere

VAE



Input image
Encoder
Distribution over latent space defined by z_mean and z_log_var
Point randomly sampled from the distribution
Decoder
Reconstructed image

How VAE works

1. an encoder module turns the input sample *input_img* into 2 parameters in a latent space: *z_mean* and *z_log_variance*
2. sample a point z from the normal distribution assumed to generate the input image: $z = z\_mean + exp(z\_log\_variance) * epsilon$; epsilon is a random tensor with small values
3. decoder module maps this point back in the latent space back to the original input image

Loss functions

- reconstruction loss: forces the decoded samples to match the initial inputs
- regularization loss: helps to learn a well-rounded latent distribution and reduces overfitting to the training data

Implementing VAE with Keras

Components:

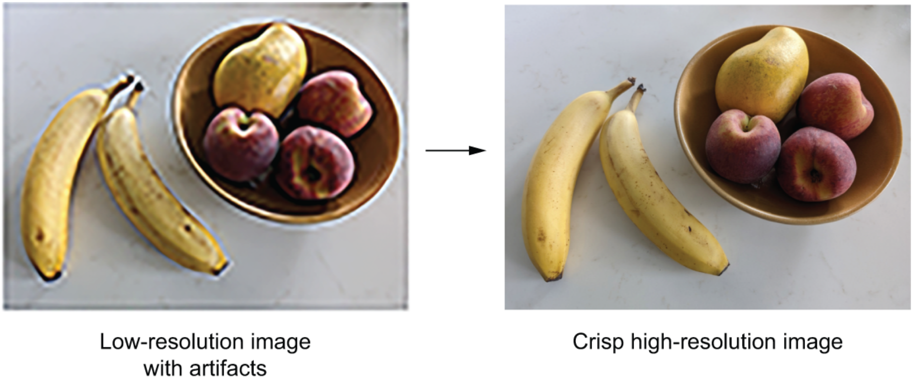
- an encoder network that turns a real image into a mean and a variance in the latent space
- a sampling layer that takes the mean and the variance and uses them to sample a random point in the latent space
- a decoder network that turns points in the latent space back into images

Other implementation details:

- we use strides to downsample the feature maps (and not maxpooling), because we need to preserve location information to be able to reconstruct the image

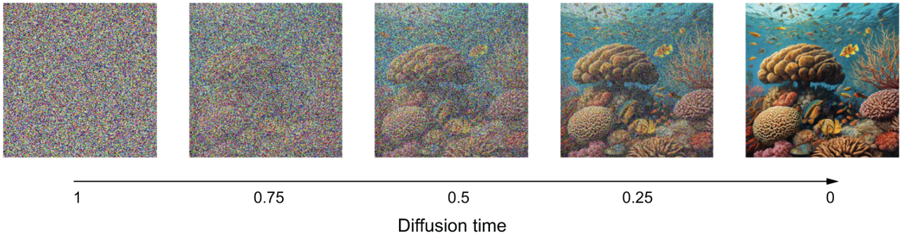
Diffusion models

- **denoising**: feeding into a model an input that features a small amount of noise. Autoencoders are good at denoising
- **image super-resolution models**: are capable of taking as input a low resolution image (potentially noisy) and output a high-quality high-resolution version of the image



How they function?

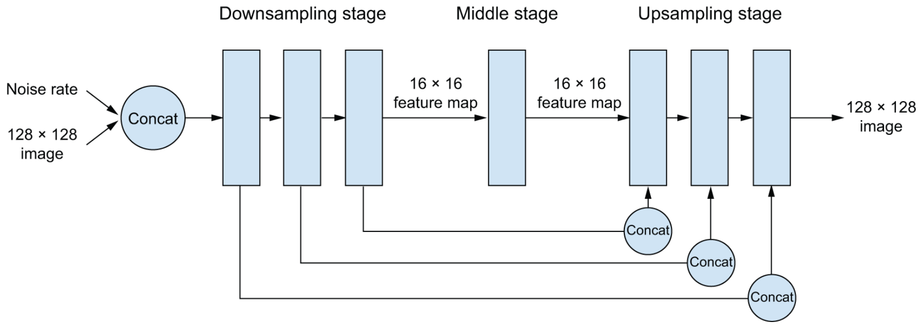
- they make educated guesses about what the image should look like - they are hallucinating a clean-up higher-resolution version
- this can lead to funny mishaps
- with autoencoders, since you can remove a small amount of noise from an image, surely it would be possible to repeat the process multiple times in a loop and remove a *large* amount of noise.
- could we denoise a *pure noise* image? response: **yes**
- in this way we hallucinate brand new images out of nothing (*reverse diffusion*)
- *diffusion*: the process to gradually add noise to an image until it disperse into nothing.



what is a diffusion model?

- a denoising autoencoder loop, capable of turning pure noise into sharp realistic imagery
- the same denoising model gets reused across each iteration, erasing a little bit of noise each time.
- to make the job of the model easier, we tell how much noise it is supposed to extract from the given image (noise_rates parameter)
- the model will output a predicted noise mask, which will be subtracted from the input to denoise it.

U-Net

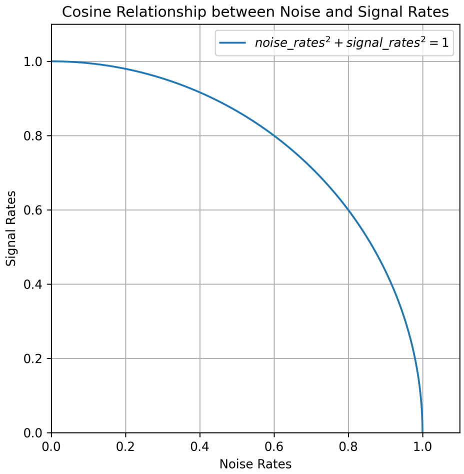

Downsampling stage
Middle stage
Upsampling stage
Noise rate
128 × 128 image
Concat
16 × 16 feature map
16 × 16 feature map
128 × 128 image
Concat
Concat
Concat

U-Net architecture

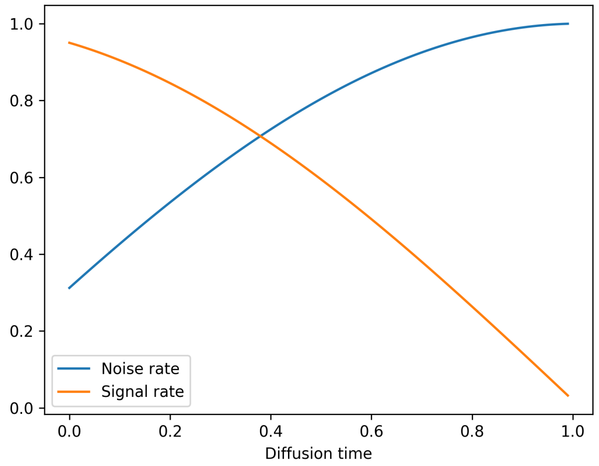
1. *a downsampling stage*: several blocks of convolution layers, inputs get downsampled from their original size to a much smaller size
2. *a middle stage*: the feature map has a constant size
3. *an upsampling stage*: the feature map gets upsampled back to initial dimension. Each upsampling block is a reverse of a downsampling block
4. the model features a concatenative residual connection going from each downsampling block to the corresponding upsampling block - we avoid loss of image detail information across successive downsampling and upsampling operations

Diffusion time and diffusion schedule

- diffusion process: a series of steps in which we apply a denoising autoencoder to erase a small amount of noise from an image
- **diffusion time**: the index of the current step in the loop. 1 indicated the start of the process. 0 indicates the end
- **diffusion schedule**: the relationship between the current diffusion time and the amount of noise and signal present in the image
- in our case we use a cosine schedule


Cosine Relationship between Noise and Signal Rates
$noise\_rates^2 + signal\_rates^2 = 1$
Signal Rates
Noise Rates
0.0
0.2
0.4
0.6
0.8
1.0

Cosine diffusion schedule



The training process

- **DiffusionModel** class implements the training procedure. we need:
    - a loss function: MSE
    - an image normalization layer: noise added to the images will have unit variance and zero mean. therefore, images should be normalized as well, sucha that value range of noise to mathc the value range of the images
- denoise method: simply calls the denoising model to retrieve the predicted noise mask and uses it reconstruct a denoised image

The training logic

- we need to implement a custom **compute_loss()** method to keep the model backend agnostic.
- compute_loss() gets as input the output of call(). We put the denoising forward pass in call(). call() takes a batch of clean images and do the following:
  1. normalize the images
  2. samples random diffusion times - the denoising model needs to be trained on the full spectrum of diffusion times)
  3. computes the corresponding noise rates and signal rates (using the diffusion schedule)
  4. adds random noise to the clean input images
  5. denoise the images
- it returns:
  - the predicted denoised images
  - the predicted noise masks
  - the actual noise masks it applied
- these two quantities are used in compute_loss() to compute the loss of the model on the noise mask prediction task

Text-to-image models

- create a model that maps text input to image output
- we need a pre-trained text encoder (e.g. a transformer encoder like RoBERTa) - maps text ot vectors in a continuous embedding space
- then we train a diffusion model on (prompt, image) pairs. each pair is a short textual description of the image

- we pass the embedded text prompt to the denoising model.
- rather than the denoising model taking as input a `noisy_images`, our model will take two inputs: `noisy_images` and `text_embeddings`.
- the model gets to use a textual representation of the final image to help guide the denoising process
- because we trained a model that can map pure noise images conditioned on a vector representation of some text, we can now pass pure noise and a never-seen-before prompt and denoise it to an image for our prompt.
- **negative prompt**: steer the diffusion process away from certain text inputs
- to add a negative prompt you train the model on triples `(image, positive_prompt, negative_prompt)`.

Exploring the latent space of a text-to-image model

- the text encoder used by the model learns a smooth, low-dimensional manifold to represent our input prompts - this space is continuous
- we learned a space where we can walk from the text representation of one prompt to another, and each intermediate points will have a semantic meaning.
- we can couple that with our diffusion process, to morph between two images, by describing each end state with a prompt

Steps of the generation process

1. take our prompts, tokenize them and embed them with the text encoder
2. take our text embeddings and pure noise, and progressively denoise the noise into an image
3. map the model outputs (from [-1,1]) to [0,255], such that to render the image