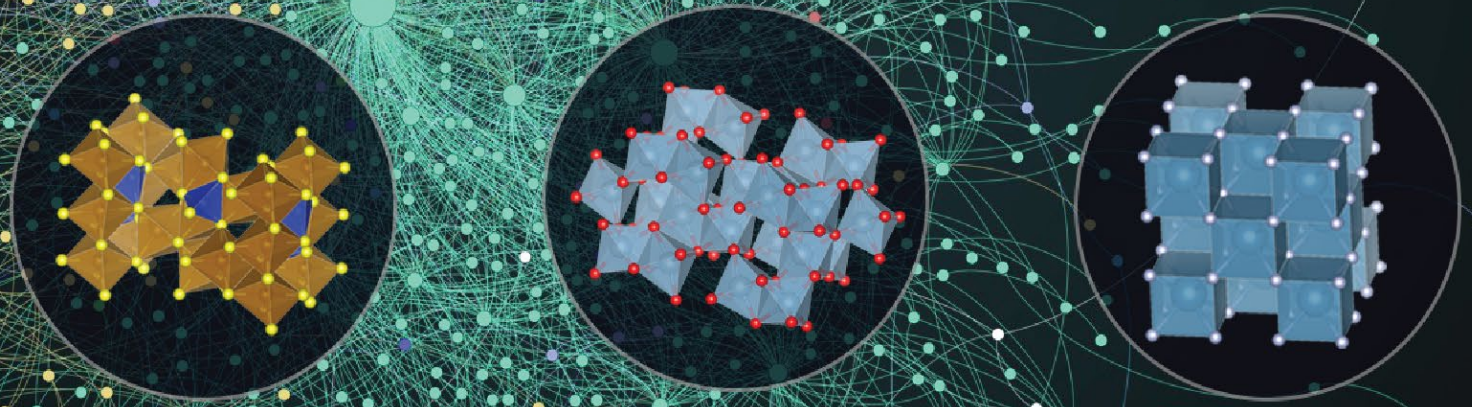
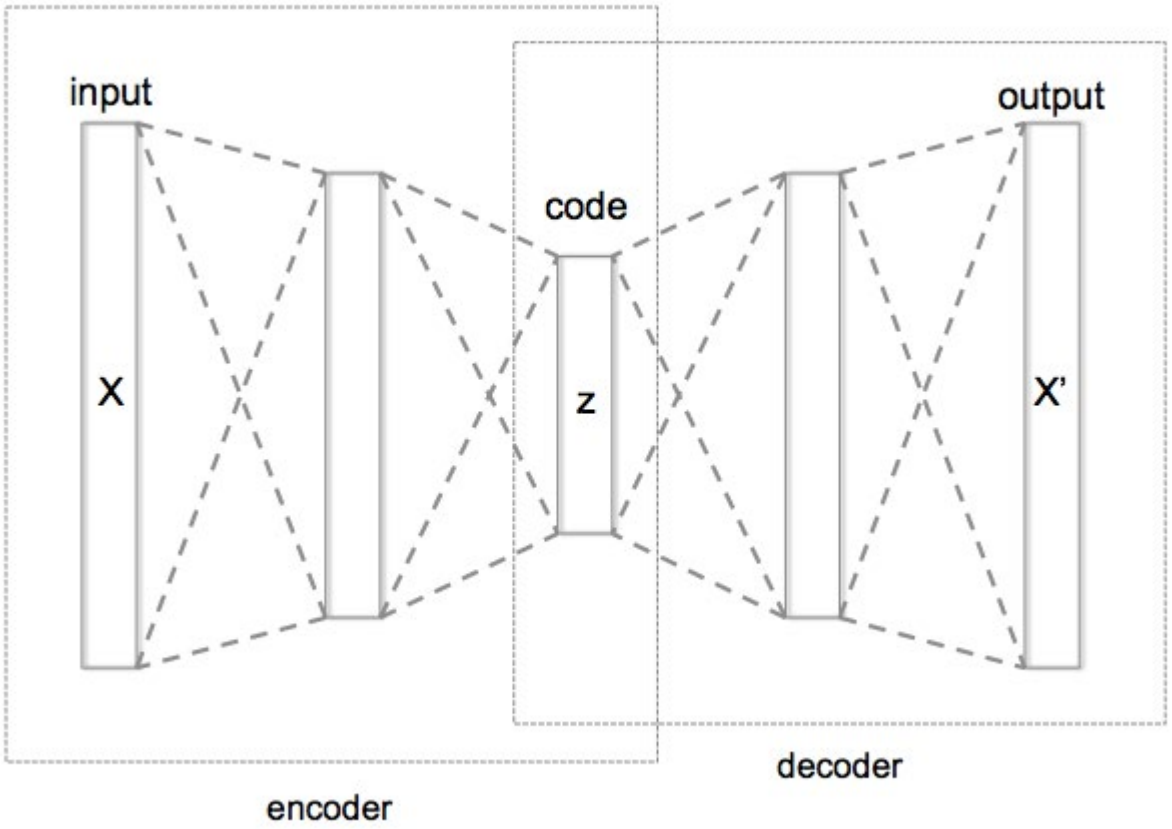
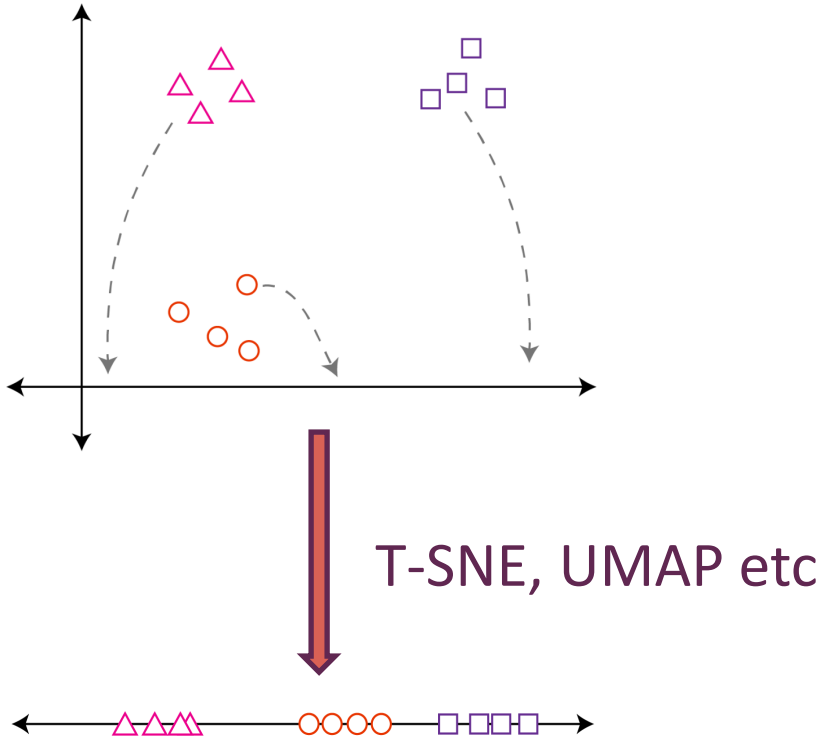
variational autoencoders



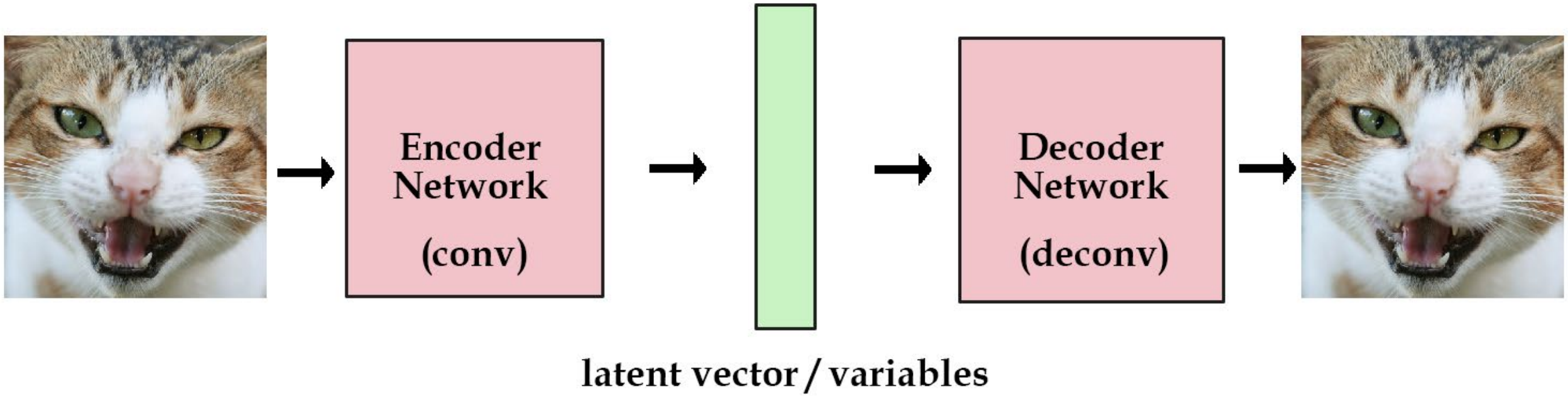
VAEs are modified versions of autoencoders that efficiently encode data





https://en.wikipedia.org/wiki/Autoencoder

Autoencoders are made up of two important steps



https://kvfrans.com/variational-autoencoders-explained/

Technically this is an entirely new type of compression mechanism!

Left: Original image (1419 KB PNG) at ~1.0 MS-SSIM. **Center:** JPEG (33 KB) at ~0.9 MS-SSIM. **Right:** Residual GRU (24 KB) at ~0.9 MS-SSIM. This is 25% smaller for a comparable image quality

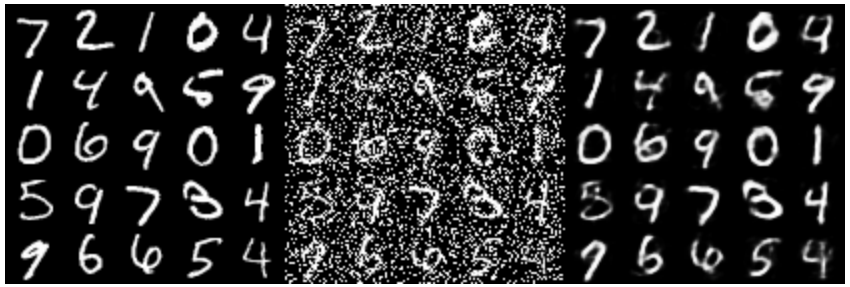
Taking a look around his nose and mouth, we see that our method doesn't have the magenta blocks and noise in the middle of the image as seen in JPEG. This is due to the blocking artifacts produced by JPEG, whereas our compression network works on the entire image at once. However, there's a tradeoff -- in our model the details of the whiskers and texture are lost, but the system shows great promise in reducing artifacts.

Left: Original. **Center:** JPEG. **Right:** Residual GRU.

While today's commonly used codecs perform well, our work shows that using neural networks to compress images results in a compression scheme with higher quality and smaller file sizes. To learn more about the details of our research and a comparison of other recurrent architectures, check out our paper. Our future work will focus on even better compression quality and faster models, so stay tuned!

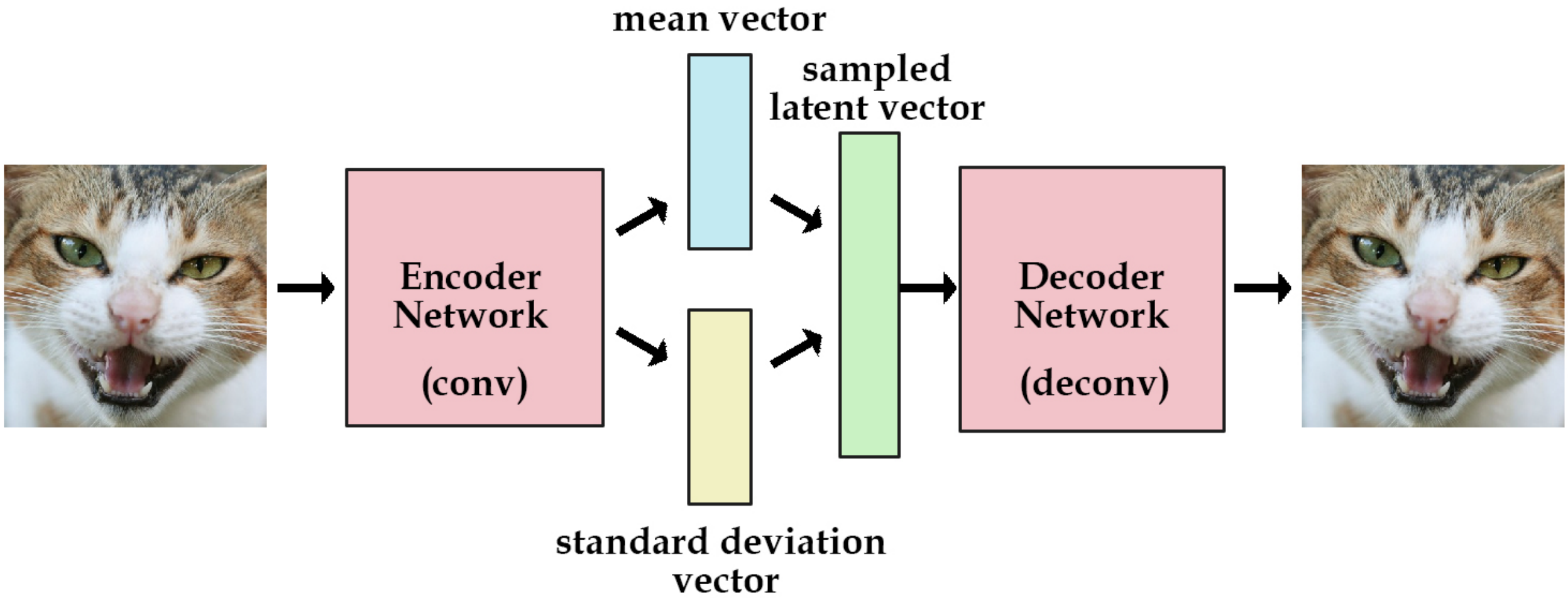
https://ai.googleblog.com/2016/09/image-compression-with-neural-networks.html

Autoencoders can also be used for denoising and neural inpainting



https://towardsdatascience.com/10-papers-you-must-read-for-deep-image-inpainting-2e41c589ced0

So how is a variational autoencoder different?



https://kvfrans.com/variational-autoencoders-explained/

The loss function in VAEs gets modified slightly

Loss function no longer just compares input to output.

Now we have two terms:

First Reconstruction loss (same as autoencoder, but we now have expectation operator because we are sampling from a distribution)

Second "KL divergence" makes sure the distribution you are learning is not too dissimilar from a normal distribution

- forces new distribution to have mean ~0 and stdev ~1

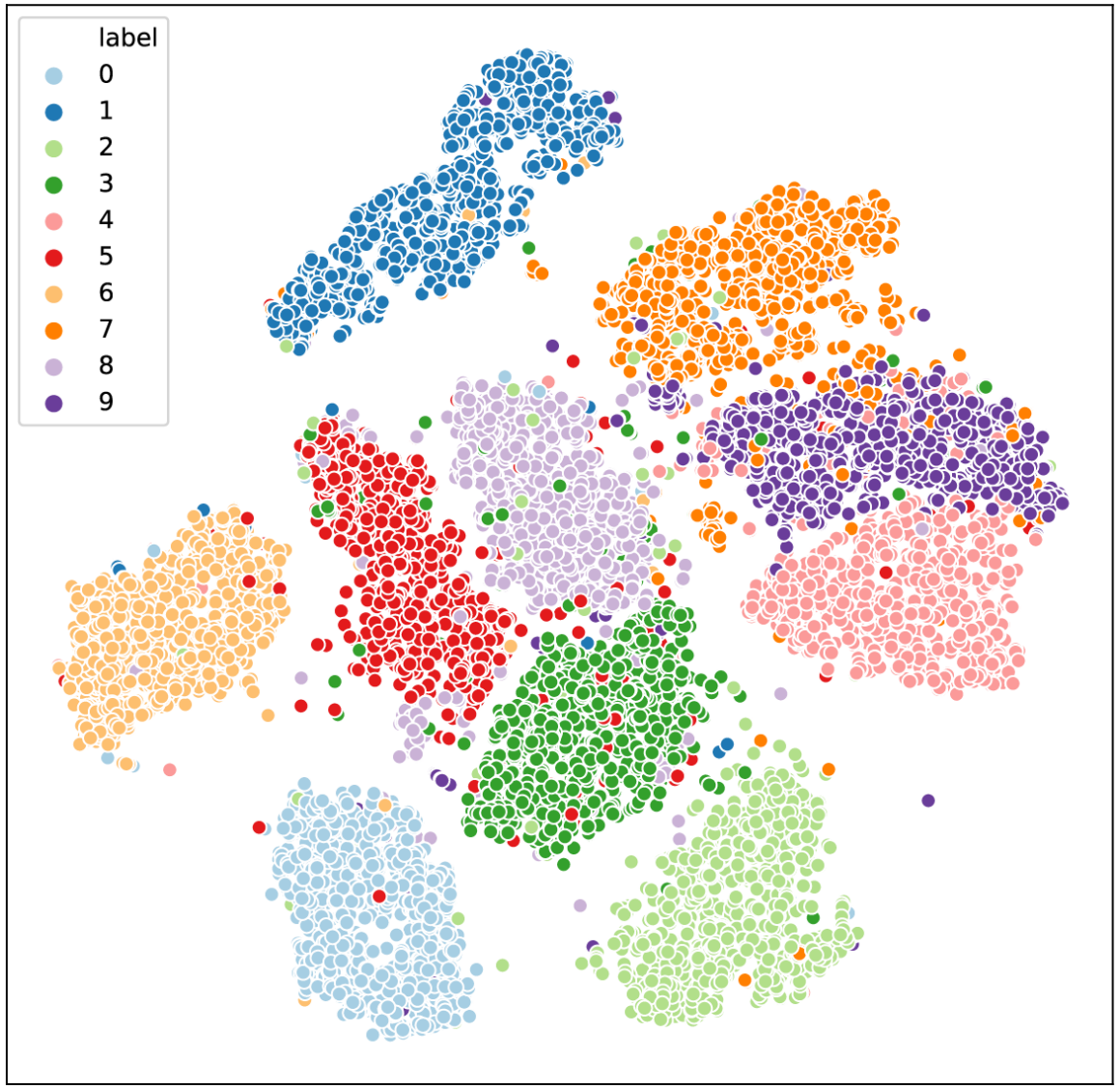
$$\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{z}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}))$$

To make model generative, we need to sample from latent space

Backpropagation would not be able to work through a simple sampling operator

Fix this with “reparameterization trick”

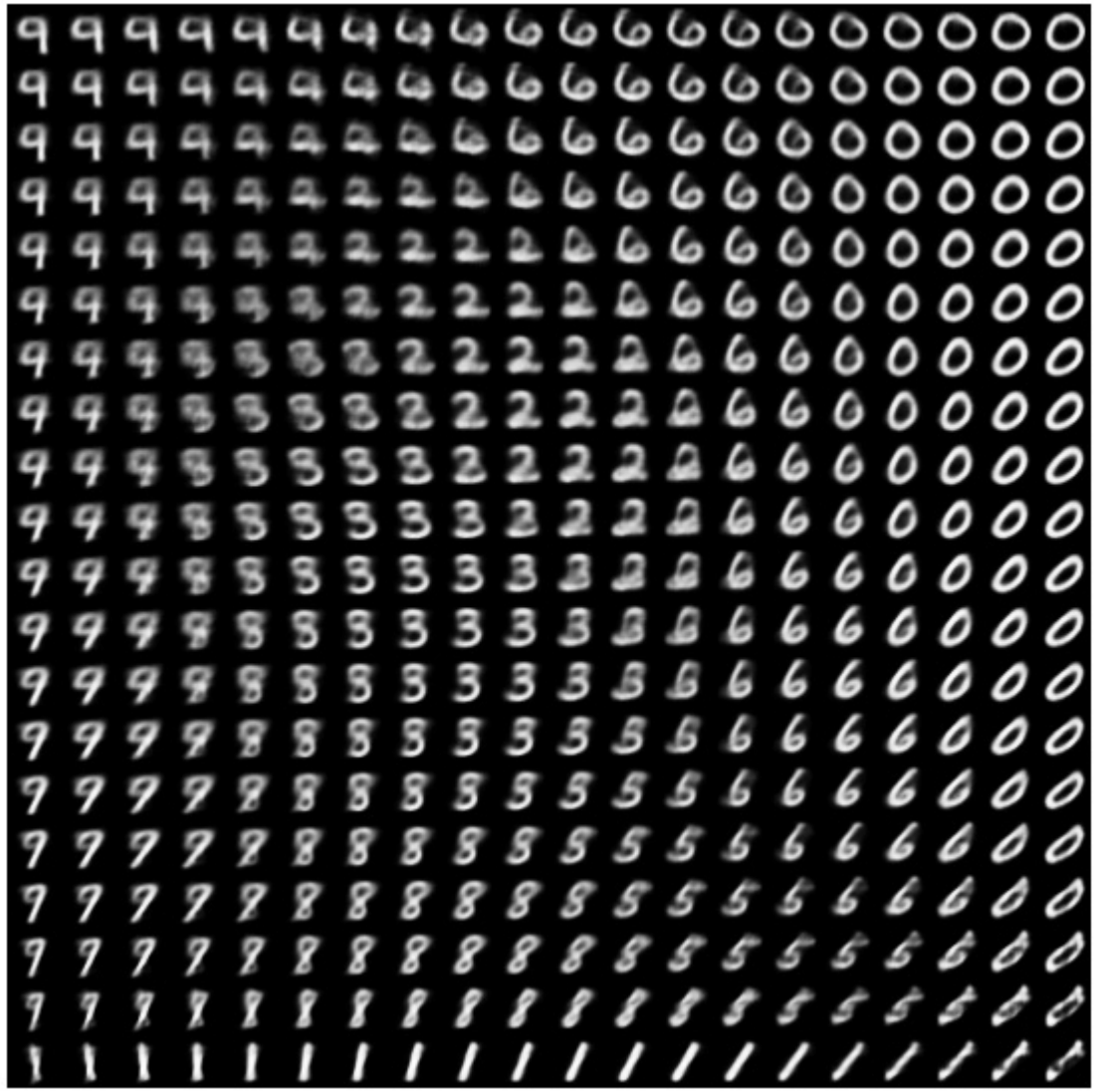
$$vector = \mu + \sigma\epsilon$$



Why can’t we just give latent vector some random numbers?

We would get garbage out. It needs to come from a normal distribution

Done correctly, the latent space is continuous! So we can systematically alter vector to see how output image changes



Disentangled autoencoders are also available

We add one parameter in loss function, β, that measures how much the scaled divergence is present in the loss function

If β is too small, then you get overfitting (high variance)
If β is too large, then you get poor performance (high bias)

$$\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{z}, \beta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta \, D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\big)$$

You can see that increasing β causes disentangled to only use 5 instead of all 10

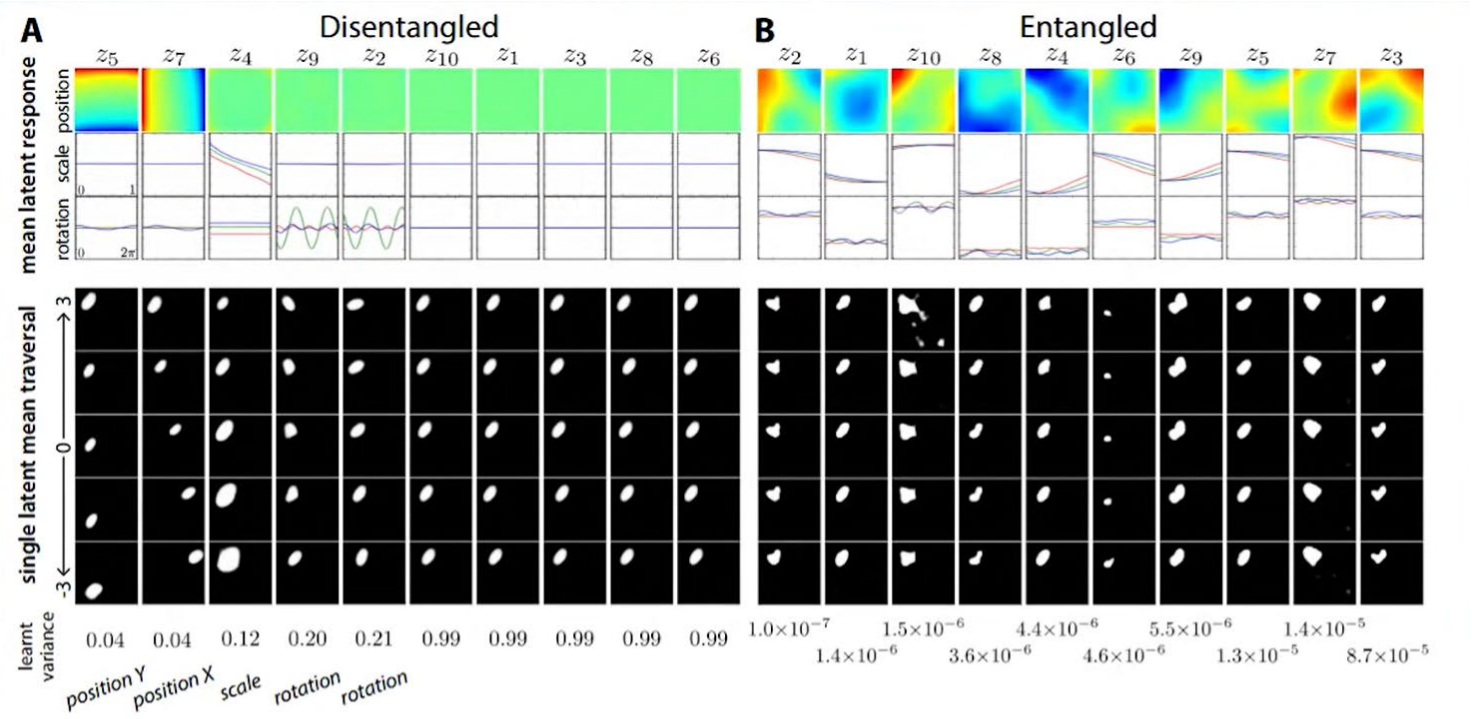


Figure 2: **A:** Disentangled representation learnt with $\beta = 4$. Each column represents a latent z_i, ordered according to the learnt Gaussian variance (last row). Row 1 (position) shows the mean activation (red represents high values) of each latent z_i as a function of all 32x32 locations averaged across objects, rotations and scales. Row 2 (scale) shows the mean activation of each unit z_i as a function of scale (averaged across rotations and positions). Row 3 (rotation) shows the mean activation of each unit z_i as a function of rotation (averaged across scales and positions). *Square* is red, *oval* is green and *heart* is blue. Rows 4-8 (second group) show reconstructions resulting from the traversal of each latent z_i over three standard deviations around the unit Gaussian prior mean while keeping the remaining 9/10 latent units fixed to the values obtained by running inference on an image from the dataset. After learning, five latents learnt to represent the generative factors of the data, while the others converged to the uninformative unit Gaussian prior. **B:** Similar analysis for an entangled representation learnt with $\beta = 0$.

http://jck.bio/disentangling_a_latent_space/

We can see that specific latent factors do specific things

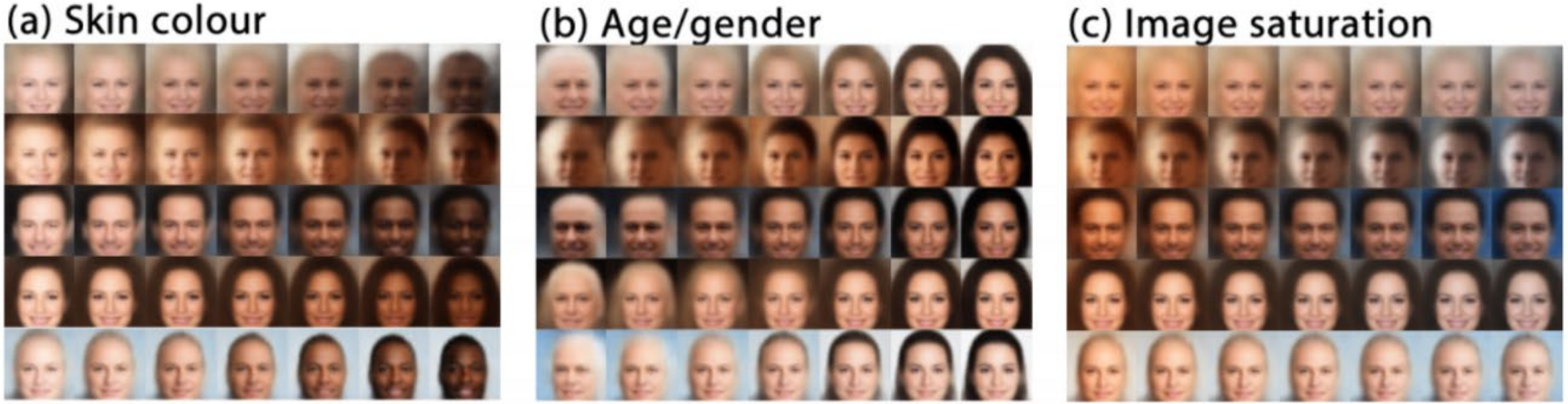


Figure 4: **Latent factors learnt by β-VAE on celebA:** traversal of individual latents demonstrates that β-VAE discovered in an unsupervised manner factors that encode skin colour, transition from an elderly male to younger female, and image saturation.

Comparing GANs with VAEs

Similarities: they both have two components. GAN has generator and discriminator. VAE has encoder and decoder

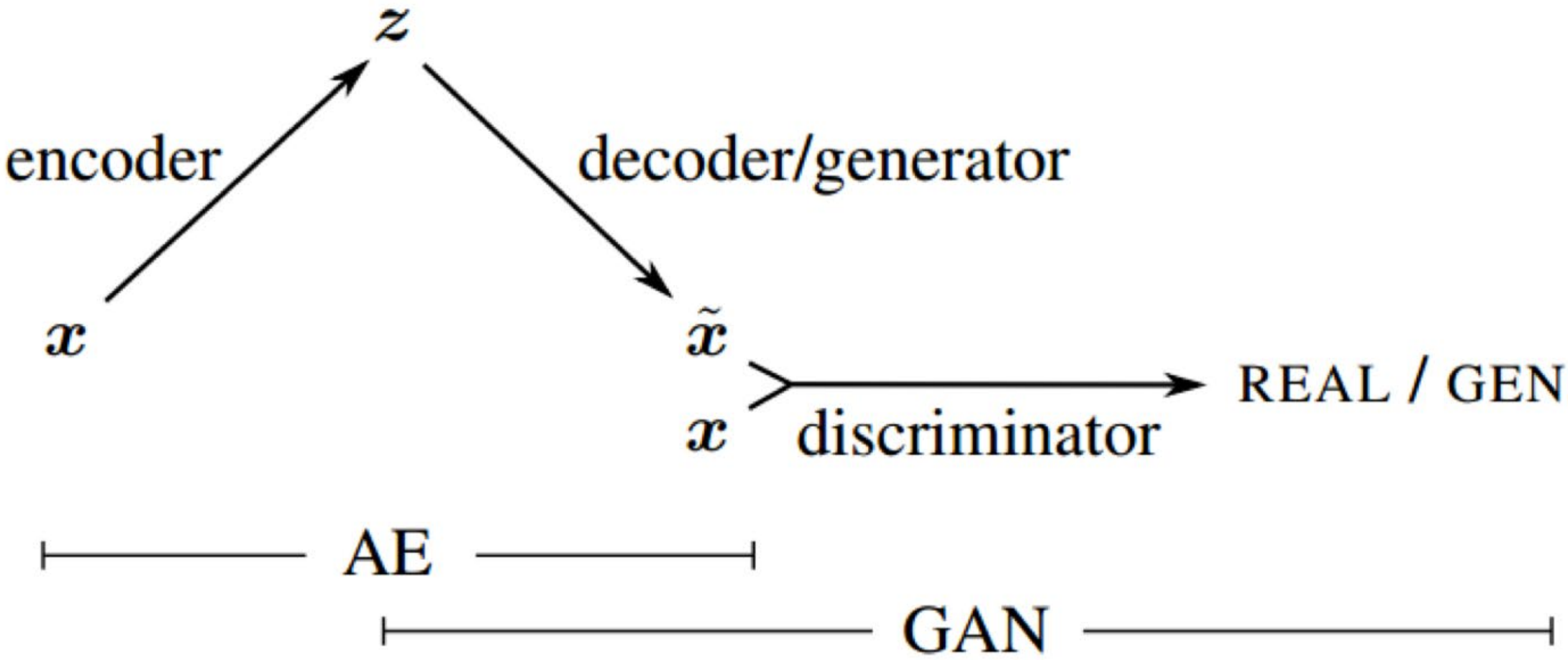
GAN plays minimax game whereas VAE tries to minimize reconstruction and latent loss

GAN not very stable because it requires finding “Nash Equilibrium” or end of game point and we don’t know how to do that

VAEs have a closed form end of training solution (minimize loss)

How good are they? VAEs are ok in theory, but make blurry images, GANs are much sharper because training is more empirical (trial and error)

Recently people have combined VAEs and GANs to create VAE-GANs



https://arxiv.org/abs/1512.09300

Comparing GANs with VAEs

Bayesian inference