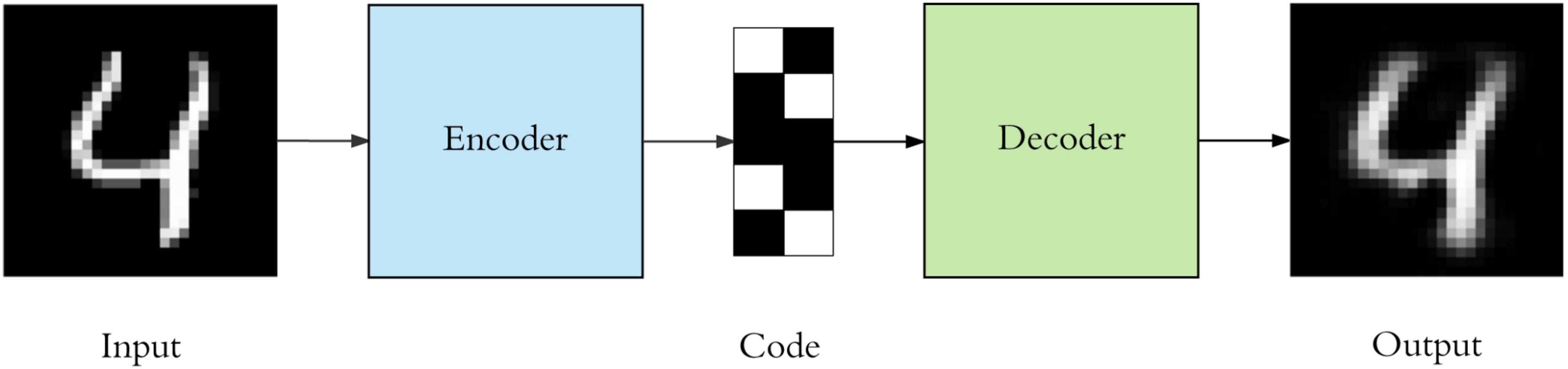
Autoencoders, VAE, GAN

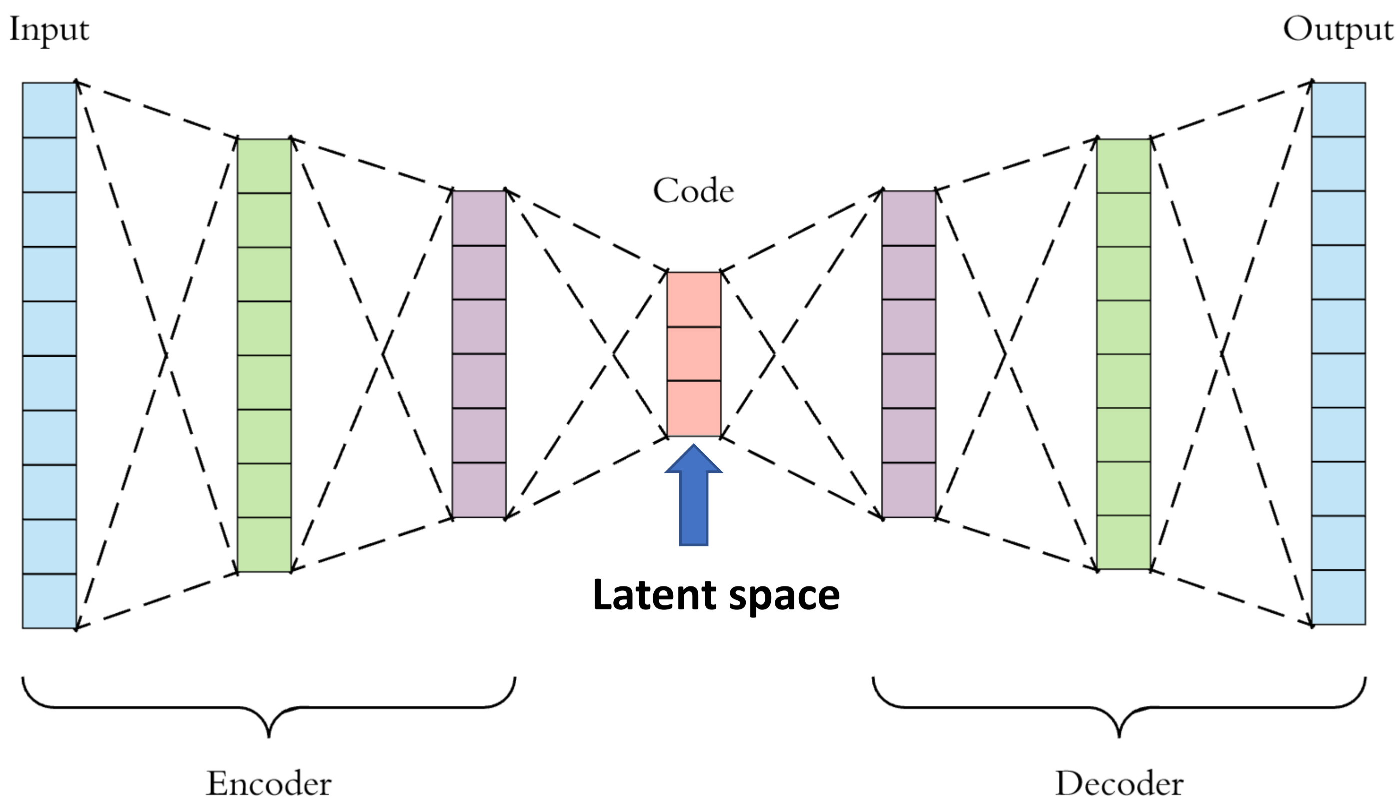
Attila Bagoly

- Unsupervised learning: no-labels (just X)
- Autoencoder: $\mathcal{M}: \mathrm{X} \rightarrow X$
- Tries to learn "identity" function, but we add **constraints**
- **Constraints:** e.g. less neurons in the middle (lower dim. repr.)
- Design:
  - Encoder: encodes the input into a lower dimensional space
  - Decoder: decodes the input from the lower dimensional space to the original space
- PCA: linear projection; Autoencoder: non-linear projection



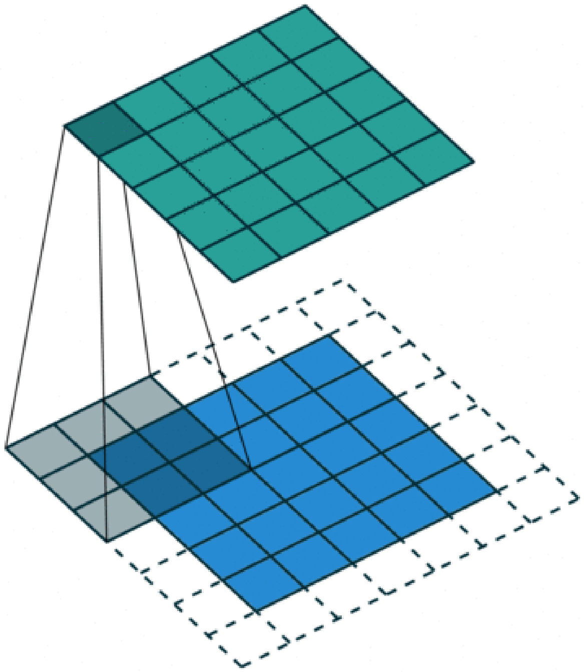
Autoencoders

- Loss: e.g. MSE, binary-crossentropy (minimizing the difference between the input and output)
- Non-triviality: lower dimensional representation



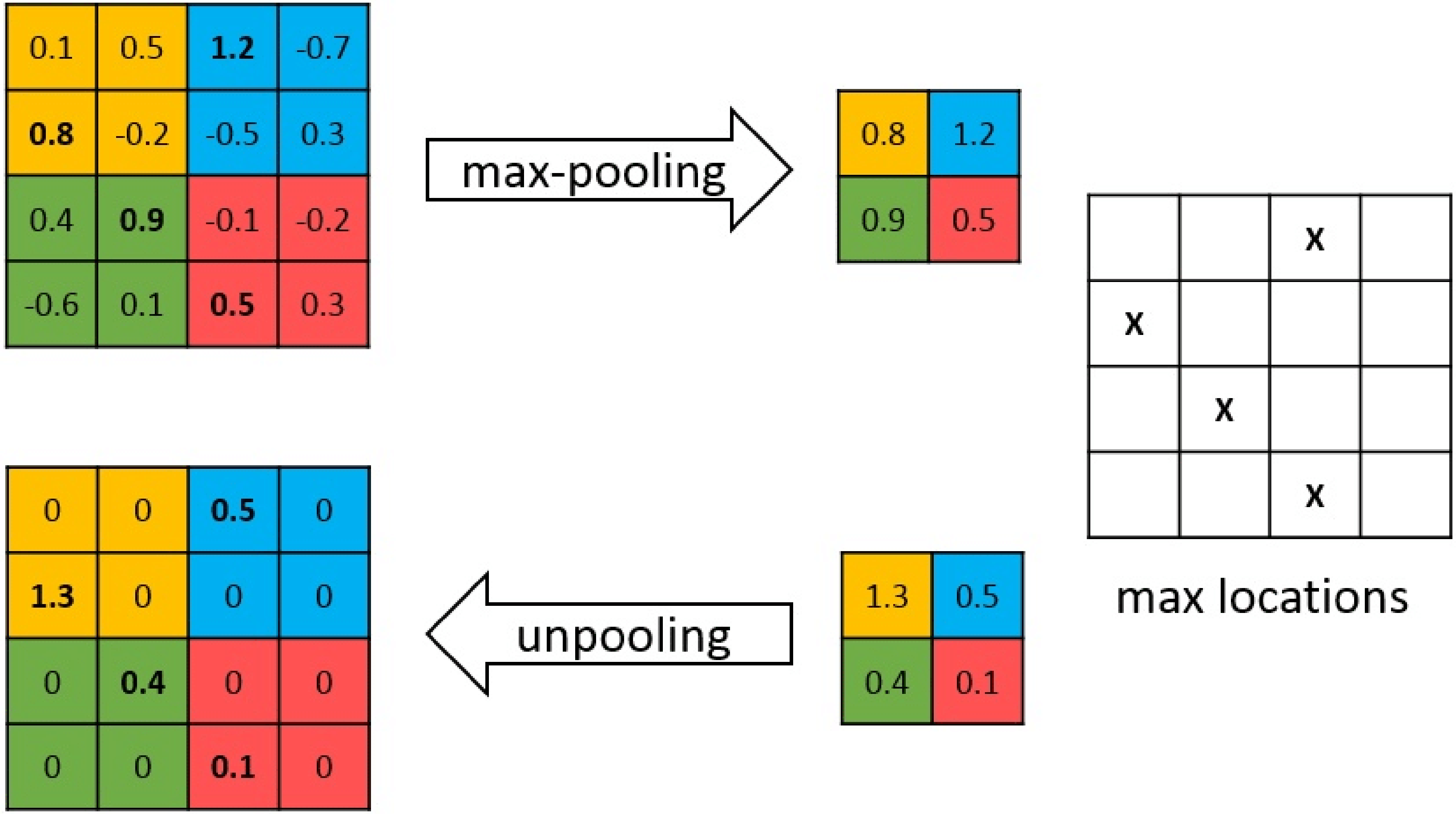
Convolutional autoencoder

- The encoder is a classical convolutional network
- Decrease the spatial dimension:
  - MaxPooling, AvgPooling: not learnable
  - Convolution with stride >1: learnable (we learn how to do the downsampling, tiny details, preserves spatial information)
  - Size calculation: $d' = \frac{d-f+2p}{s} + 1$
  - Padding: 'valid': p=0; 'same': $d = d' \rightarrow p$



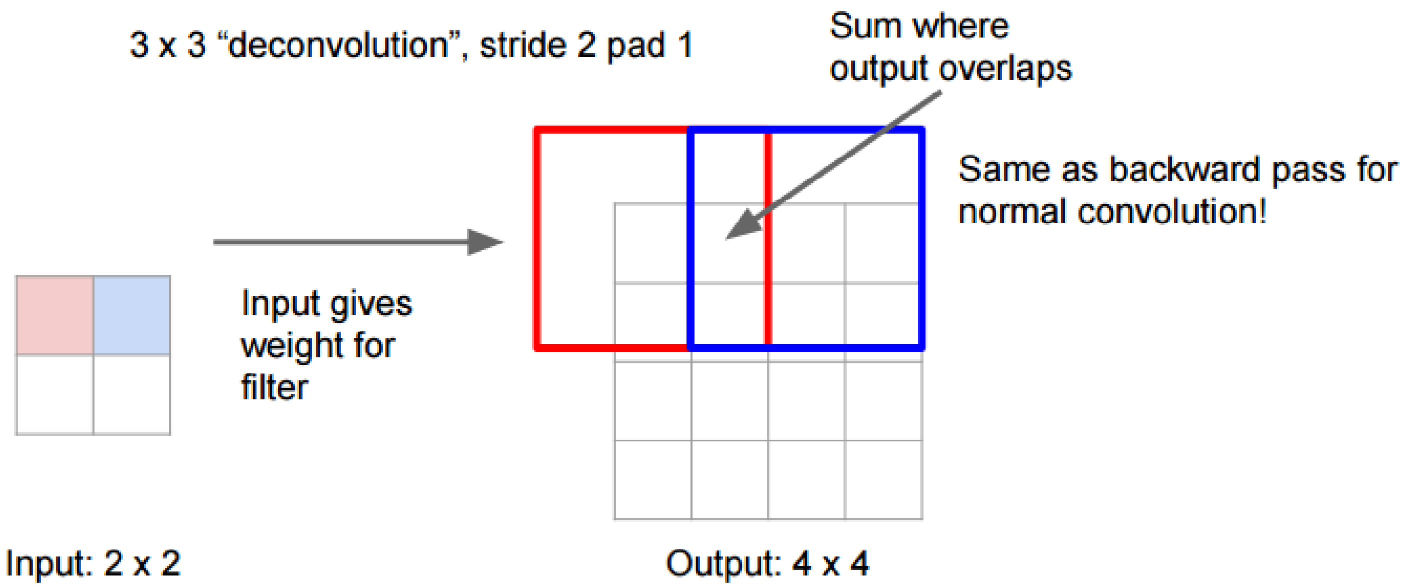
- What about in the decoder?
- We need to reverse Pooling and Conv
- Encoder: downsampling
- Decoder: upsampling

- Pooling: remember the max positions
- Unpooling: lot of zeros, except in the max positions
- Fixed layer: doesn’t learn



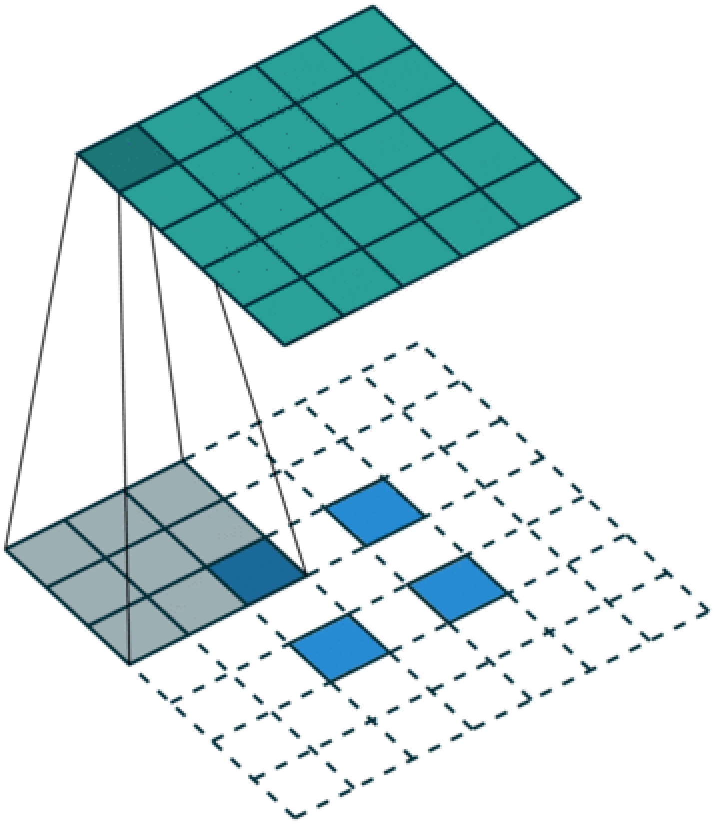
Upsampling: Transposed convolution

- Other name: Fractionally strided convolution
- Sometimes called: deconvolution
- But deconvolution (inverse convolution) exists and it is mathematically very different: but both results the same dimension. Real deconv not used in deep learning!
- Transposed convolution: learnable layer (learns how to do the upsampling)



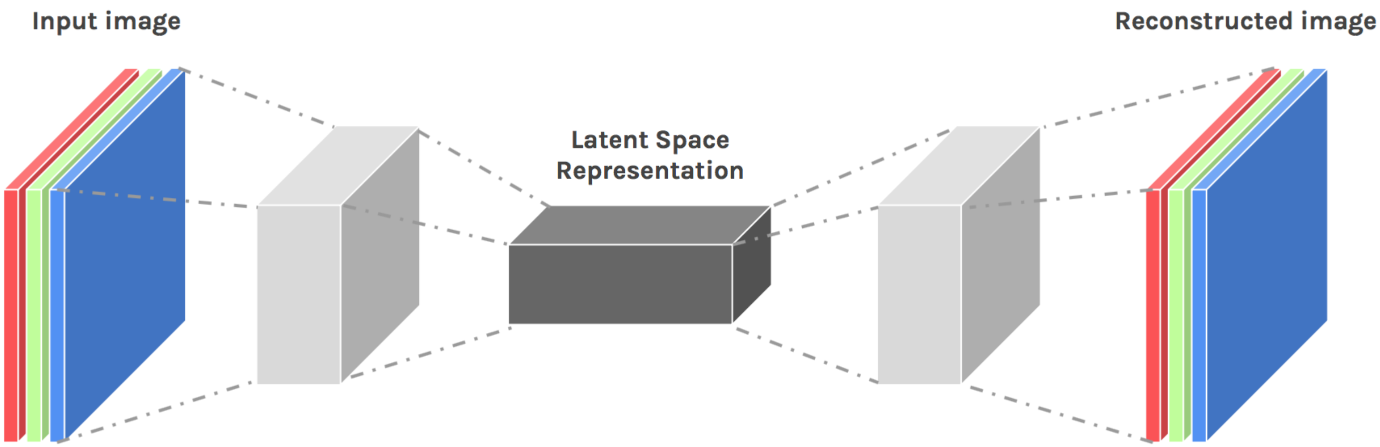
Upsampling: Transposed convolution

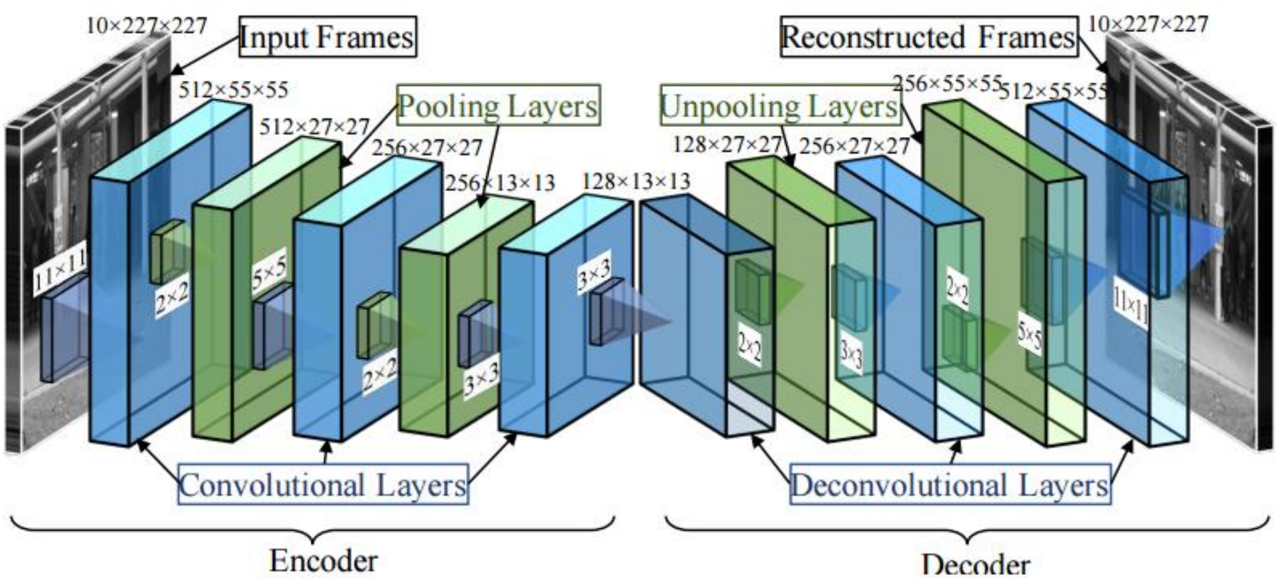
- Transposed convolution is also a convolution
- But with some fancy padding



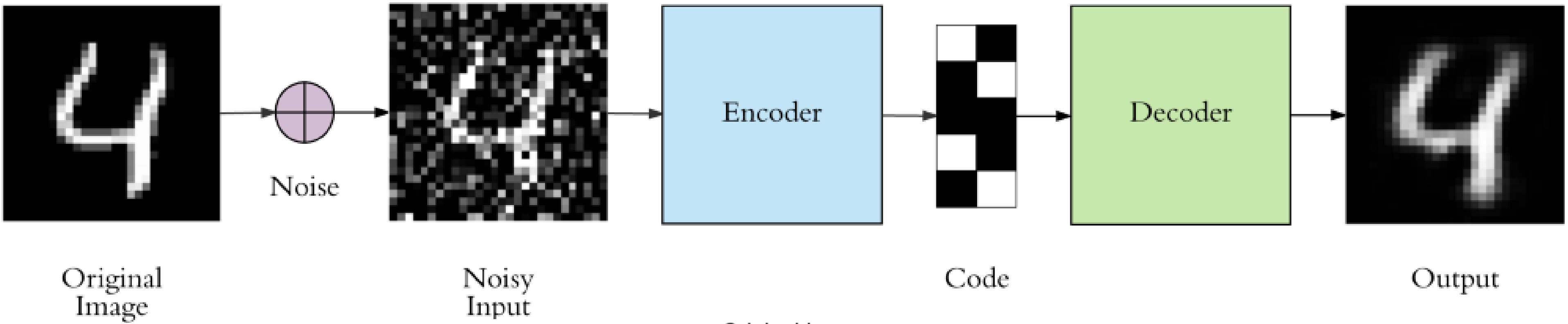
Convolutional autoencoder

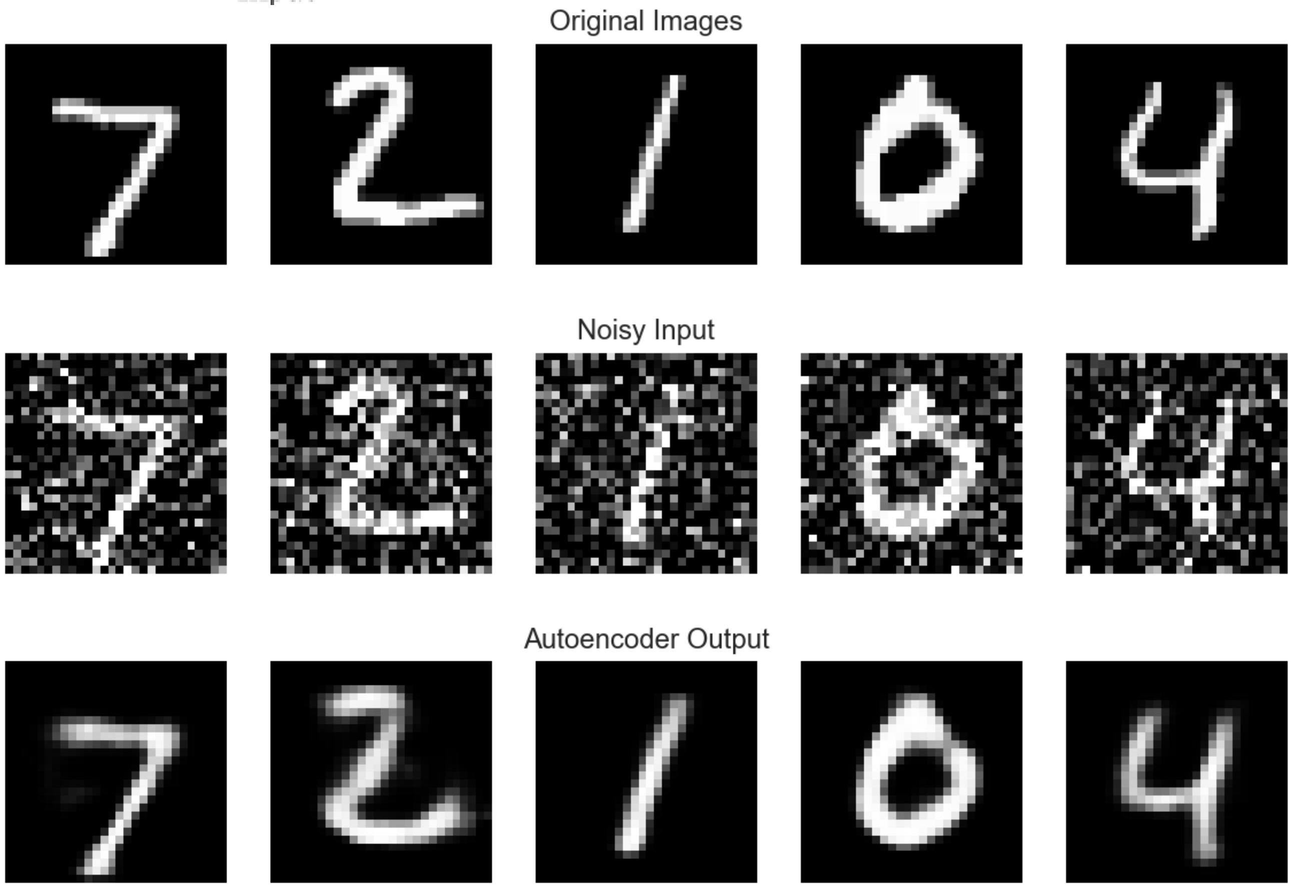
- Decoder: upsample with 'max unpooling' or with transposed convolution





Denoising autoencoder

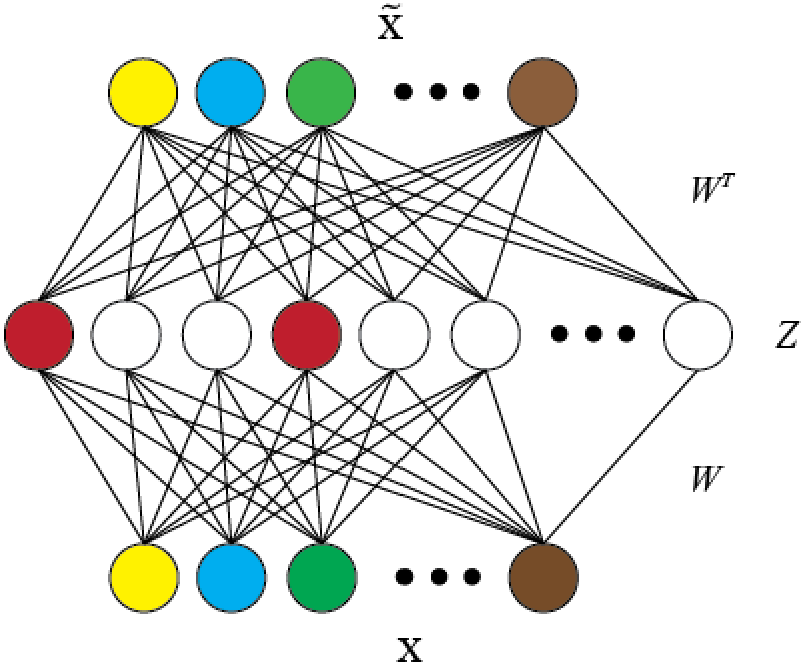




Sparse autoencoder

- Average activation in 2. layer:

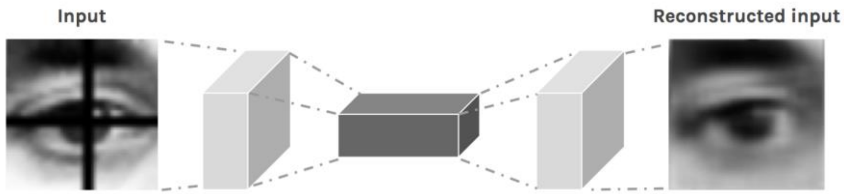
$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} \left[a_j^{(2)} \left(x^{(i)} \right) \right]$$

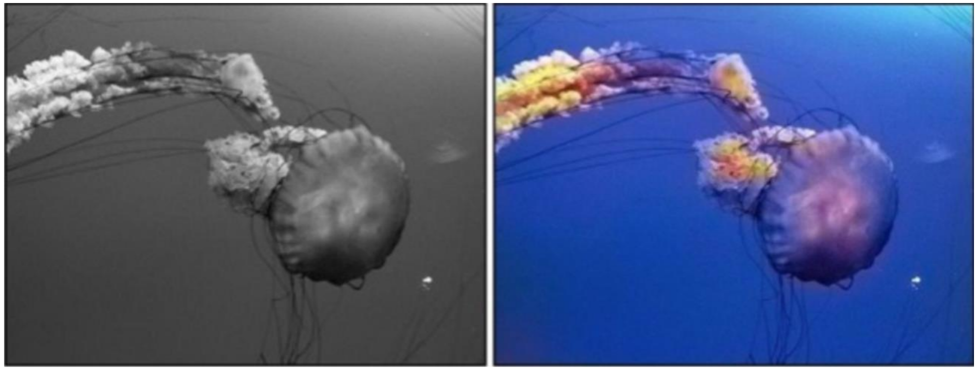


- Sparsity parameter: ρ
- Constraint: $\hat{\rho}_j = \rho$
- We want: average activation of the hidden unit to be close to: ρ
- Constraint: loss penalty term

$$\sum_{j=1}^{s_2} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j} = \sum_{j=1}^{s_2} \mathrm{KL}(\rho \| \hat{\rho}_j)$$

- Pretraining
- Data compression: hashing
- Image search
- Information retrieval
- Denoising, reconstruction
- Image colorization
- Generating higher resolution images

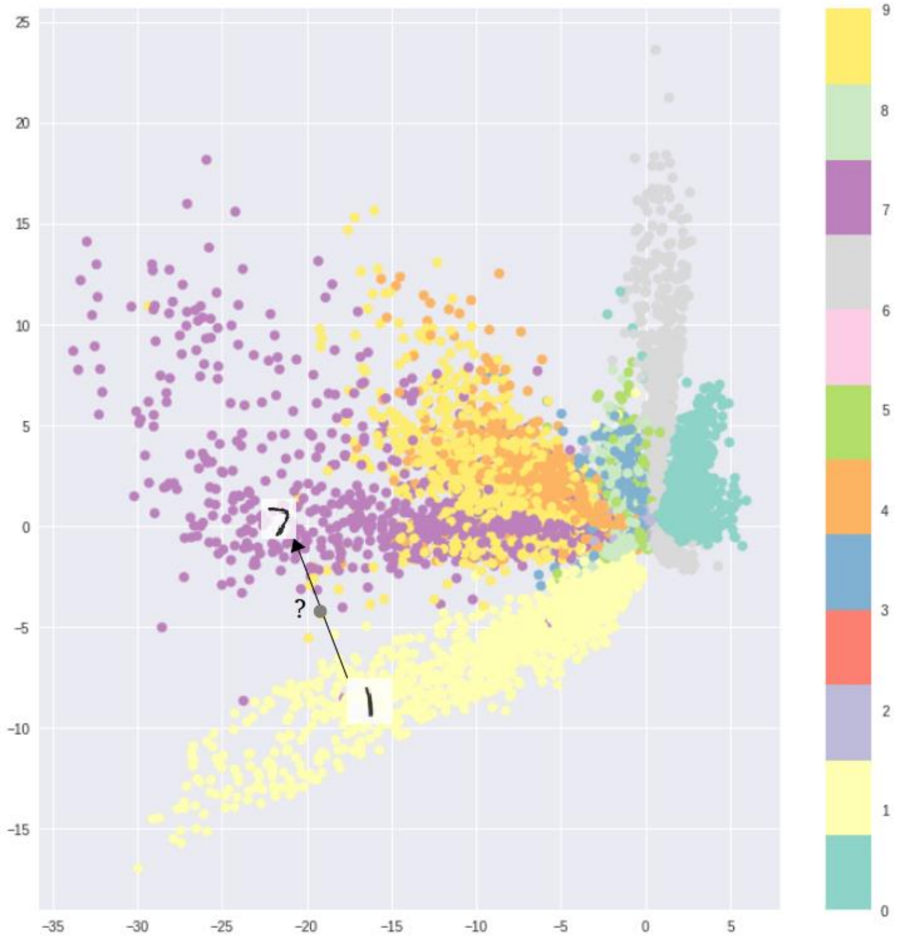




Autoencoder demo notebooks

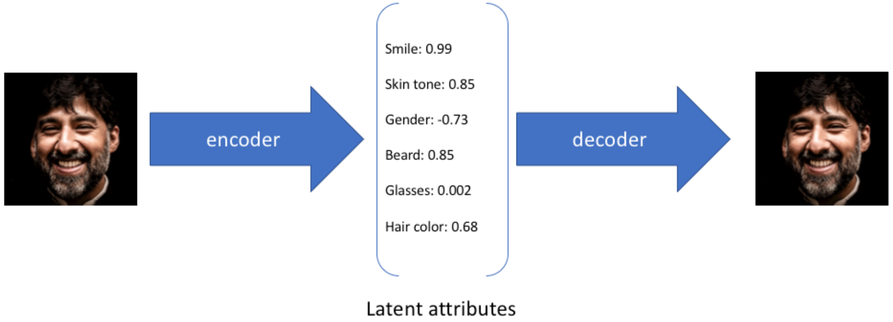
Problems with autoencoder

- Latent space not continuous
- Not well structured: hard to interpret it
- Clusters: easier to decode
- Gaps: if we change the latent code a little bit, we can get totally unrealistic images
- Happy, not happy encoded -> what if we move in "happy direction"?
- We can fall into gaps
- Can't generate new images

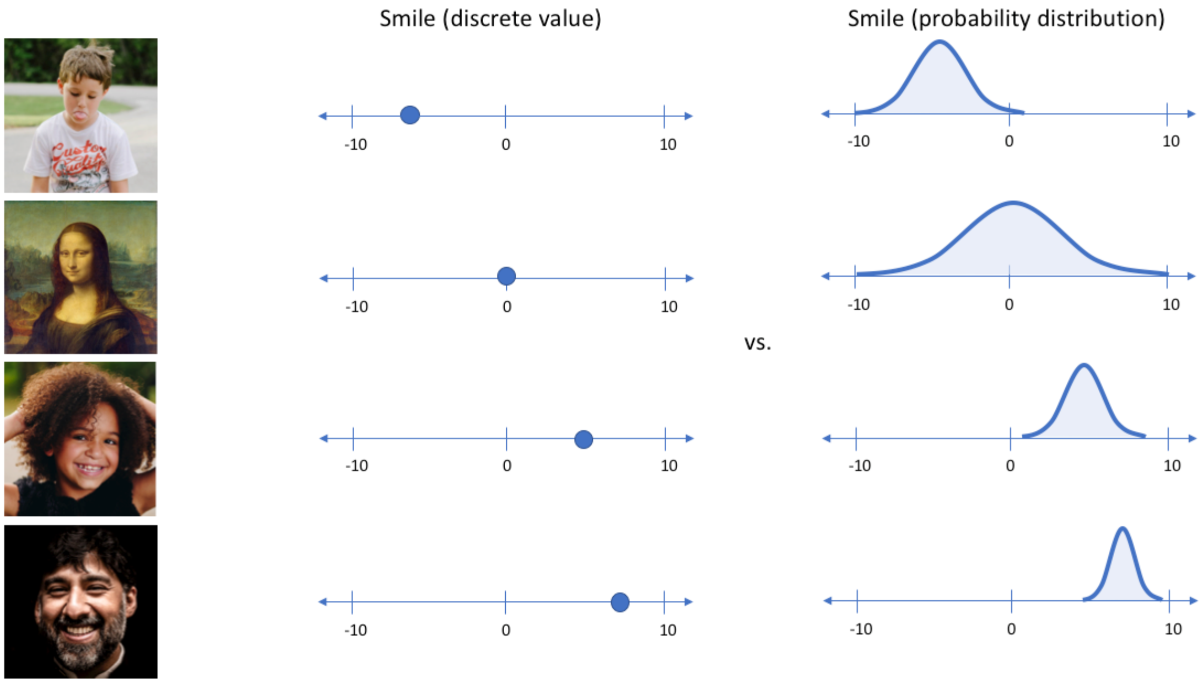


Variational autencoder

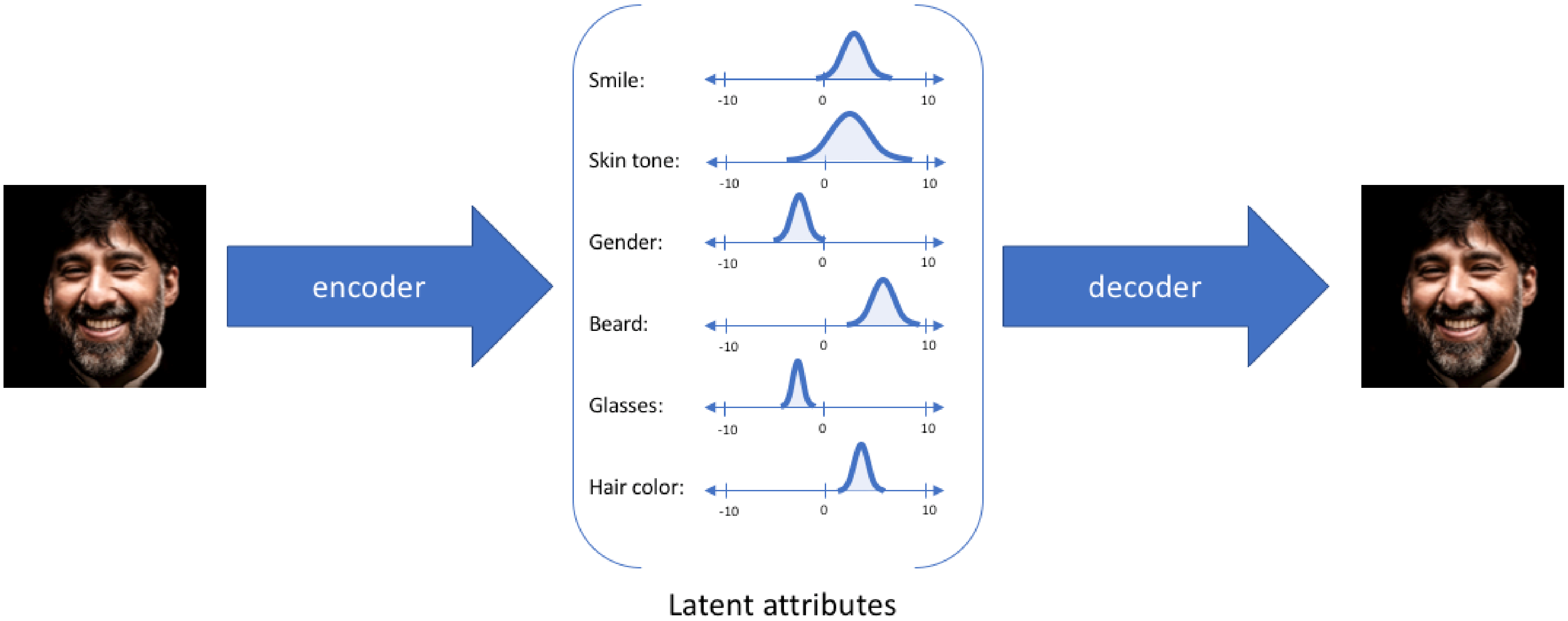
- Ideal autencoder:



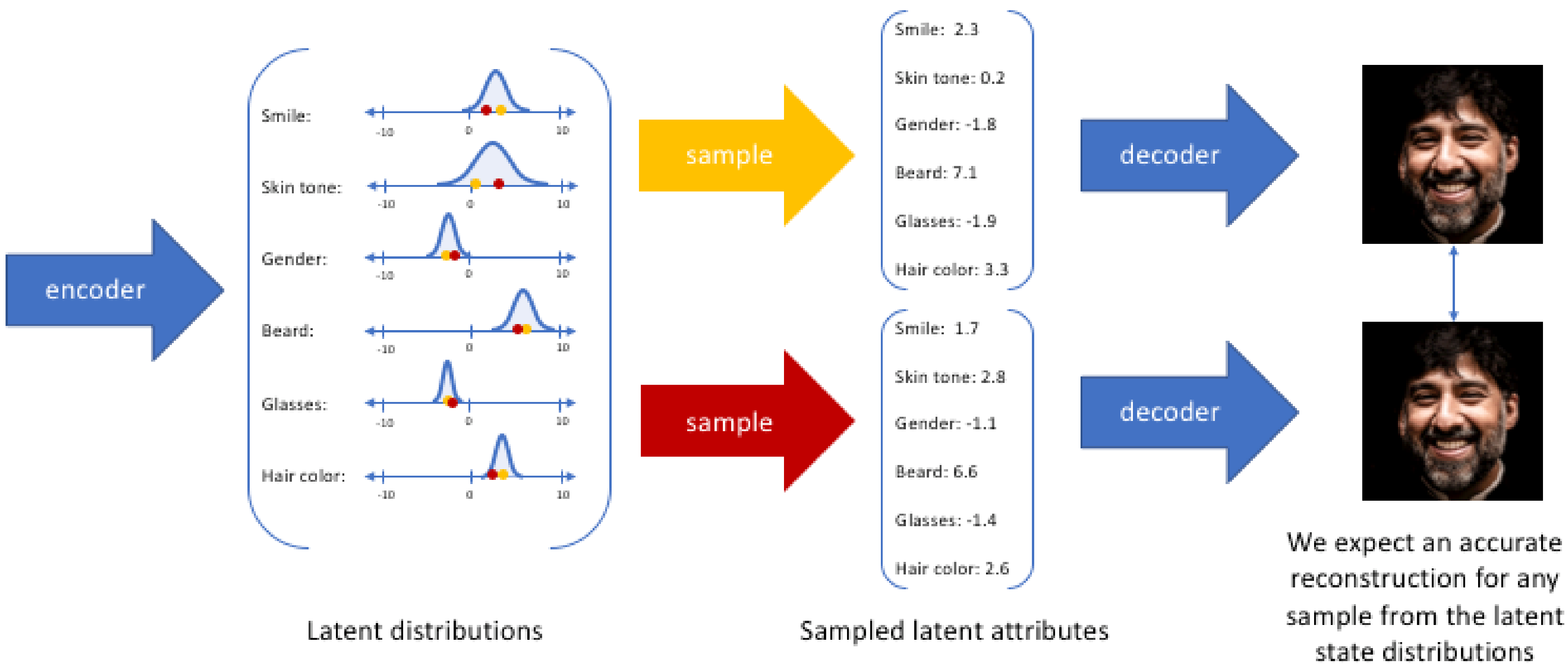
- **VAE**: continuous, structured latent space by design
- Each latent variable range of possible values
- Each latent var for given input: prob. dist



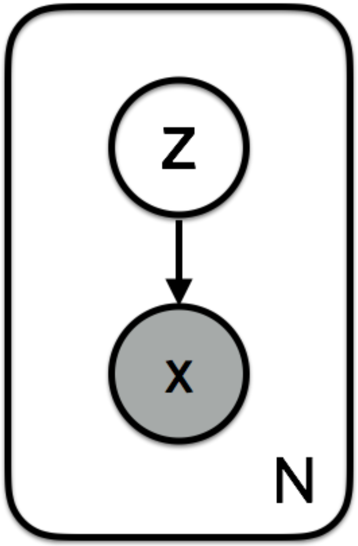
- Given input: each latent variable is a probability distribution
- Encoder: encodes the input to prob. distributions
- Decoder: decodes randomly sampled z from these distrs.



- Encoder: range of values; Decoder: random sample
  smooth latent representation
- Nearby in latent space: very similar reconstruction



- We have latent variable z, and observation x
- Latent variable: $z \sim p(z)$ (prior)
- Draw datapoint: $x \sim p(x|z)$ (likelihood)
- We want to learn what is z given x
- In other words: $p(z|x)$:



$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x|z)p(z)}{\int p(x|z)p(z)dz}$$

- Problem: integral intractable
- Solution: variational inference: approximate posterior with q$(z|x)$, which has tractable distribution
- We choose the parameters of q to be a good approximate

VAE

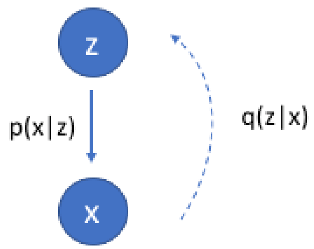
- We need $p(z|x)$, but we can't calculate it
- We approximate it with $q(z|x)$
- We want to q$(z|x)$to be close to $p(z|x)$:

$$\min \mathrm{KL}\big(q(z|x)|p(z|x)\big)$$

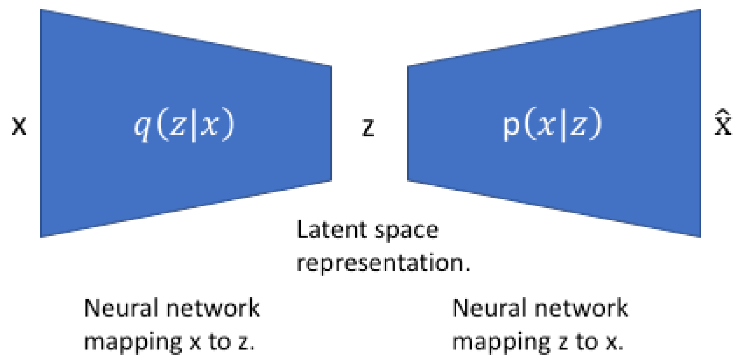
- After some calculation we will get:

$$\mathbb{E}_{q(z|x)} \log p(x|z) - \mathrm{KL}\big(q(z|x)|p(z)\big)$$

- First term: reconstruction error
- Second: we want $q(z|x)$ to be close to the prior



We'd like to use our observations to understand the hidden variable.



VAE

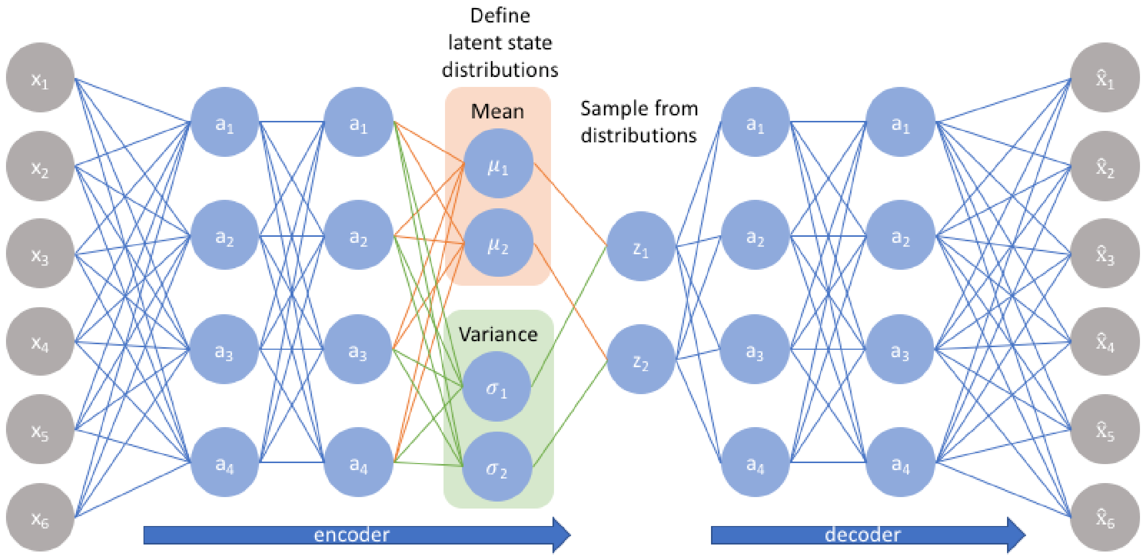
- Autoencoder: $\mathcal{L}(\hat{x}, x)$ (gaps in latent code z)
- Variational autoencoder ($\sum$ each dim. in latent space):

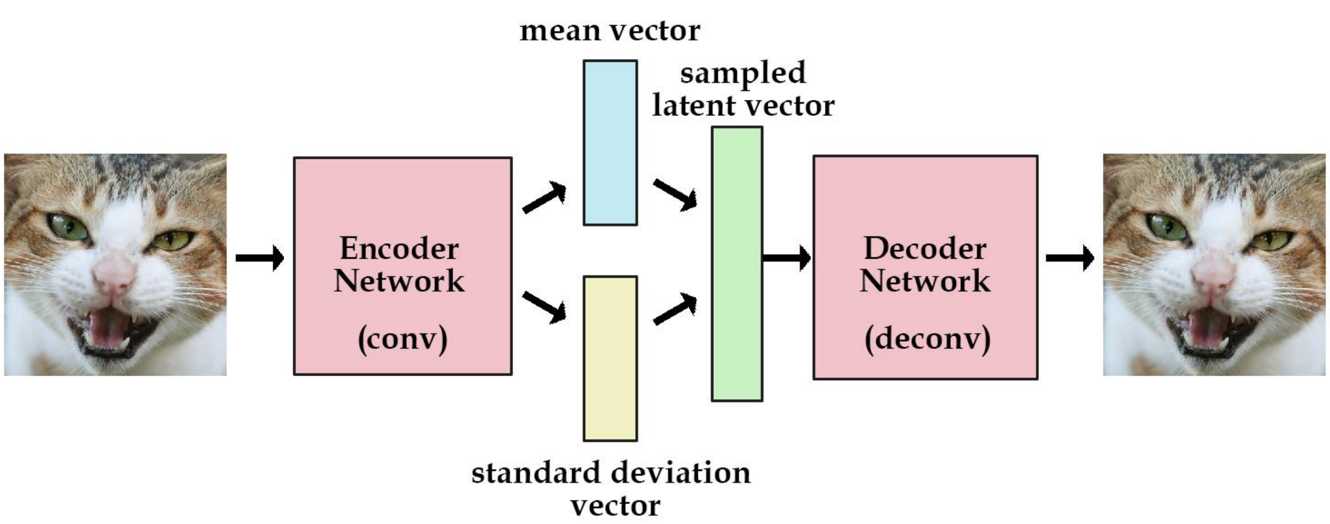
$$\mathcal{L}(\hat{x}, x) + \sum_j \mathrm{KL}\left(q_j(z|x) | p(z)\right)$$

- Prior: $p(z) \sim \mathcal{N}(0, I)$
- q: $q(z|x) \sim \mathcal{N}\left(\mu(x), \Sigma(x)\right)$
- Kullback-Leibler divergence (k dimension):

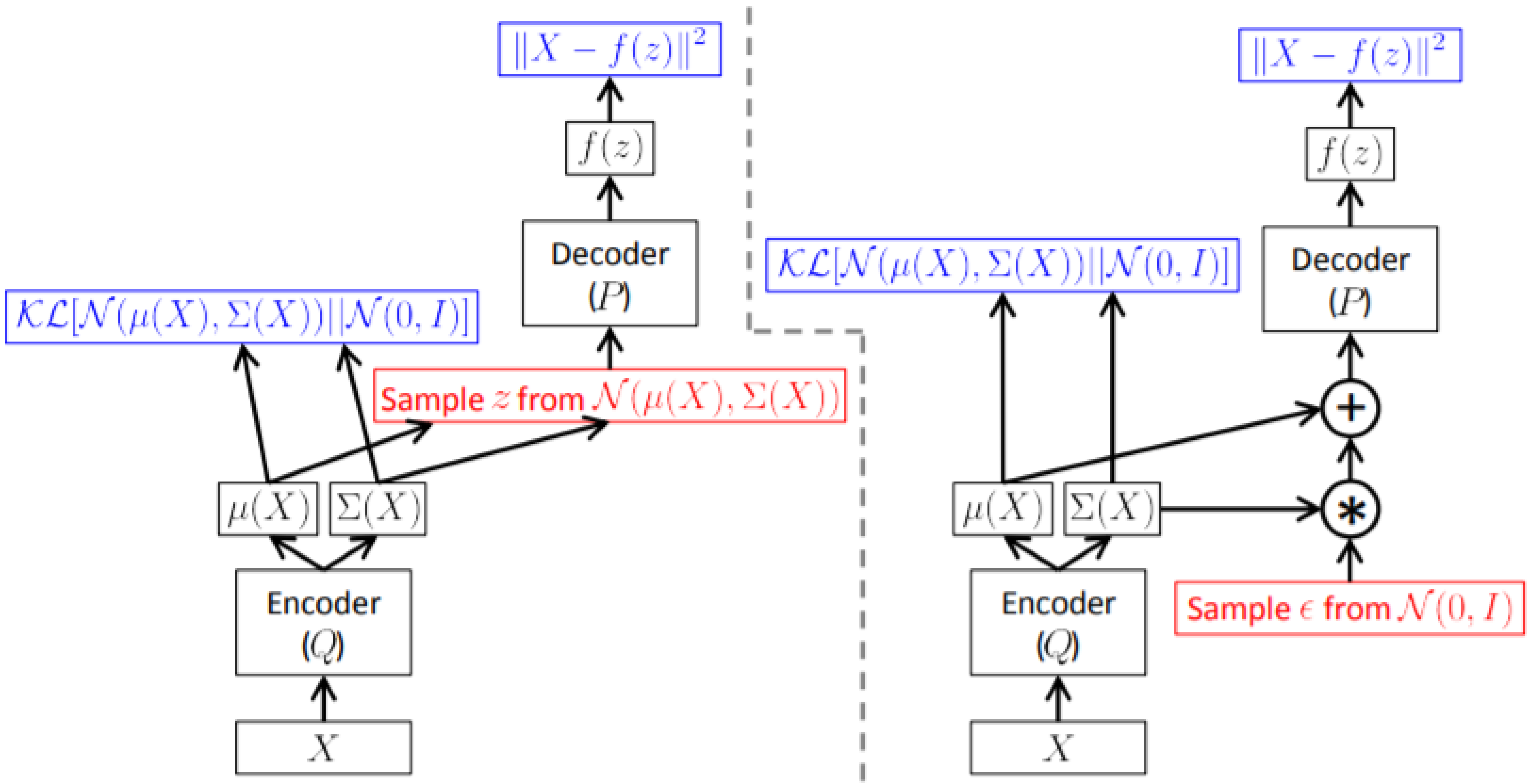
$$\mathrm{KL}\left(\mathcal{N}\left(\mu(x), \Sigma(x)\right) | \mathcal{N}(0, I)\right) = \frac{1}{2}\left(\mathrm{tr}\,\Sigma(x) + \left(\mu(x)\right)^T \mu(x) - k - \log\det\Sigma(x)\right)$$

VAE architecture



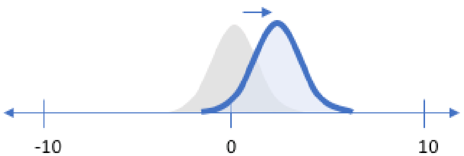


VAE architecture



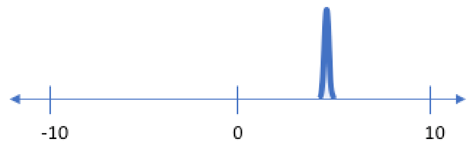
VAE latent space

Penalizing reconstruction loss encourages the distribution to describe the input



Our distribution deviates from the prior to describe some characteristic of the data

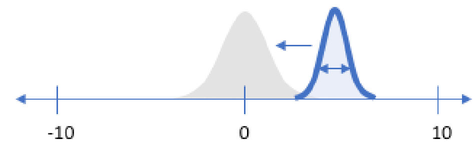
Without regularization, our network can “cheat” by learning narrow distributions



With a small enough variance, this distribution is effectively only representing a single value

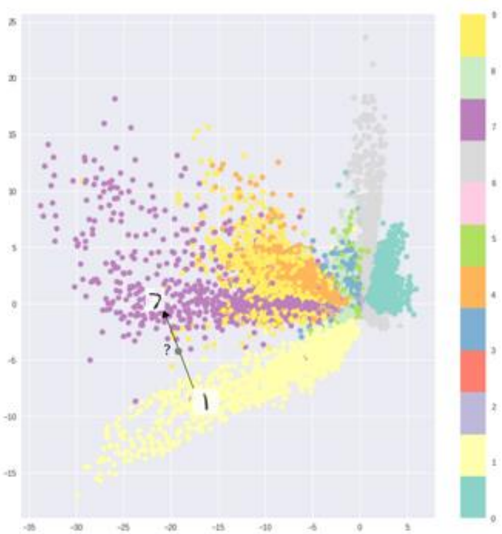
Penalizing KL divergence acts as a regularizing force

Attract distribution to have zero mean

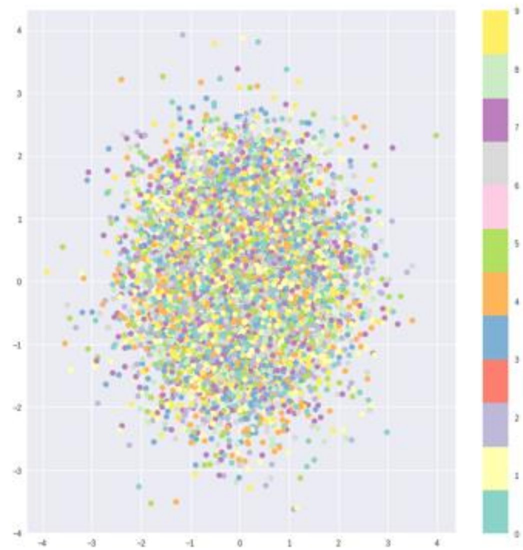


Ensure sufficient variance to yield a smooth latent space

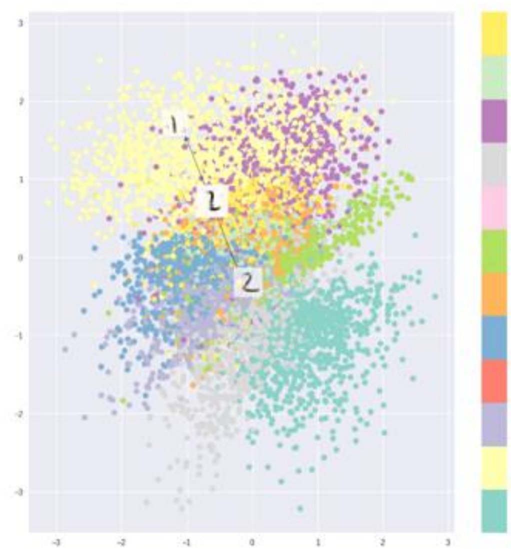
Only reconstruction loss



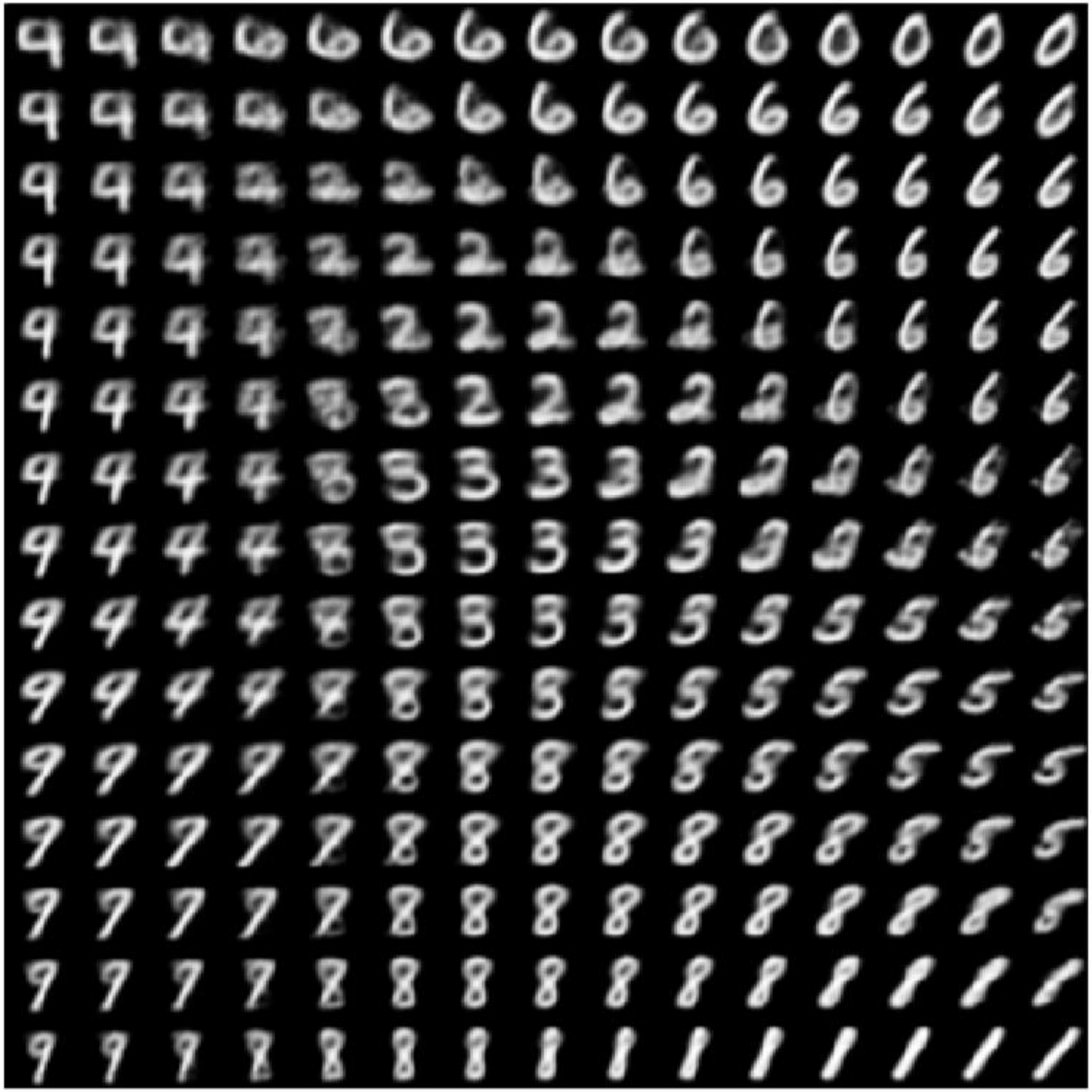
Only KL divergence



Combination

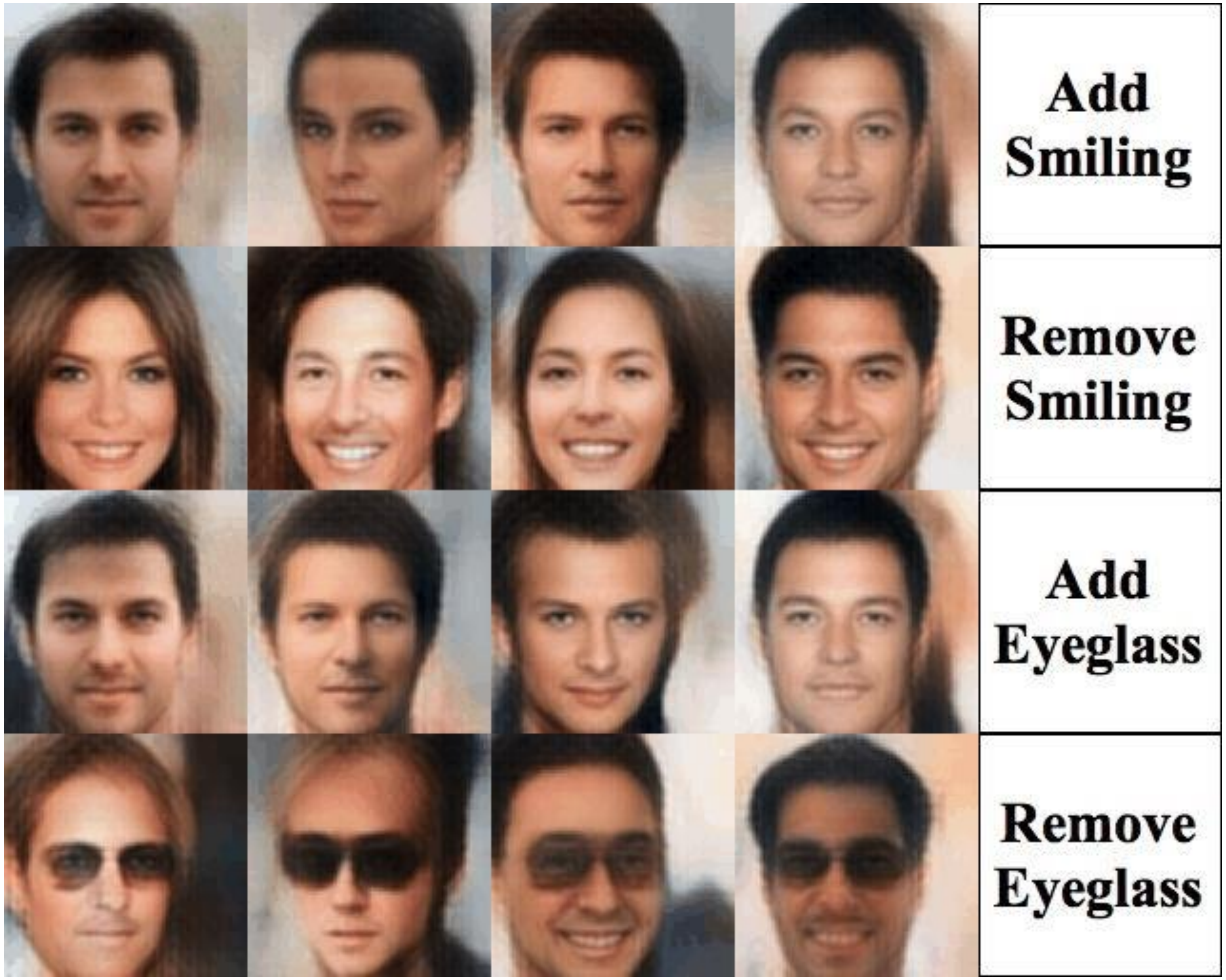


VAE mnist



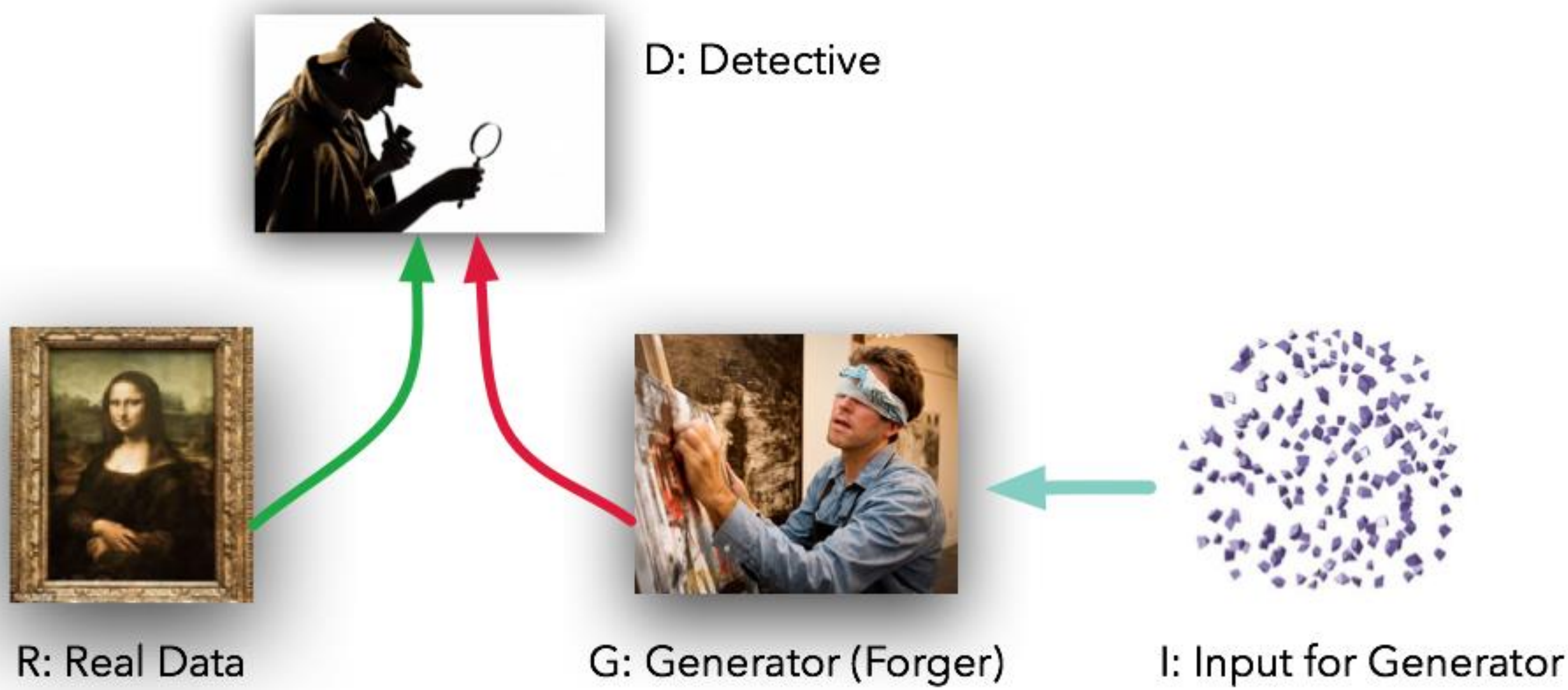
- **DEMO notebooks**

VAE example

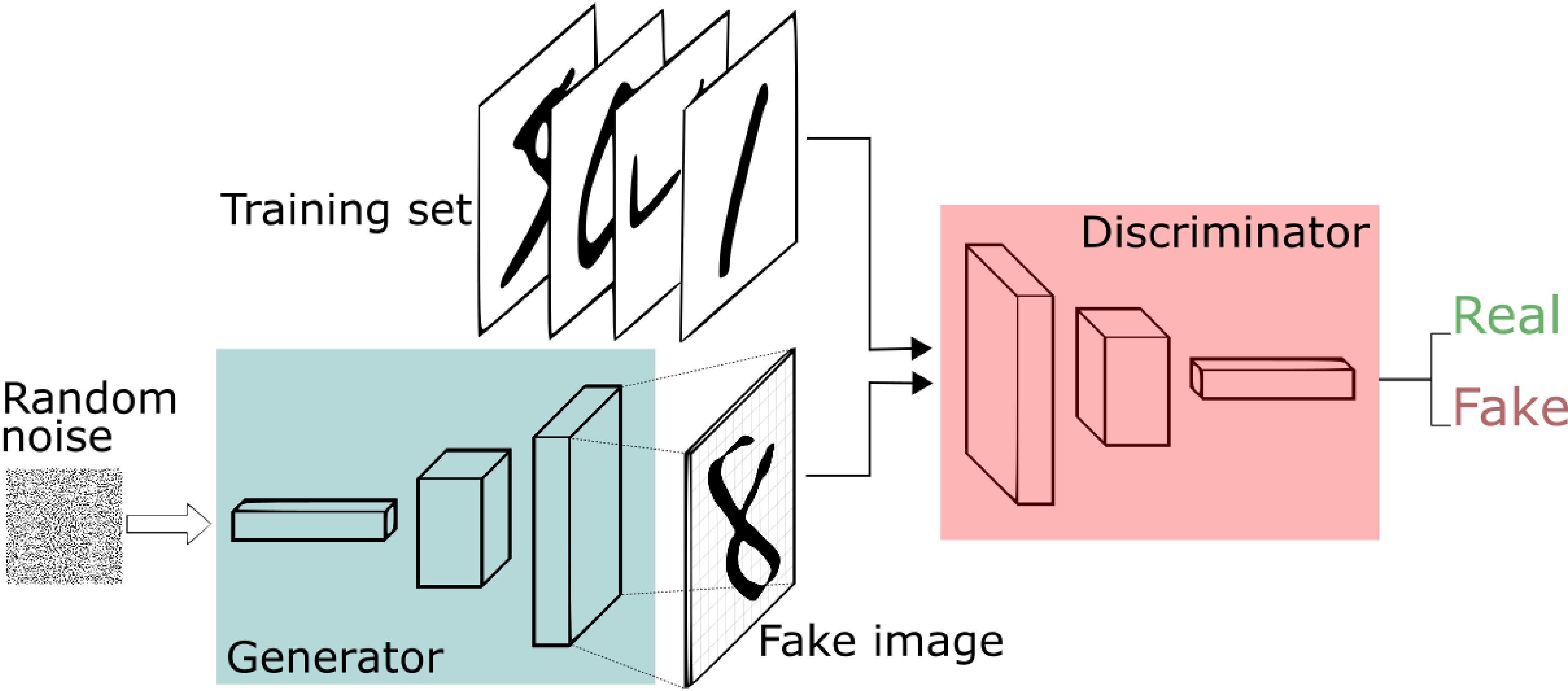


- https://www.youtube.com/watch?v=Q1XuXwPVFko
- https://magenta.tensorflow.org/music-vae

- Imagine an art forger who want's to make Mona Lisa to sell it
- But the gallery has an art "detective": game



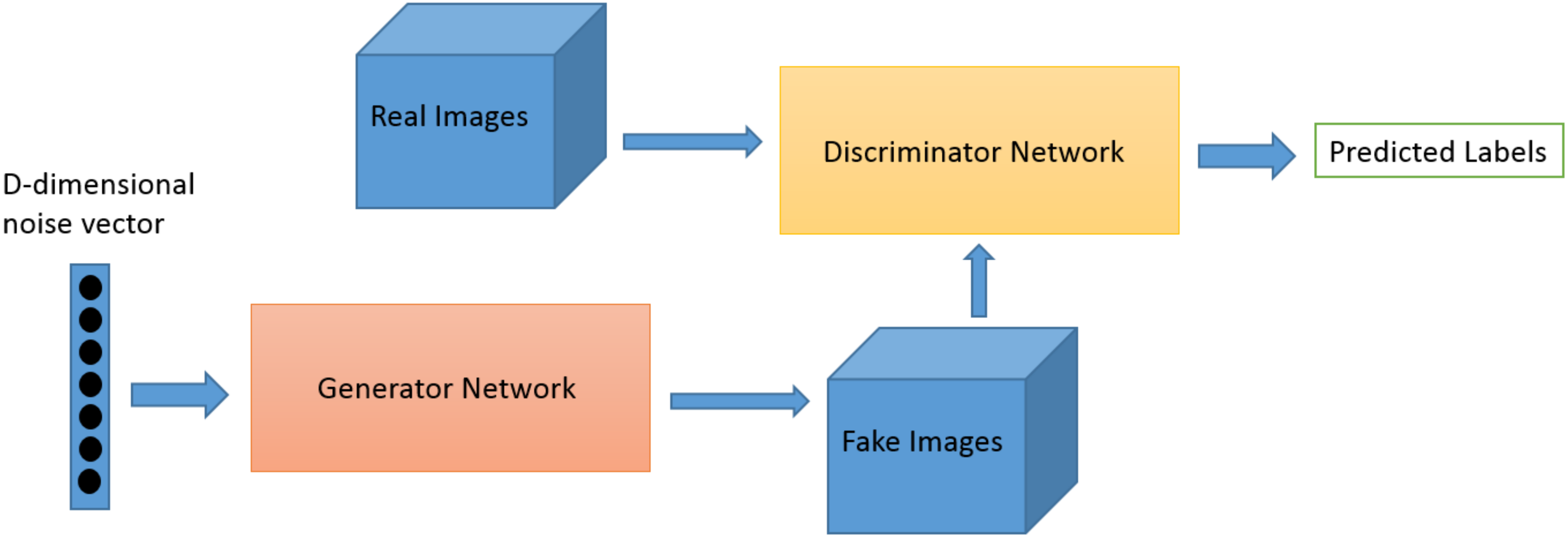
- GAN: two competing network
- Generator: tries to create real-like pictures
- Discriminator: wants to detect the forgery



- The cost used for the discriminator:

$$J^{(D)}\left(\Theta^{(D)},\Theta^{(G)}\right) = -\frac{1}{2}\mathbb{E}_{x\sim p(X)}\log D(x) - \frac{1}{2}\mathbb{E}_{x\sim p(X)}\log\left(1 - D\left(G(x)\right)\right)$$

- This is the binary-crossentropy (real=1, fake=0)



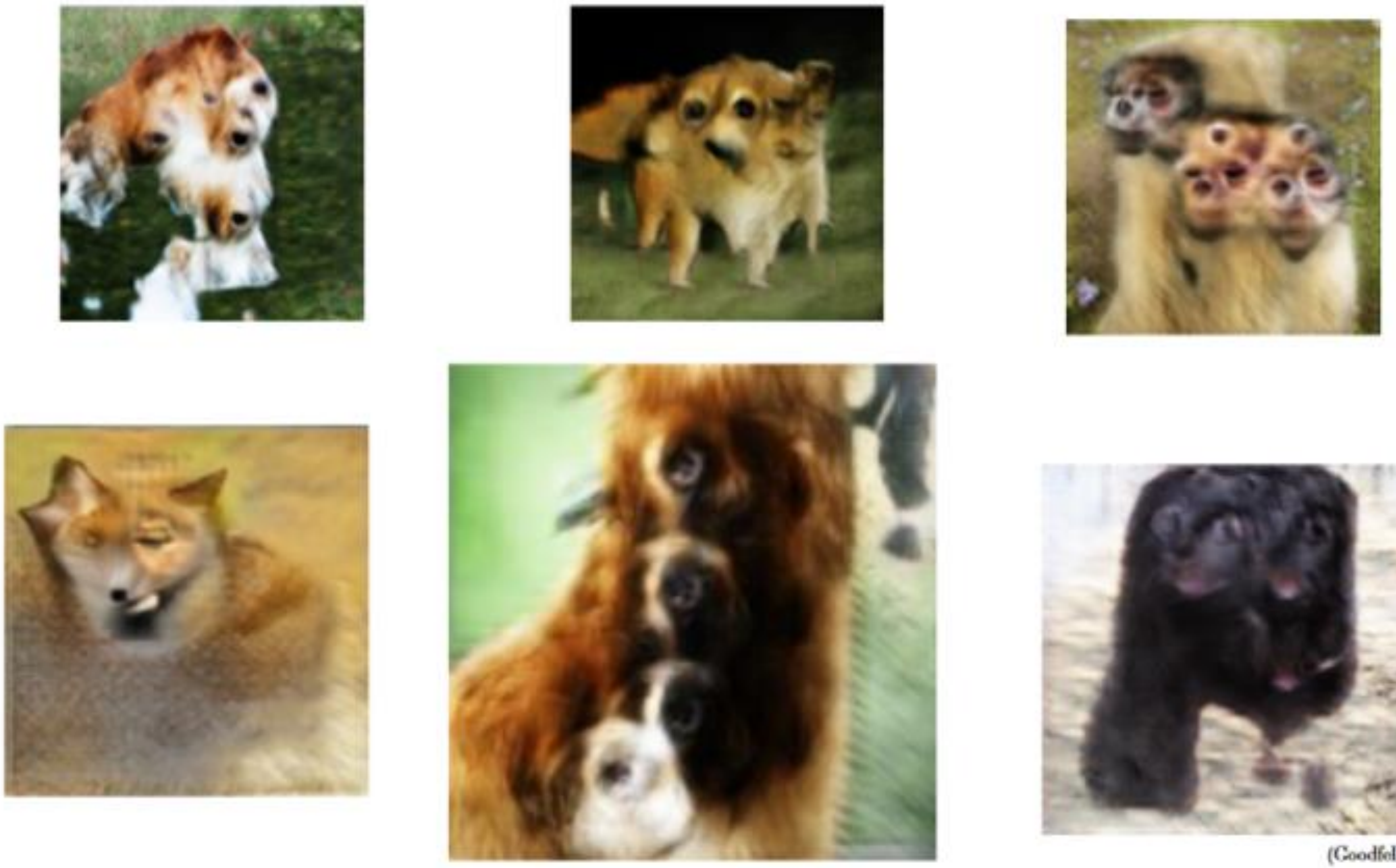
- Discriminator tries to distinguish between real and fake data
- Generator tries to fool the discriminator
- What loss can be used for this?
- Simplest case:

$$J^{(G)}\left(\Theta^{(D)}, \Theta^{(G)}\right) = -J^{(G)}\left(\Theta^{(D)}, \Theta^{(G)}\right) =$$
$$\frac{1}{2}\mathbb{E}_{x\sim p(X)} \log D(x) + \frac{1}{2}\mathbb{E}_{x\sim p(X)} \log\left(1 - D\big(G(x)\big)\right)$$

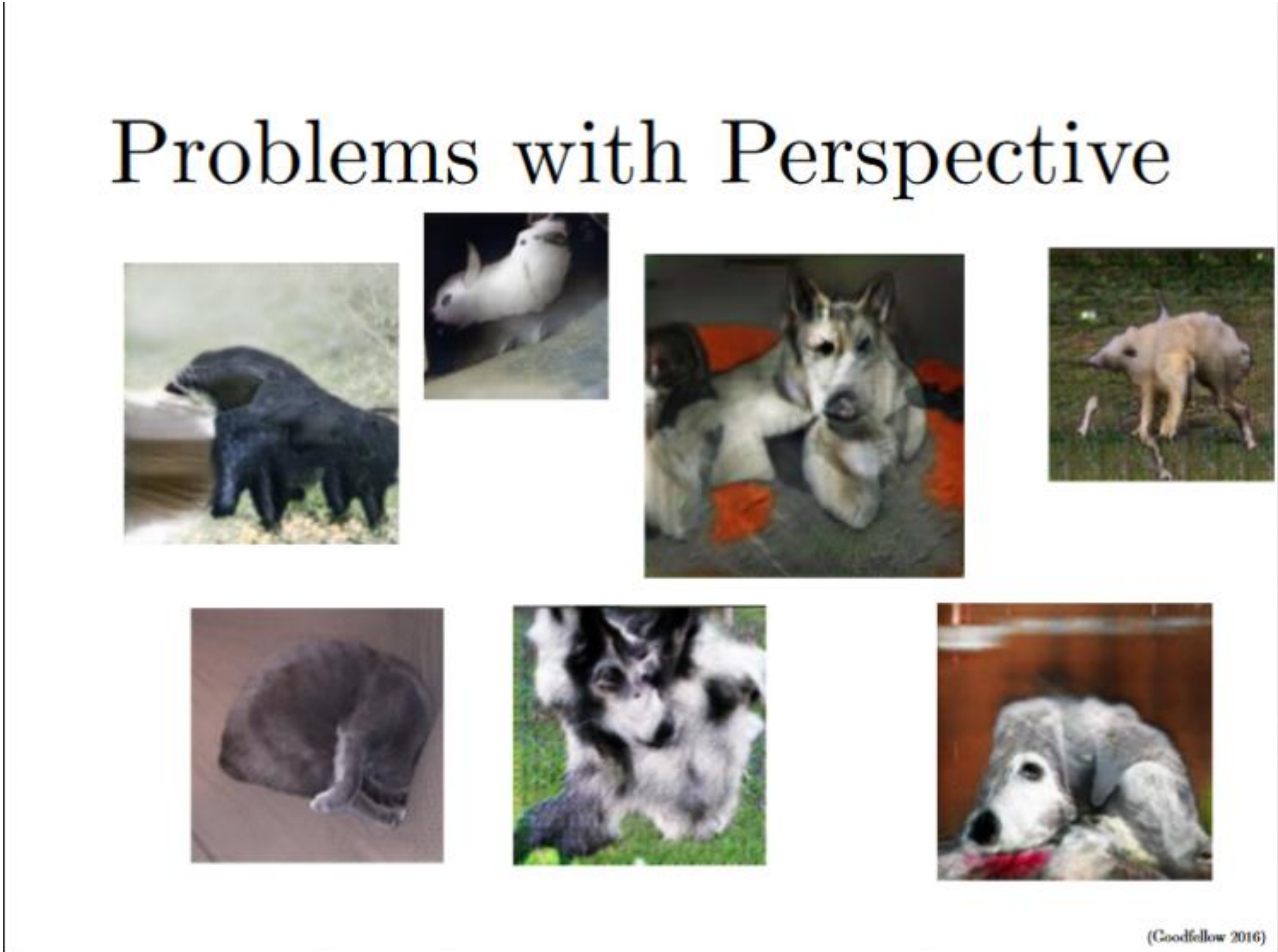
- So the discriminator tries to make $D\big(G(x)\big)$ close to 0
- The generator tries to make $D\big(G(x)\big)$ close to 1 (log big negative number)
- Minimax game: $\min_G \max_D V(D, G)$

$$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

Problems with Counting

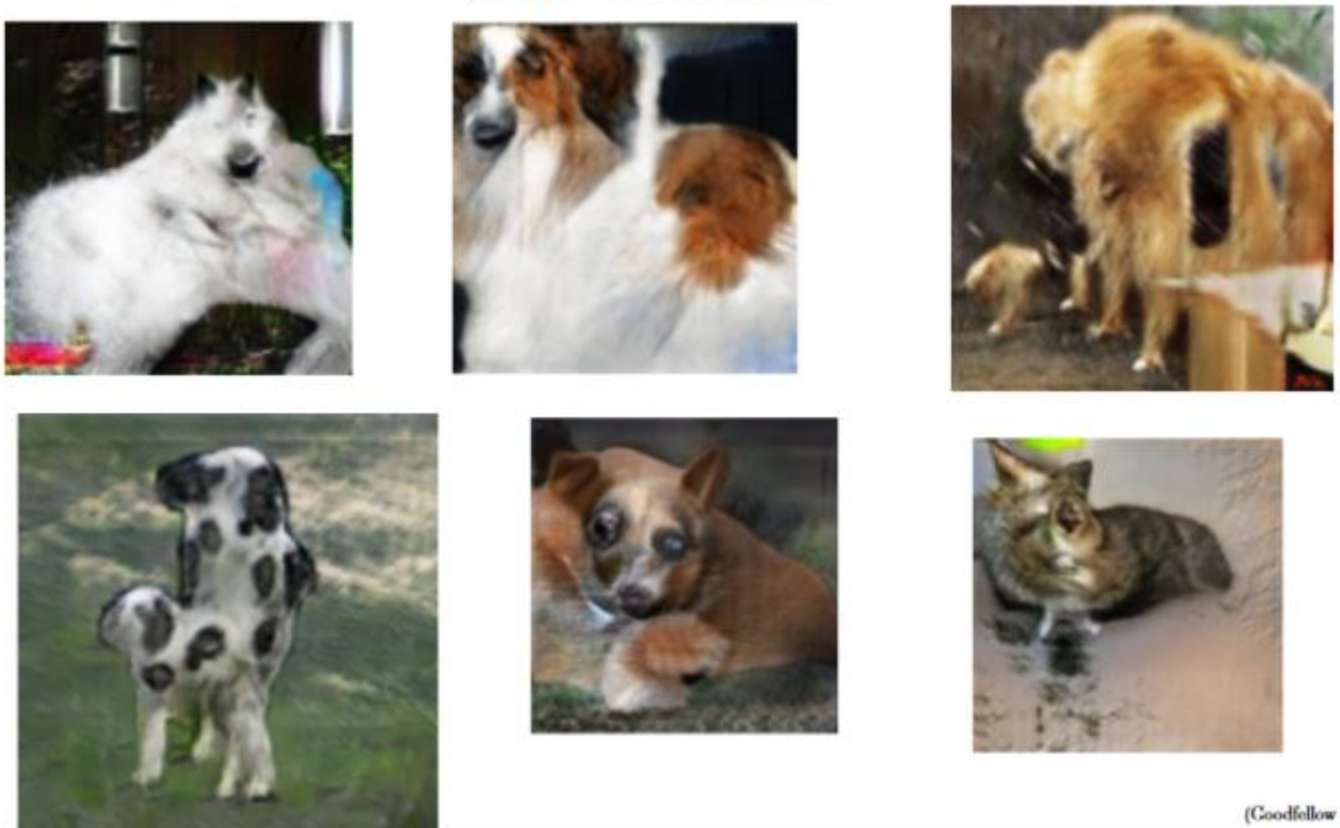


(Goodfellow 2016)


Problems with Perspective
(Goodfellow 2016)

Problems with GANs

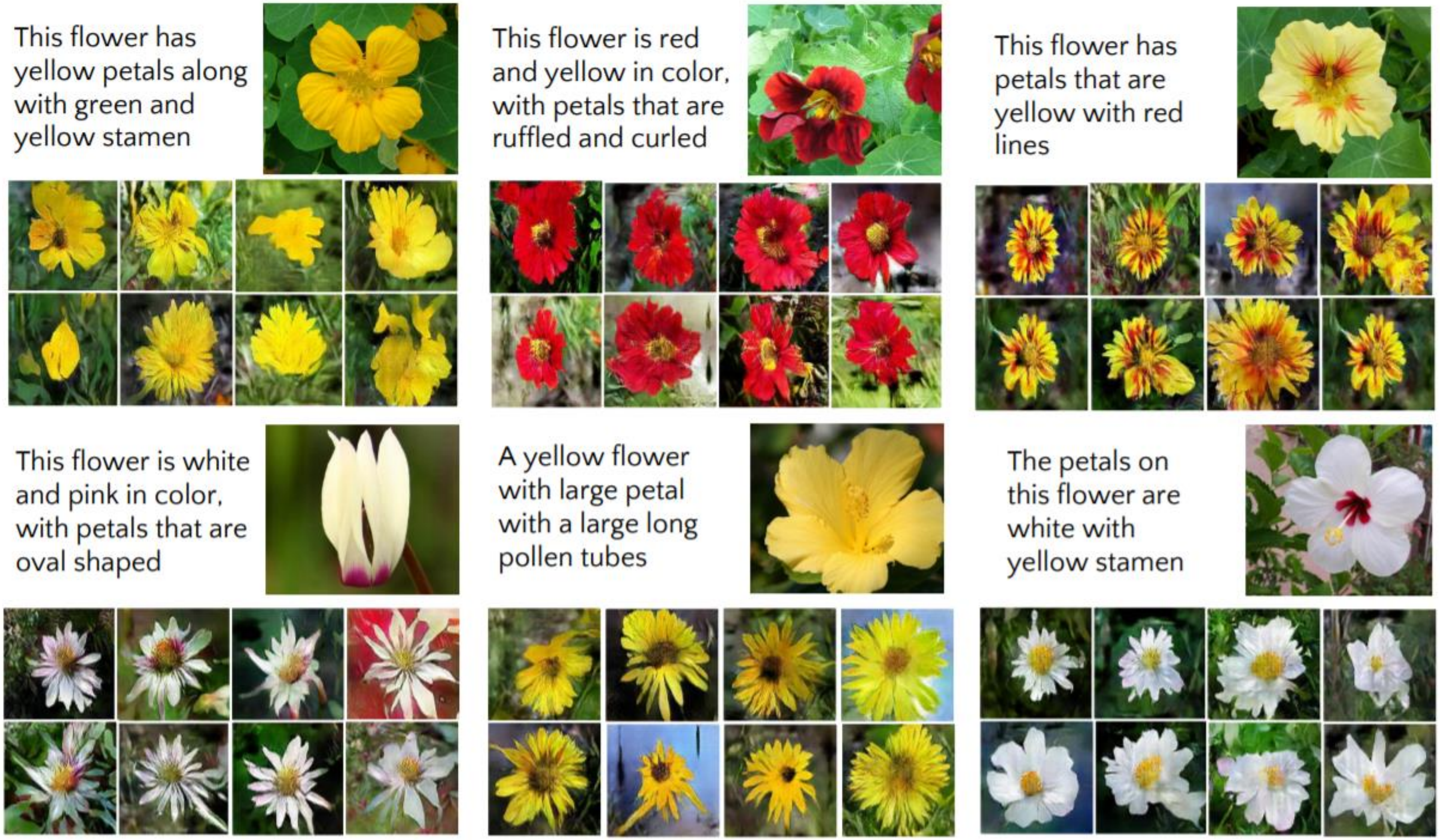
Problems with Global Structure



(Goodfellow 2016)

Examples

- Text to image: https://arxiv.org/pdf/1703.06412.pdf



Examples

- Houses: https://www.youtube.com/watch?v=JCEuwO5BPnk&t=97s
- Zebras:

https://www.youtube.com/watch?v=9reHvktowLY

- Faces:

https://www.youtube.com/watch?v=G06dEcZ-QTg

- And a lot of other amazing examples

- Hacks: https://github.com/soumith/ganhacks

- **DEMO notebooks**

Homeworks

Értékelés:

Házik:

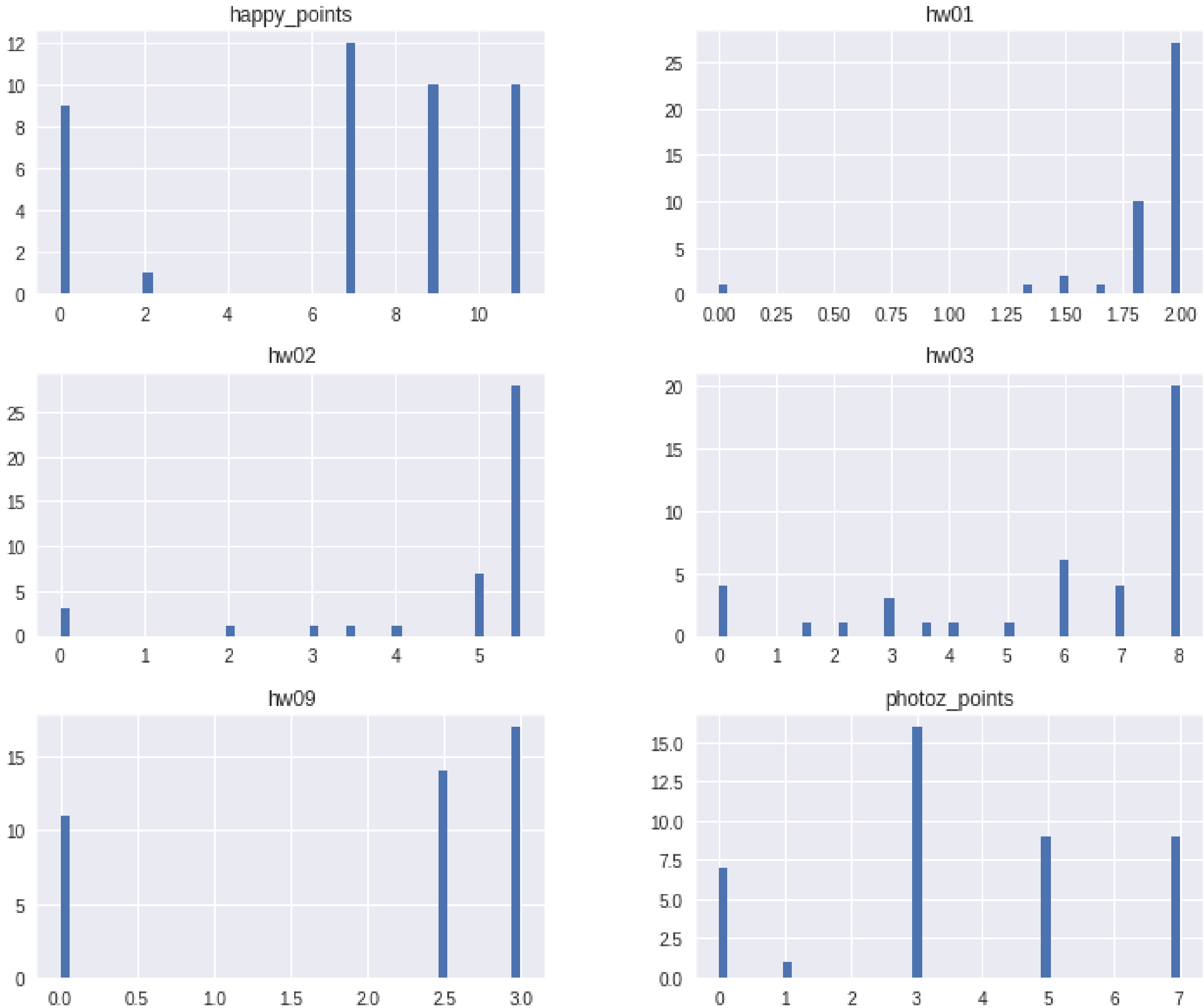
- hw01 2 pont
- hw02 5.5 pont
- hw03 8 pont
- hw09 3 pont

Photoz kaggle:

- 1-10: 7 pont
- 11-20: 5 pont
- 21-baseline: 3 pont
- baseline alatt: 1 pont

Photoz kaggle:

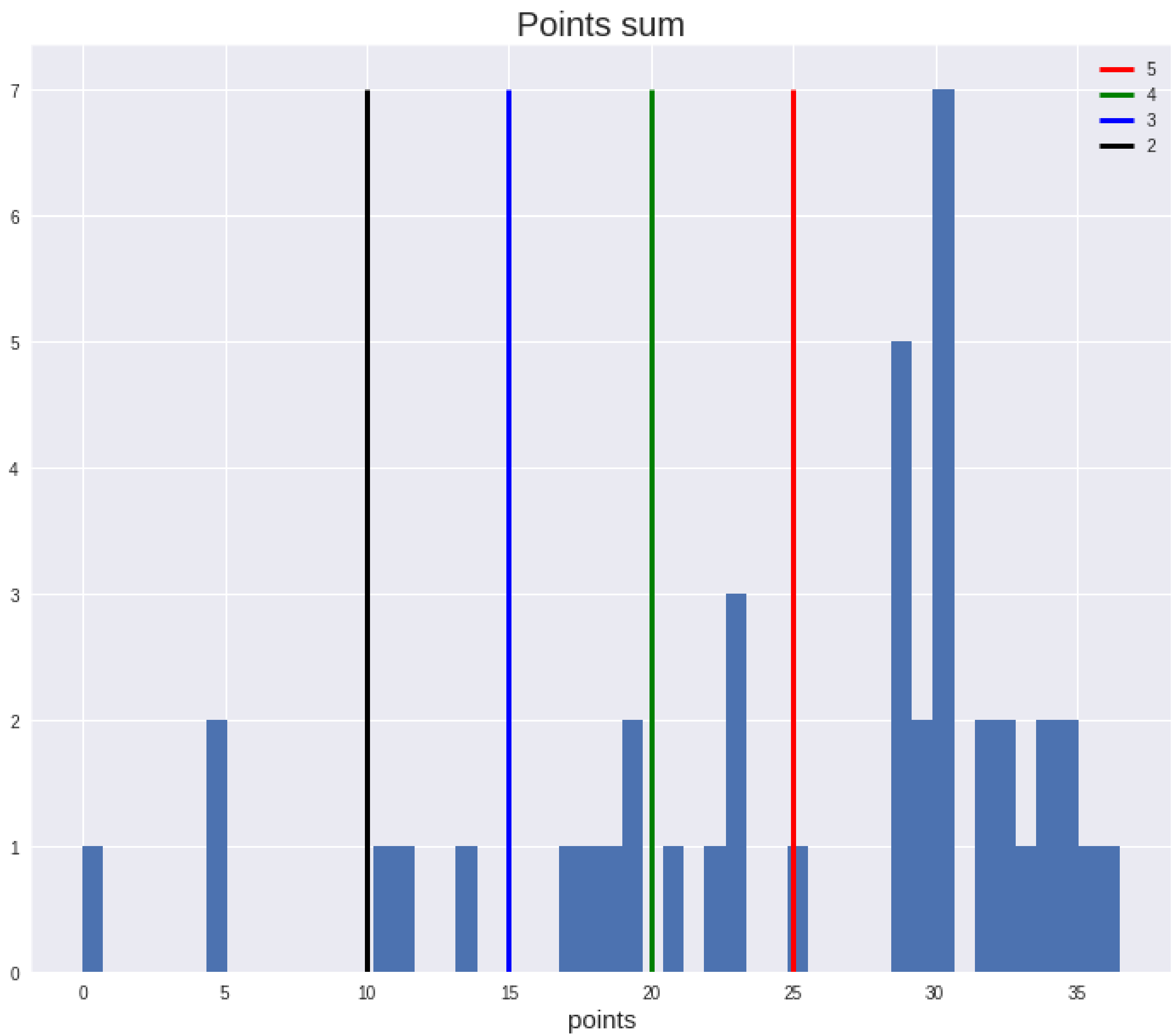
- 1-10: 11 pont
- 11-20: 9 pont
- 21-baseline: 7 pont
- baseline alatt: 2 pont



Max pont 36.5

Max házi + (kaggle baseline + ϵ) = 28.5

Homeworks



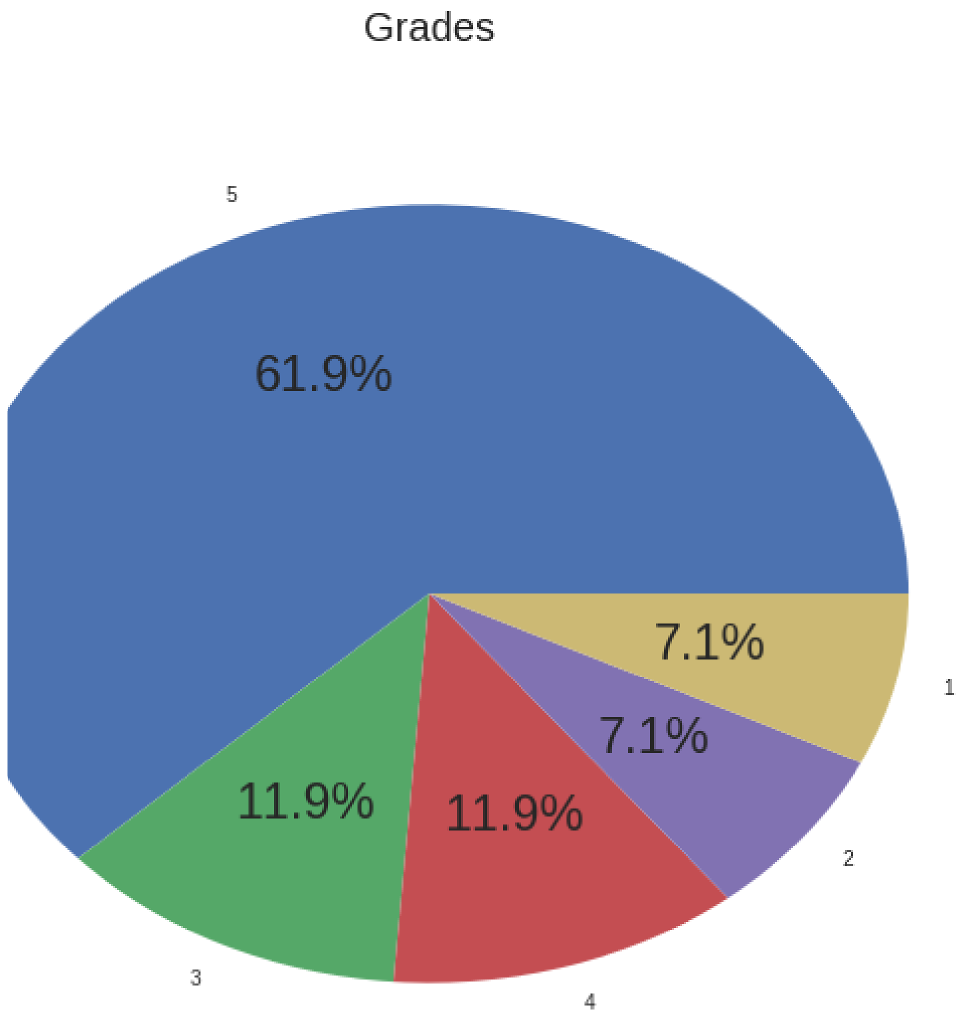
```
Counter({1: 3, 2: 3, 3: 5, 4: 5, 5: 26})
```

	githubName	hw01	hw02	hw03	hw09	happy_points	photoz_points	points_sum	grades
16	danielgrajzel	1.835	0.0	0.0	0.0	0.0	3.0	4.835	1
19	gichy	2.000	3.0	0.0	0.0	0.0	0.0	5.000	1
45	Zongi93	0.000	0.0	0.0	0.0	0.0	0.0	0.000	1

	githubName	hw01	hw02	hw03	hw09	happy_points	photoz_points	points_sum	grades
10	balint225	1.835	5.5	3.000	0.0	0.0	0.0	10.335	2
17	e-velin	2.000	0.0	0.000	0.0	9.0	0.0	11.000	2
18	ggalgoczi	2.000	2.0	2.165	0.0	0.0	7.0	13.165	2

	githubName	hw01	hw02	hw03	hw09	happy_points	photoz_points	points_sum	grades
1	CliffyH	2.000	5.5	6.000	3.0	0.0	3.0	19.500	3
24	ilxstatus	2.000	5.5	4.000	2.5	2.0	3.0	19.000	3
26	kazozoka	2.000	5.0	6.000	0.0	0.0	5.0	18.000	3
27	kissmate6	1.835	5.0	3.000	2.5	0.0	5.0	17.335	3
37	oresme	2.000	3.5	3.665	2.5	7.0	0.0	18.665	3

	githubName	hw01	hw02	hw03	hw09	happy_points	photoz_points	points_sum	grades
4	PentadD	2.000	5.5	3.0	2.5	7.0	3.0	23.000	4
6	Turcsi	1.835	5.0	6.0	0.0	7.0	3.0	22.835	4
28	kommancs96	1.835	5.0	8.0	3.0	0.0	5.0	22.835	4
33	masterdesky	1.500	5.0	6.0	0.0	7.0	1.0	20.500	4
41	zentaijanos	2.000	4.0	1.5	3.0	9.0	3.0	22.500	4



Homework for extra points

- https://github.com/qati/DeepLearningCourse/tree/master/assignments/emoji

References

- https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798
- https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d
- David, Eli & Netanyahu, Nathan. (2016). DeepPainter: Painter Classification Using Deep Convolutional Autoencoders. 20-28. 10.1007/978-3-319-44781-0_3.
- Stanford CS231n, Lecture 11: https://www.youtube.com/watch?v=nDPWywWRIRo
- https://towardsdatascience.com/autoencoders-introduction-and-implementation-3f40483b0a85
- https://blog.manash.me/implementing-pca-feedforward-and-convolutional-autoencoders-and-using-it-for-image-reconstruction-8ee44198ea55
- http://www.ericlwilkinson.com/blog/2014/11/19/deep-learning-sparse-autoencoders
- http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/
- https://towardsdatascience.com/autoencoders-are-essential-in-deep-neural-nets-f0365b2d1d7c
- https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf
- https://www.jeremyjordan.me/variational-autoencoders/
- https://jaan.io/what-is-variational-autoencoder-vae-tutorial/
- http://kvfrans.com/variational-autoencoders-explained/
- https://github.com/houxianxu/DFC-VAE
- https://deeplearning4j.org/generative-adversarial-network
- https://www.analyticsvidhya.com/blog/2017/06/introductory-generative-adversarial-networks-gans/
- https://medium.com/@devnag/generative-adversarial-networks-gans-in-50-lines-of-code-pytorch-e81b79659e3f
- https://arxiv.org/pdf/1701.00160.pdf
- https://github.com/hjweide/adversarial-autoencoder
- Deep Learning with Python, by FRANÇOIS CHOLLET