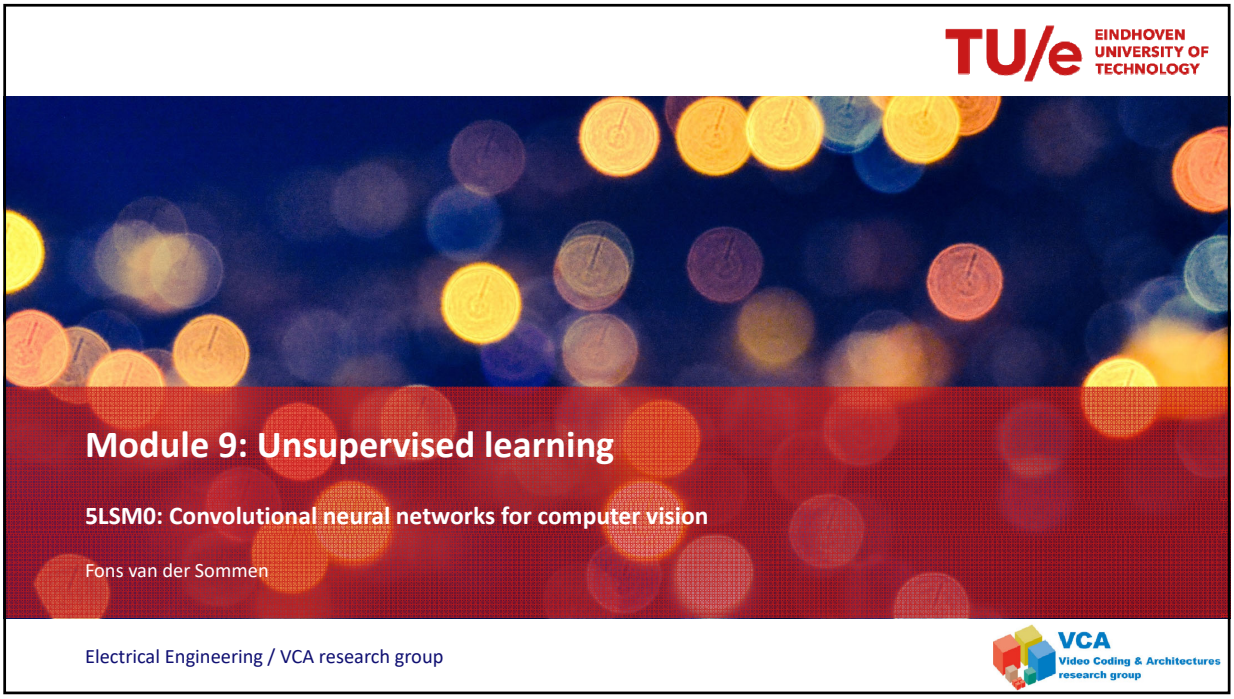


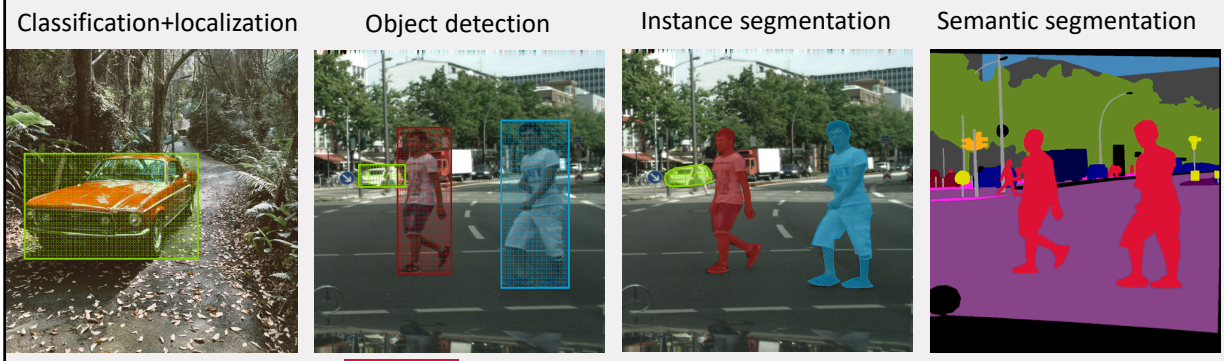
Module 9: Unsupervised learning

5LSM0: Convolutional neural networks for computer vision

Fons van der Sommen

Electrical Engineering / VCA research group

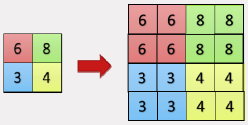
Last time



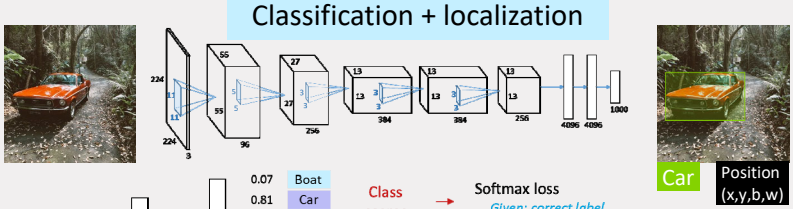
Single object | Multiple objects | No objects, just pixels

Last time

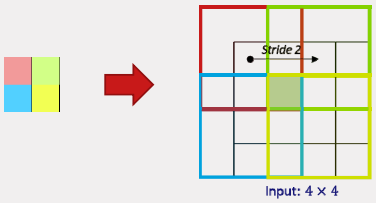
Unpooling



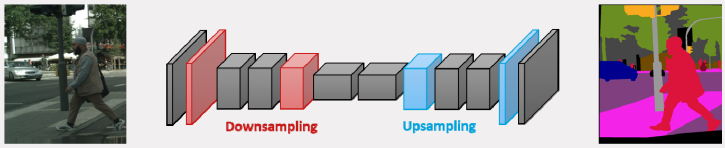
Classification + localization



Transpose convolution

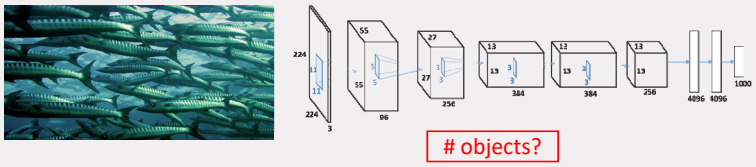


FCN for semantic segmentation

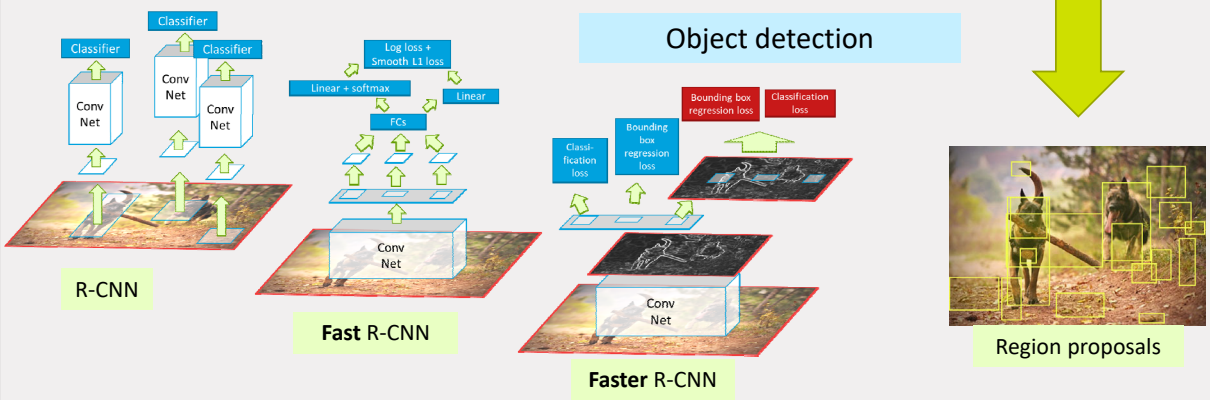


- Pooling
- Strided convolution
- Unpooling
- Strided transpose convolution

Last time

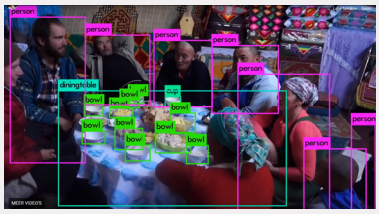


Object detection

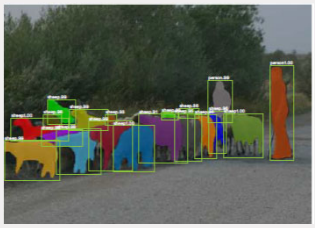


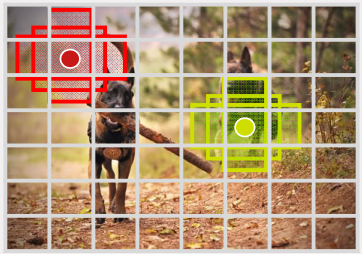
Last time

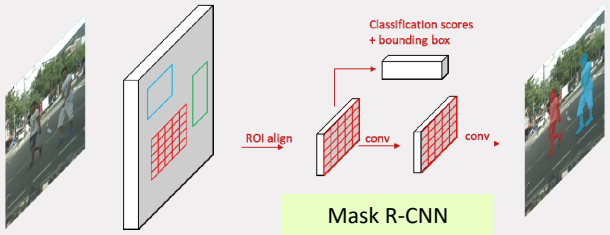
YOLO / SsSD



Instance segmentation







Mask R-CNN

This time

Unsupervised learning

- … with convnets!

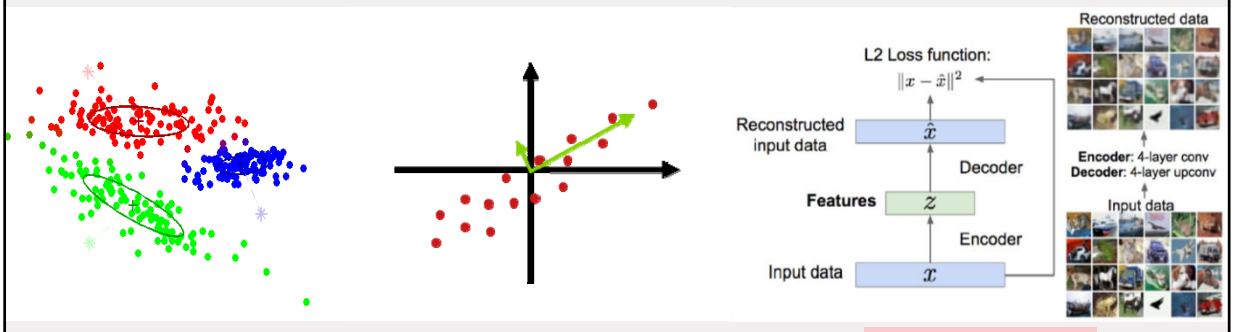
Generative models

- PixelRNN and PixelCNN
- Variational Autoencoders (VAEs)
- Generative Adversarial Networks (GANs)

Supervised vs Unsupervised

Supervised learning	Unsupervised learning
Labeled data Data $X \in \mathbb{R}^{N \times D}$ and targets $t \in \mathbb{R}^{N \times M}$	**Just data, no labels** Data $X \in \mathbb{R}^{N \times D}$
Goal: learn a function to map $X \rightarrow t$	**Goal**: learn some underlying hidden structure of the data
Examples: Classification ($X \in \mathbb{R}, t \in S$) Regression ($X \in \mathbb{R}, t \in \mathbb{R}$) Segmentation ($X \in \mathbb{R}, t \in \mathbf{S}$)	**Examples:** Clustering Dimensionality reduction Density estimation

Unsupervised learning

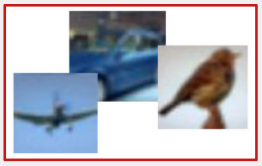


Clustering | Dimensionality reduction | Feature learning

Generative models

Learn the underlying distribution of the (training data)

- So we can generate new samples



Training data $p_{data}(x)$

Generated data $p_{model}(x)$

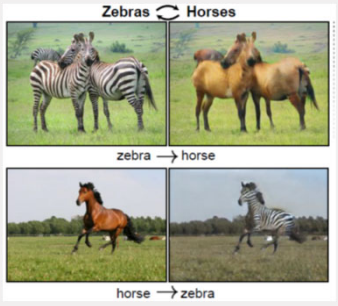
Goal: learn model $p_{model}(x)$ which is close to the underlying distribution $p_{data}(x)$ of the training data.

Form of density estimation

- Explicit density estimation: explicitly define and solve for p_{model}
- Implicit density estimation: learn a model that can sample from p_{model}

Generative models

Why are generative models so cool?



Zhu et al. "*Unpaired image-to-image translation using cycle-consistent adversarial networks"* CVPR 2017.

Yeh et al. "*Semantic image inpainting with deep generative models*." CVPR 2017.

These are **not** actual people

Karras, Tero, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks." *arXiv preprint arXiv:1812.04948* (2018).

PixelRNN

Type of fully visible belief network

- Explicit density model
- Use chain rule to decompose likelihood of an image x into product of 1-d distributions

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

Likelihood of image x

Probability of the i^{th} pixel value given all previous pixel values

What does this even mean?

- Then maximize likelihood of training data

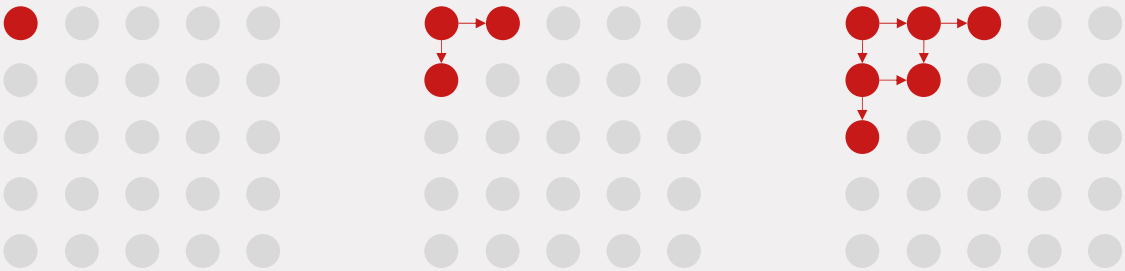
Complex distribution! → Use neural network to model this.

Van der Oord et al., "*Pixel Recurrent Neural Networks*" , ICML 2016F

PixelRNN

Generate image pixels from corner

- Dependency on previous pixels modeled using a Recurrent Neural Network (RNN) [module 11]



Drawback: sequential generation is super slow!

Van der Oord et al., "*Conditional Image Generation with PixelCNN Decoders*", NIPS 2016

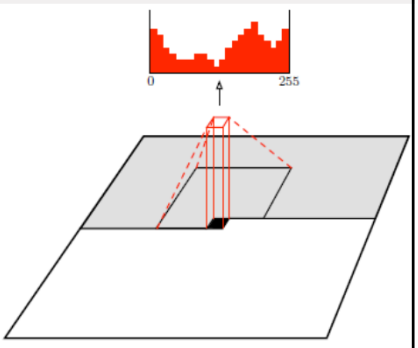
PixelCNN

Generate image pixels from corner

- New: dependency on previous pixels now modeled using a CNN over a context region

- Training: maximize likelihood of training images

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

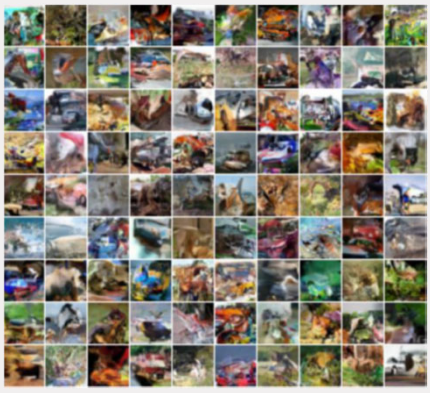


Training is faster than PixelRNN!

Generation is still slow due to sequential procedure

Van der Oord et al., "*Conditional Image Generation with PixelCNN Decoders*", NIPS 2016

PixelCNN



Trained on 32 × 32 pixel CIFAR-10

Trained on 32 × 32 pixel ImageNet

PixelRNN & pixelCNN

Pros:

- Can explicitly compute likelihood (quality) of your generated samples $p(x)$
- Explicit likelihood of training data gives good evaluation metric
- Good samples

Cons:

- Sequential generation → slow..

Can't we generate an image in a single pass?

- Variational Autoencoders!

Variational autoencoders

So far we have defined a tractable density function

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

Variational AutoEncoders (VAEs) define intractable density function

$$p_\theta(x) = \int p(z) p_\theta(x|z) dz$$

We cannot optimize this directly

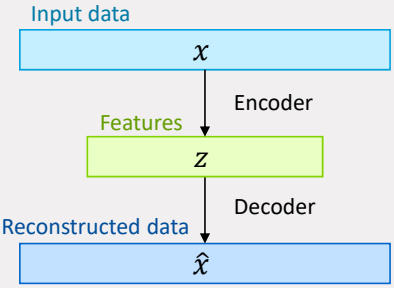
Variational autoencoders

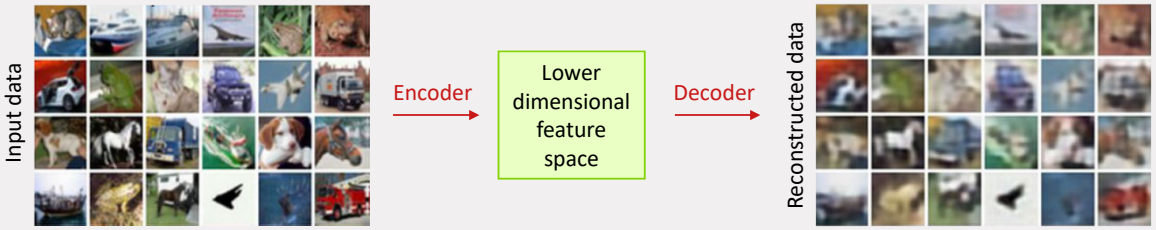
Background: Autoencoders

- Learn a lower-dimensional feature representation

How to learn this transformation?

- Add a decoder to reconstruct the original data





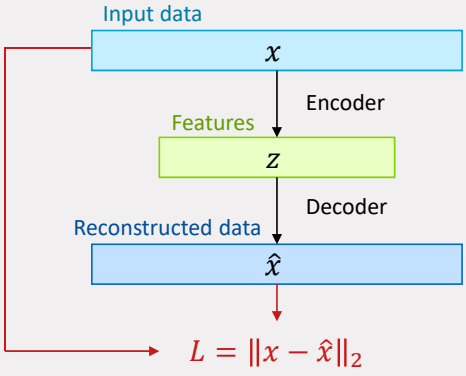
Variational autoencoders

Background: Autoencoders

- Learn a lower-dimensional feature representation

How to learn this transformation?

- Add a decoder to reconstruct the original data
- The mismatch between x and $\hat{x}$ defines the loss



Variational autoencoders

Background: Autoencoders

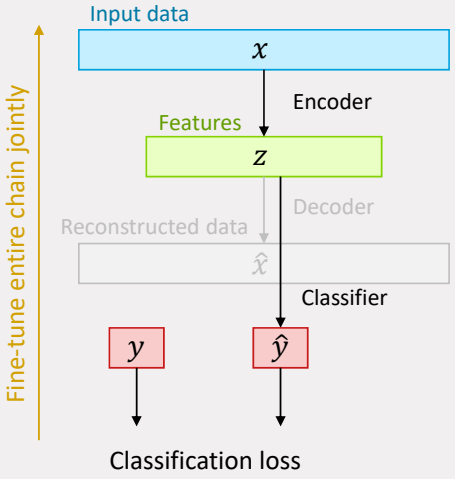
- Learn a lower-dimensional feature representation

How to learn this transformation?

- Add a decoder to reconstruct the original data
- The mismatch between x and $\hat{x}$ defines the loss

Use the learned feature representation

- After training, discard the decoder
- Fine-tune for supervised learning task



We can use a large set of unlabeled data to learn a good initialization for a supervised model!

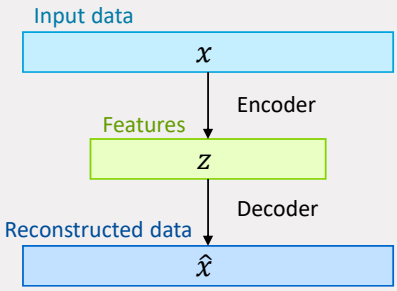
Variational autoencoders

Autoencoder summary

- Autoencoders can reconstruct data and can learn features to initialize a supervised model
- Features capture factors of variation in training data.

Can we generate new images from an autoencoder?

- Variational autoencoders!



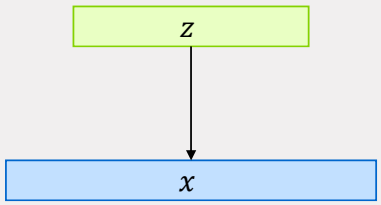
Variational autoencoders

Probablistic spin on autoencoders

- Let us sample from the model to generate data!
- Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from underlying latent representation z

Sample from true conditional $p_{\theta^*}(z)$

Sample from true conditional $p_{\theta^*}(x|z^{(i)})$



Intuitive interpretation

- z contains information on how much each of a set of latent attributes is present in x.
- Example: if our training data contained images of faces, elements of z could represent the size of the ears, the position & color of the eyes, etc..

Variational autoencoders

→ Estimate the true parameters θ^* of this generative model

How should we represent this model?

- Choose prior $p(z)$ simple: Gaussian
- Conditional $p(x|z)$ is complex → Neural network
  - *This generates the image from the latent representation*

How to train this model?

- Earlier we saw: maximize likelihood of training data:

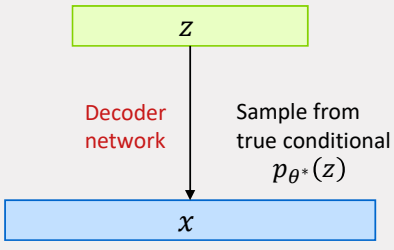
$$p_\theta(x) = \int p(z)p_\theta(x|z)dz$$

Q: Problem?

Sample from true conditional $p_{\theta^*}(x|z^{(i)})$

Decoder network

Sample from true conditional $p_{\theta^*}(z)$



Variational autoencoders

Let's take a closer look

- Data likelihood:

Intractible to compute for every z

$$p_\theta(x) = \int p(z) p_\theta(x|z) dz$$

Simple Gaussian prior — Output of decoder network

- Posterior density:

$$p_\theta(z|x) = \frac{p_\theta(x|z) p_\theta(z)}{p_\theta(x)}$$

Intractible, so cannot compute the posterior → cannot optimize model

Solution:

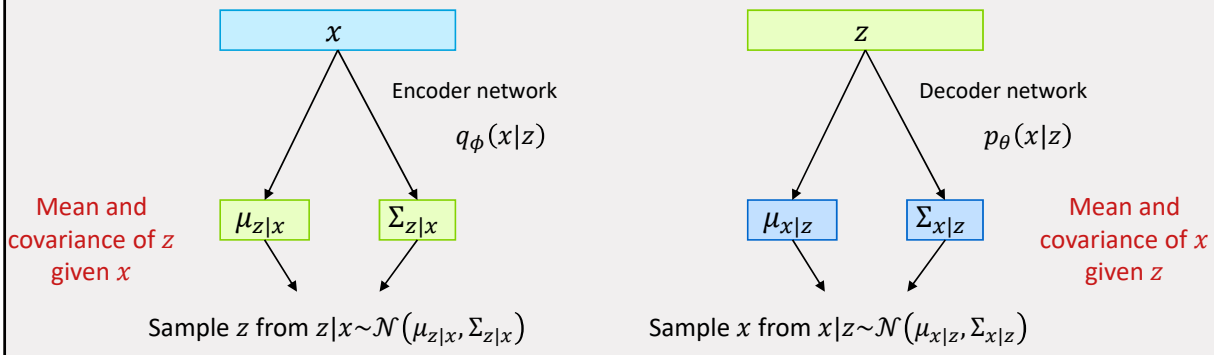
- Define an additional encoder network $q_\phi(z|x)$ to approximate $p_\theta(z|x)$

Variational autoencoders

Probablistic generation of data

Kingma and Welling, "*Auto-Encoding Variational Bayes*", ICLR 2014

- Encoder and decoder networks are probablistic



Variational autoencoders

$p_\theta(x^{(i)})$ independent of z

Bayes' rule

Multiply by constant

Rewrite using log identities

$$\log p_\theta(x^{(i)}) = \mathbf{E}_z[\log p_\theta(x^{(i)})] = \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})}\right]$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})} \cdot \frac{q_\phi(z|x^{(i)})}{q_\phi(z|x^{(i)})}\right]$$

$$= \mathbf{E}_z[\log p_\theta(x^{(i)}|z)] - \mathbf{E}_z\left[\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z|x^{(i)})}\right]$$

$$= \mathbf{E}_z[\log p_\theta(x^{(i)}|z)] - D_{KL}\left(q_\phi(z|x^{(i)})||p_\theta(z)\right) + D_{KL}\left(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)})\right)$$

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling

This KL term (between Guassians for encoder and z prior) has nice closed form solution

$p_\theta(z|x^{(i)})$ intractable, can't compute this KL term. Bet we know divergence always ≥ 0

Variational autoencoders

Variational lower bound

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)$$

Training: minimize lower bound

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \mathcal{L}(x^{(i)}, \theta, \phi)$$

$$\log p_\theta(x^{(i)}) = \mathbf{E}_z[\log p_\theta(x^{(i)})] = \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})}\right]$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)}|z)p_\theta(z)}{p_\theta(z|x^{(i)})} \cdot \frac{q_\phi(z|x^{(i)})}{q_\phi(z|x^{(i)})}\right]$$

$$= \mathbf{E}_z[\log p_\theta(x^{(i)}|z)] - \mathbf{E}_z\left[\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z|x^{(i)})}\right]$$

Force features to follow our prior

Force features to reconstruct the input data

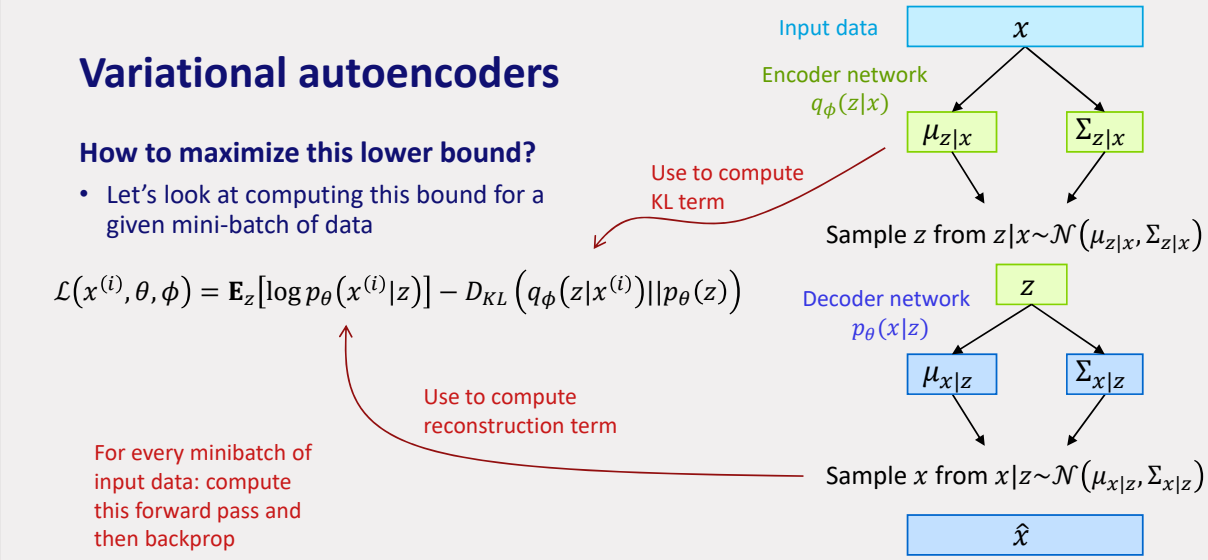
$$= \underbrace{\mathbf{E}_z[\log p_\theta(x^{(i)}|z)] - D_{KL}\left(q_\phi(z|x^{(i)})||p_\theta(z)\right)}_{\text{Tractable lower bound } \mathcal{L}(x^{(i)}, \theta, \phi) \text{ which we can take the gradient of and optimize}} + \underbrace{D_{KL}\left(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)})\right)}_{\text{always} \geq 0}$$

Variational autoencoders

How to maximize this lower bound?

- Let's look at computing this bound for a given mini-batch of data

$$\mathcal{L}(x^{(i)}, \theta, \phi) = \mathbf{E}_z[\log p_\theta(x^{(i)}|z)] - D_{KL}\left(q_\phi(z|x^{(i)})||p_\theta(z)\right)$$



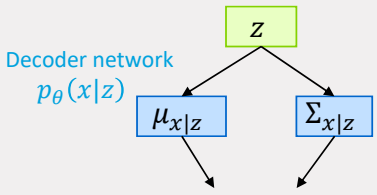
For every minibatch of input data: compute this forward pass and then backprop

Variational autoencoders

Generating data

- Sample from prior in feature space and decode!

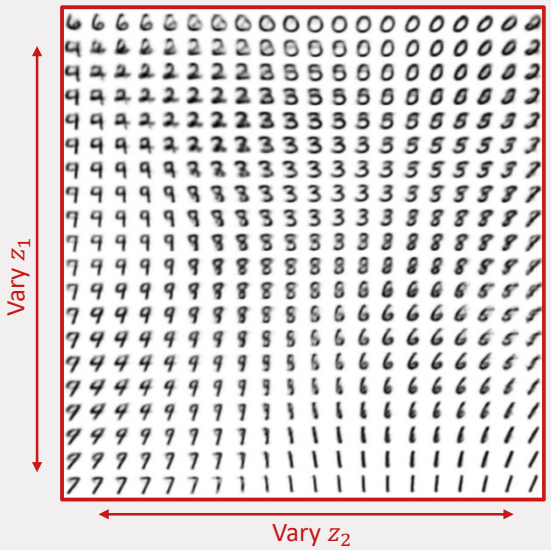
Sample z from $z|x \sim \mathcal{N}(0, I)$



Sample x from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$

$\hat{x}$

Example: data manifold for $z \in \mathbb{R}^2$



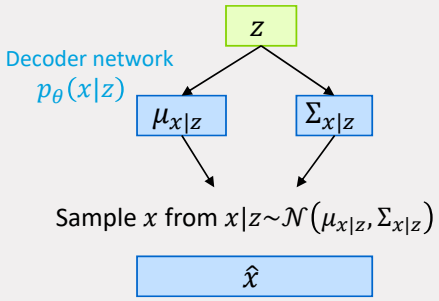
Variational autoencoders

Example: Faces

Generating data

- Sample from prior in feature space and decode!

Sample z from $z|x \sim \mathcal{N}(0, I)$



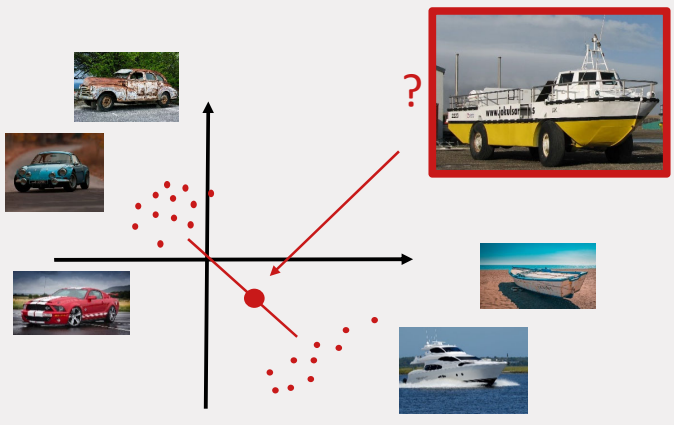
Q1: Benefit of enforcing diagonal covariance?

Q2: What if we would have sampled from the feature space of the vanilla AE?

Variational autoencoders

Intuitive explanation VAEs

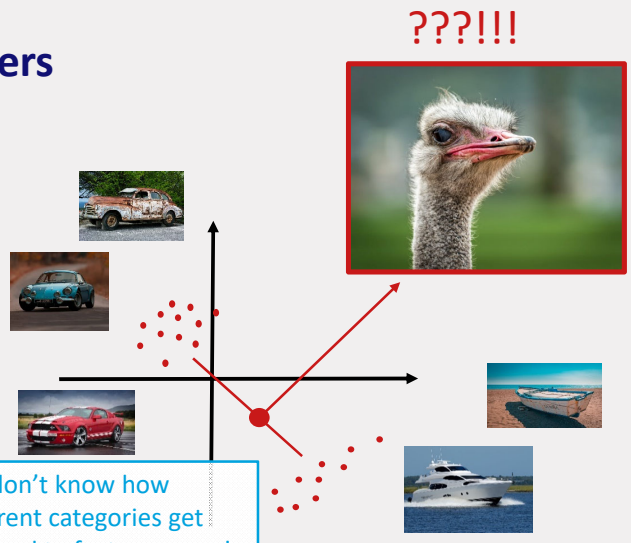
- What if we would like to interpolate between two points in feature space when using a vanilla autoencoder
- We have computed a bunch of feature representations, so we know where in feature space certain objects are
- Is this interpolation straightforward when using normal autoencoders?



Variational autoencoders

Intuitive explanation VAEs

- What if we would like to interpolate between two points in feature space when using a vanilla autoencoder
- We have computed a bunch of feature representations, so we know where in feature space certain objects are
- Is this interpolation straightforward when using normal autoencoders?



Variational autoencoders

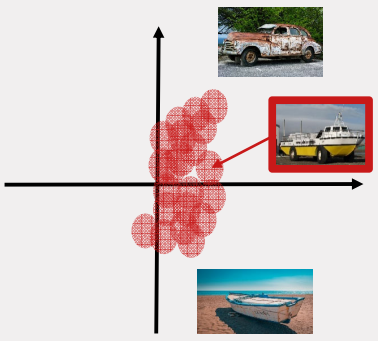
VAEs add two crucial components to fix this

1. A data sample x is not mapped to a single point in feature space, rather it is smeared out over a small region by means of a probability distribution
   - *This means that when decoding, there should be overlap between objects that look similar and, hence, forces them to "stick" together*

$$\mathcal{L}(x^{(i)}, \theta, \phi) = \mathbf{E}_z[\log p_\theta(x^{(i)}|z)] - D_{KL}\left(q_\phi(z|x^{(i)})||p_\theta(z)\right)$$

2. A cost is introduced for distributions that deviate from a zero-mean, unity std dev Gaussian.
   - *This keeps the feature space compact*

The region responsible for our favorite boatcar connects to both car and boat samples in feature space!



So far…

Pixel RNN/CNNs

- PixelCNNs define tractable density function, optimize likelihood of training data:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \dots, x_{i-1})$$

Variational Autoencoders

- VAEs define intractiable density function with latent z:

$$p_\theta(x) = \int p(z) p_\theta(x|z) dz$$

- We cannot optimize this directly, instead we derive a lower bound on the likelihood to maximize

What if we give up on explicitly modeling density altogether?

- We just want the ability to sample

TU/e

Generative Adversarial Networks

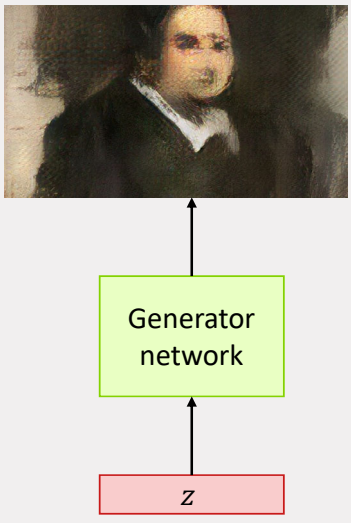
Generative Adversarial Networks (GANs)

- Don't work with an explicit density function!
- Instead, learn to generate from training distribution through a 2-player game…

Problem: Want to sample from complex, high-dimensional training distribution

Solution: Sample from a simple distribution, e.g. white noise, and learn to transformation from simple distribution to training distribution

Model this transformation with a CNN!

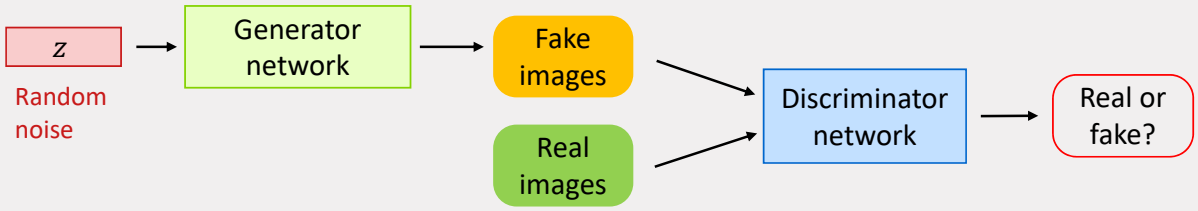


TU/e

Generative Adversarial Networks

It's a two player game

- Generator network: try to fool the discriminator by generating real-looking images
- Discriminator network: try to distinguish between real and fake images



Generative Adversarial Networks

It's a two player game

- Generator network: try to fool the discriminator by generating real-looking images
- Discriminator network: try to distinguish between real and fake images

Train jointly in minimax game

- Objective function:

Discriminator outputs likelihood of data sample being real [0, 1]

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d}\left(G_{\theta_g}(z) \right) \right) \right]$$

Generator wants to minimize | Discriminator wants to maximize | Expected discriminator output for real data | Expected discriminator output for fake data

Generative Adversarial Networks

Q: What does this $D(G(z)) \to 0$ imply?

Minimax objective function

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d}\left(G_{\theta_g}(z) \right) \right) \right]$$

Alternate between

- Gradient ascent on discriminator

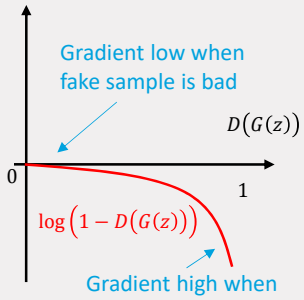
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d}\left(G_{\theta_g}(z) \right) \right) \right]$$

- Gradient descent on generator

$$\min_{\theta_g} \left[\mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d}\left(G_{\theta_g}(z) \right) \right) \right]$$

In practice, this doesn't work very well…

Generator learns slow when it's doing a bad job…



Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generative Adversarial Networks

Note: training GANs is very challenging. Better objective functions for this is an active area of research.

Minimax objective function

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d}\left(G_{\theta_g}(z) \right) \right) \right]$$

Alternate between

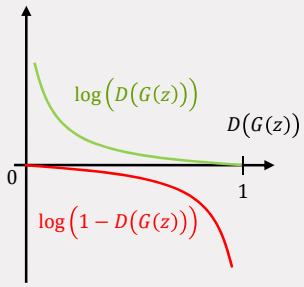
- Gradient ascent on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d}\left(G_{\theta_g}(z) \right) \right) \right]$$

- Instead: Gradient ascent on generator

$$\max_{\theta_g} \left[\mathbb{E}_{z \sim p(z)} \log \left(D_{\theta_d}\left(G_{\theta_g}(z) \right) \right) \right]$$

Much better!
Gradient large when generated sample is likely fake → better learning



Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generative Adversarial Networks

Training GANs: putting it all together

Train discriminator

Train generator

for number of training iterations **do**

  for k steps **do**

    → Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$

    → Sample minibatch of m examples $\{x^{(1)}, \dots, x^{(m)}\}$ of real data

    → Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \log D_{\theta_d}(x) + \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)$$

  end for

  → Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$

  → Update the generator by ascending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)$$

end for

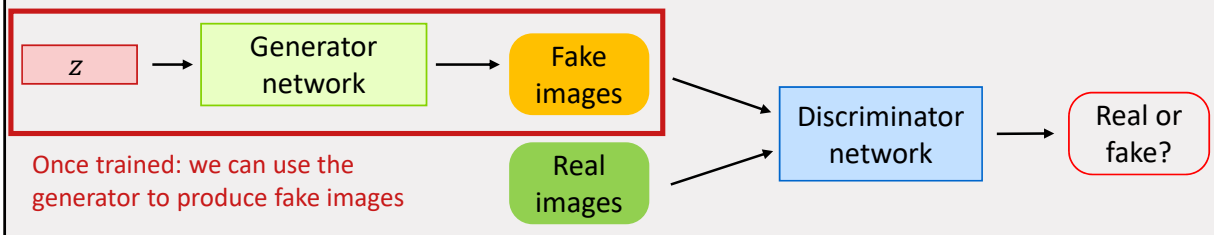
Discriminate between fake and real data!

Improve generated fakes to fool discriminator!

Generative Adversarial Networks

It's a two player game

- Generator network: try to fool the discriminator by generating real-looking images
- Discriminator network: try to distinguish between real and fake images



Once trained: we can use the generator to produce fake images

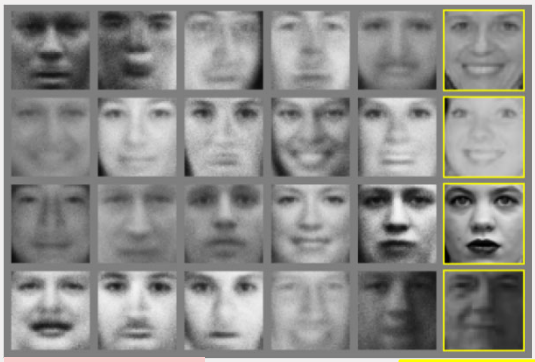
Generative Adversarial Networks

GANs: first results (2014)

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generated digits

Nearest neighbors



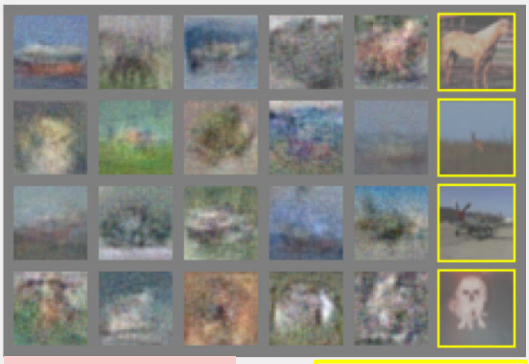
Generated faces

Nearest neighbors

Generative Adversarial Networks

GANs: first results (2014)

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014



Results on CIFAR-10

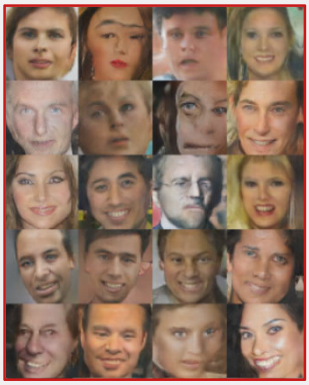
Nearest neighbors

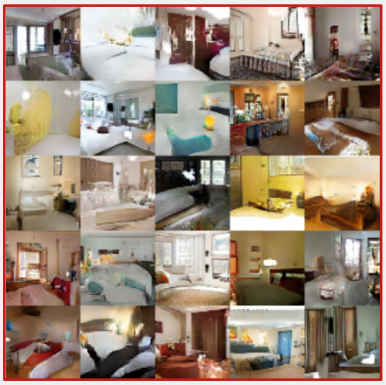
Nearest neighbors

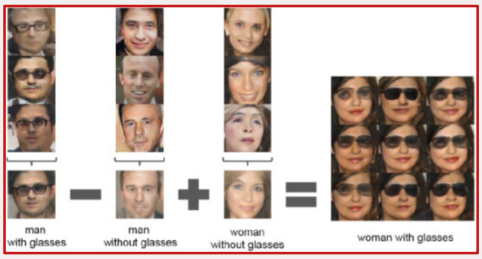
Generative Adversarial Networks

GANs: major improvement (2016)

Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016







Generative Adversarial Networks

GANs: major improvement (2016)

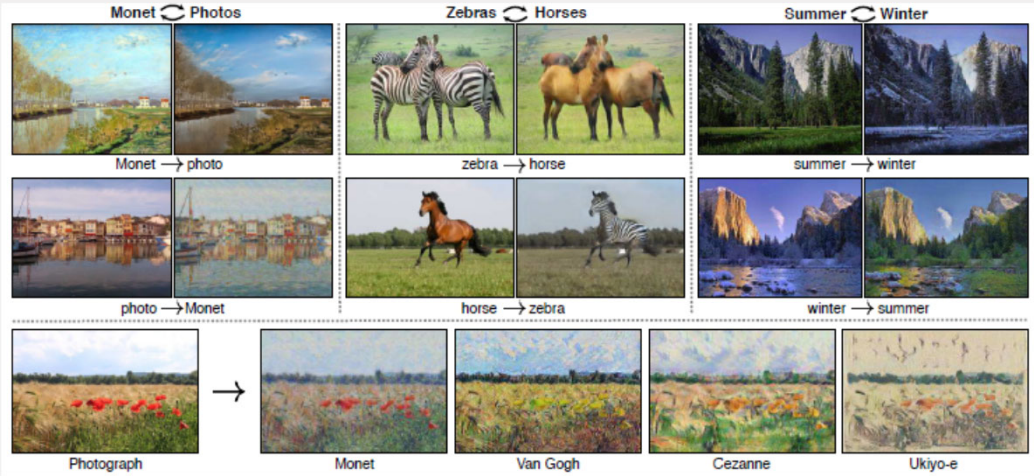
Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

Interpolation

Q: Why does this work?

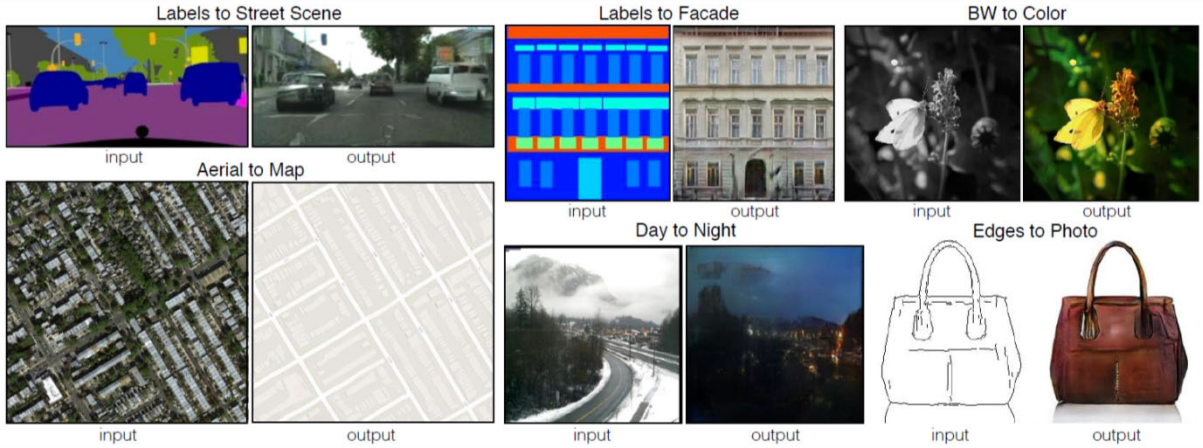
Generative Adversarial Networks

Zhu et al., "*Unpaired image-to-image translation using cycle-consistent adversarial networks*", CVPR 2017



Generative Adversarial Networks

Isola et al., "*Image-to-Image Translation with Conditional Adversarial Nets*", CVPR 2017

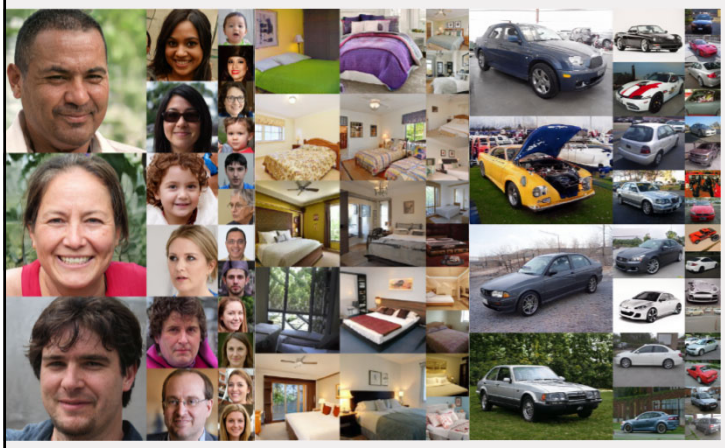


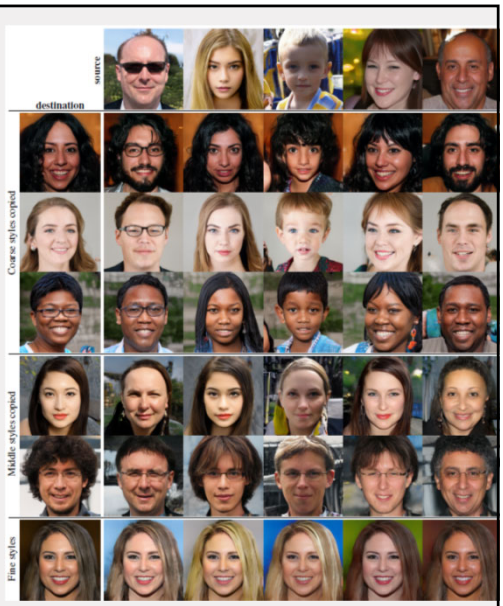
Generative Adversarial Networks

Isola et al., "*Image-to-Image Translation with Conditional Adversarial Nets*", CVPR 2017

https://arxiv.org/abs/1611.07004

Generative Adversarial Networks





Karras et al., "*A style-based generator architecture for generative adversarial networks.*" *arXiv preprint arXiv:1812.04948* (2018).

Summary

UP NEXT:
SEMI-SUPERVISED LEARNING

Unsupervised learning with convnets

Generative models

- PixelRNN and PixelCNN
  - *Explicit density model, optimizes exact likelihood, good samples.*
  - *Inefficient sequential generation*
- Variational Autoencoders (VAEs)
  - *Optimize variational lower bound on likelihood. Useful latent representation, inference queries.*
  - *Current sample quality not the best.*
- Generative Adversarial Networks (GANs)
  - *Game-theoretic approach, best samples!*
  - *Can be tricky and unstable to train, no inference queries (e.g. $p(x)$, $p(x|z)$)*