Introduction to Deep learning

Graduate School Orléans Numérique

Daniel Förster
UFR Sciences et Techniques Université d'Orléans

Agenda

Convolutional neural networks

- Convolutional operation in 1D and 2D
- Things to consider, variations: padding, strides, kernel sizes, downsampling, etc.
- Applications

Optimizing the training process

- Problem of under- and overfitting
- Optimizing the training process
- Regularization: Prevent overfitting

Autoencoder models

- Architecture
- Training
- Applications

Why convolutional neural networks?

With "fully connected" or "dense" layers:

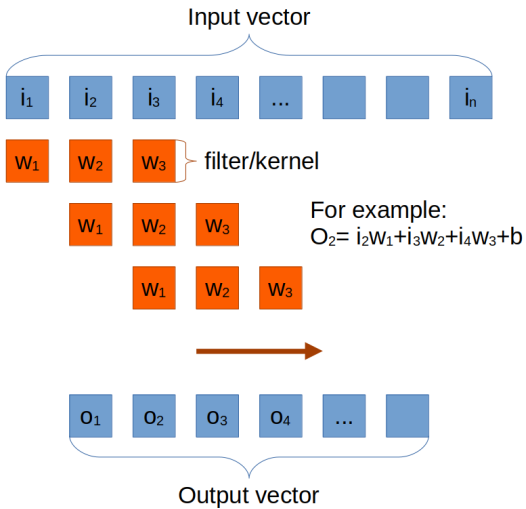
- Images are high-dimensional $\rightarrow$ large number of model parameters
- Notion of "nearby", that nearby pixels are statistically related, does not exist
- Object remains the same when shifted; however, the object must be "learned" at every position

$\rightarrow$ Embody some prior knowledge in the architecture, makes learning more efficient and decreases number of network parameters.

This is achieved by **convolutional neural networks**.

Networks equivariant or invariant to other types of transformations, such as reflections, rotations, and scaling are the subject of research.

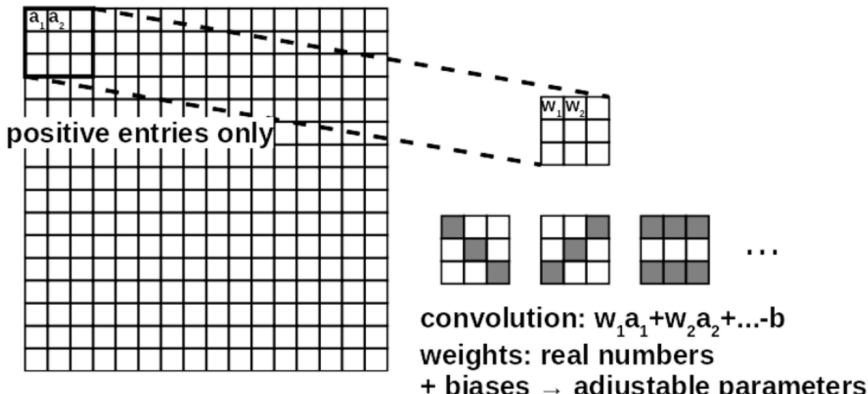
Convolution Operation 1D



Extracts local patterns

→ Convolution is **equivariant regarding translation**

Convolution Operation 2D



Convolution Operation 2D, example

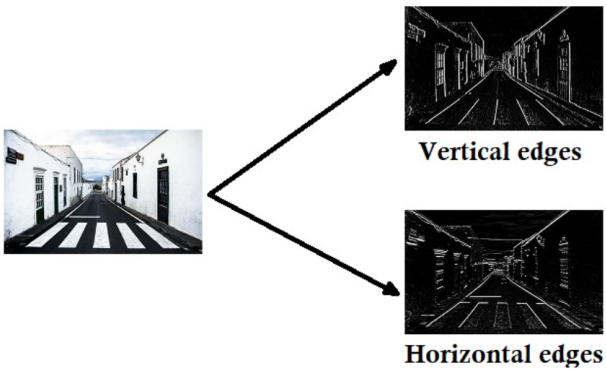


Image source [1]

1	0	-1
1	0	-1
1	0	-1

Vertical

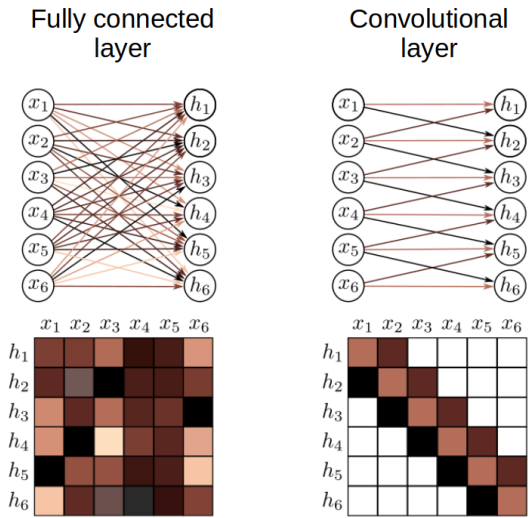
1	1	1
0	0	0
-1	-1	-1

Horizontal

Image source [2]

Each filter extracts different features from the input data, resulting in multiple "**feature maps**" or "**channels**".

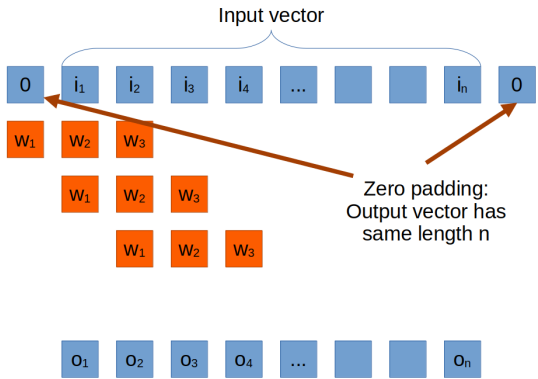
Comparison with fully connected layers



A convolutional layer is a **special case of a fully connected layer** with most weights set to zero and others are constrained to be identical.

Image source [3]

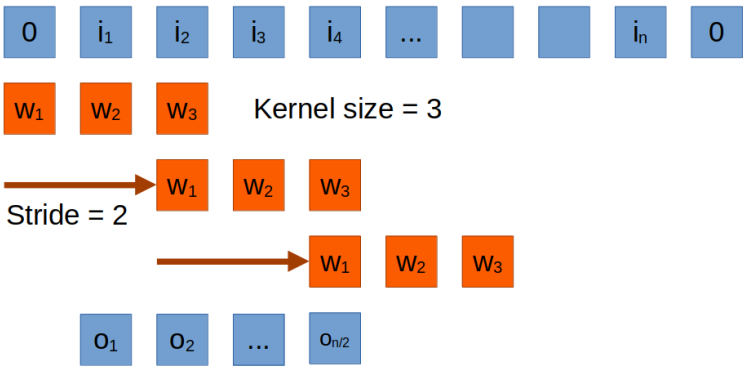
Padding



Padding adds extra pixels (typically zeros) to the input data.

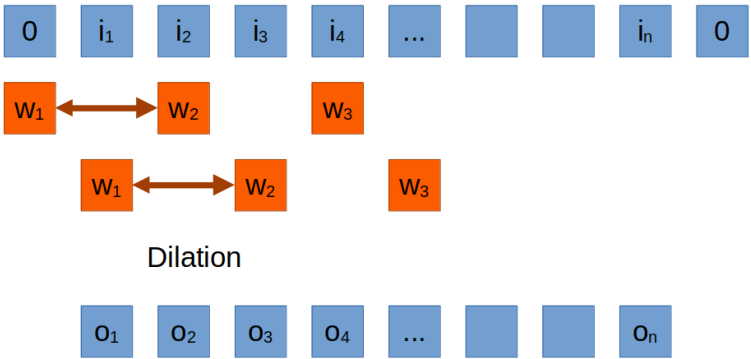
It **maintains the spatial dimensions** of the input data after the convolution operation, which may be important for some architectures.

Stride and kernel size



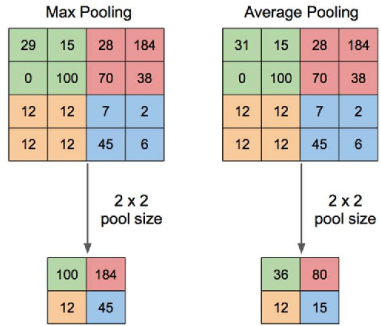
- **Kernel size:** size of the filter determines the size of the local patterns extracted from the input data
- **Stride:** step size with which the filter is moved across the input data

Dilation



Dilation can be used to increase field of view

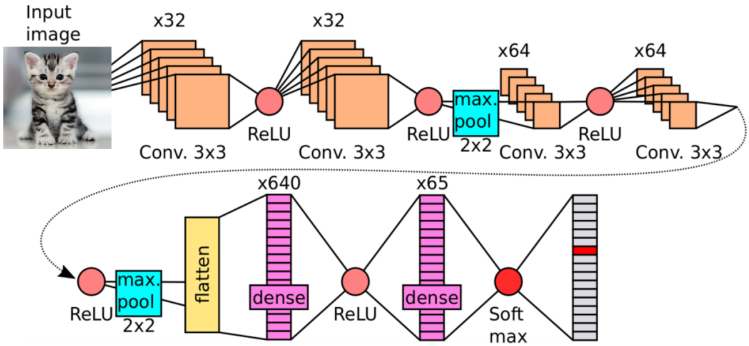
Downsampling



- Reduces the number of pixels by a factor of 4 (example here)
- Alternative to stride = (2, 2)

Image source [4]

Architecture of convolutional networks



Typical use case: **image classification**

Typical architecture:

- **convolutional layers first**, increasing number of filters
- interlaced by **max-pooling layers**
- followed by a **flattening operation**
- and finally **fully connected layers**

Examples for CNN in 1D and 3D

1D:

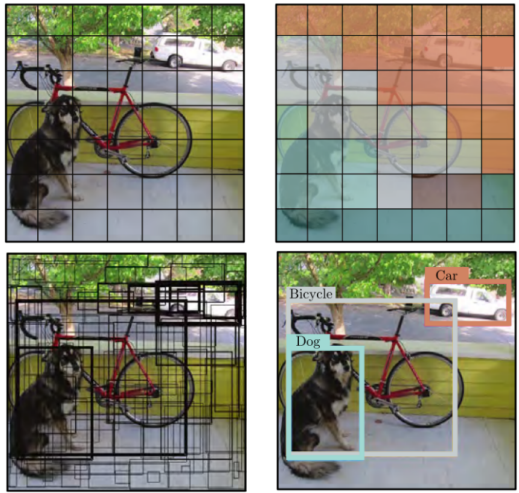
- Speech recognition
- Electrocardiogram classification
- Diffractogram analysis

3D:

- Video analysis
- Volumetric measurements

$\rightarrow$ exploit correlations in the input data, **statistically related "nearby" input data**

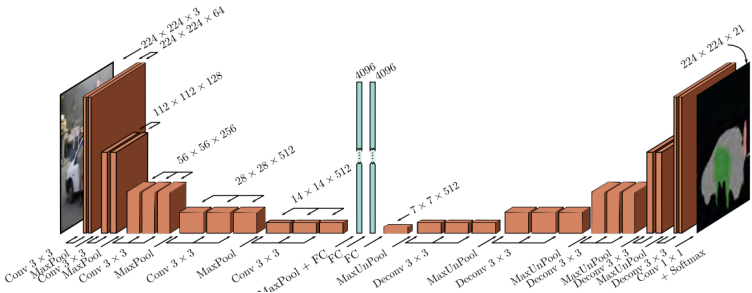
Applications - Object detection



- System predicts the most likely class for each cell
- Furthermore: two bounding boxes per cell, and a confidence value
- Finally, the most likely bounding boxes are retained

Image source [2]

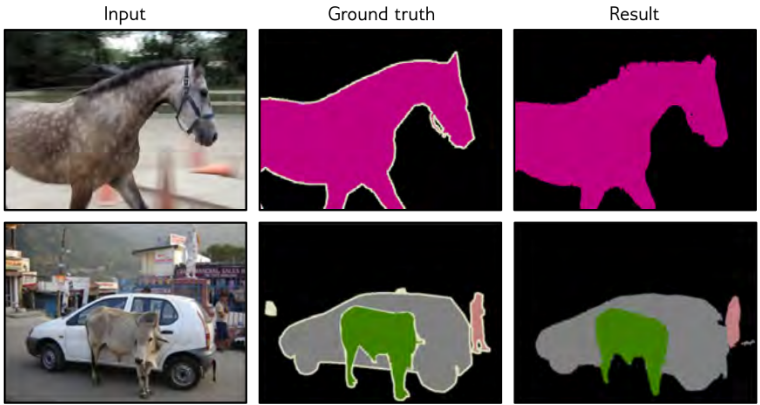
Applications - Semantic segmentation



- Two parts: "Encoder" and "Decoder"
- Final layer: classification for each pixel

Image source [2]

Applications - Semantic segmentation



- Two parts: "Encoder" and "Decoder"
- Final layer: classification for each pixel

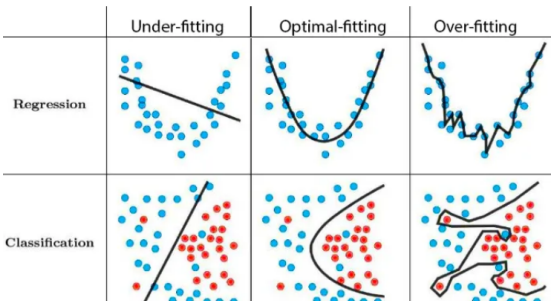
Image source [2]

Agenda

Optimizing the training process

- Problem of under- and overfitting
- Optimizing the training process
- Regularization: Prevent overfitting

Under- and overfitting

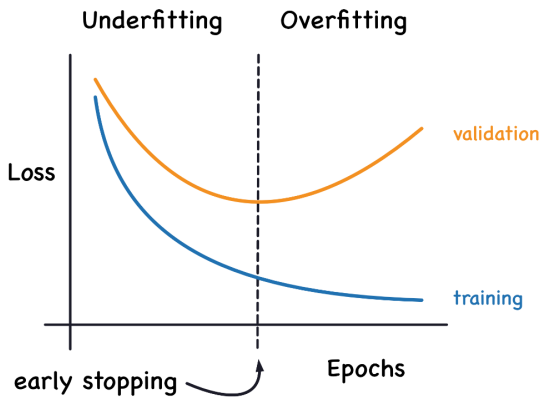


Overfitting: Model performs well on training data, but poorly on unseen data. It has essentially "memorized" the training data and fails to generalize to new instances.

Underfitting: Model performs poorly on both training and unseen data. It's too simplistic and fails to capture underlying patterns in the data.

Image source [5]

Diagnosing and combatting overfitting



- Learning rate scheduling: adjust the learning rate during training
- Early stopping: if the model's performance stops improving on the validation dataset for a predetermined number of epochs ("patience"), training is halted
- Mini batches: updates weights based on a subset of data points, not just on a single example as in the script of last week

Image source [6]

Data augmentation

Artificially expand the training dataset by applying transformations on original data

- **Combat Overfitting:** avoids memorizing specific data instances → better generalization
- **Simulate real-world:** prepares models for varied data
- **Compensate for Small Dataset:** adds artificial data

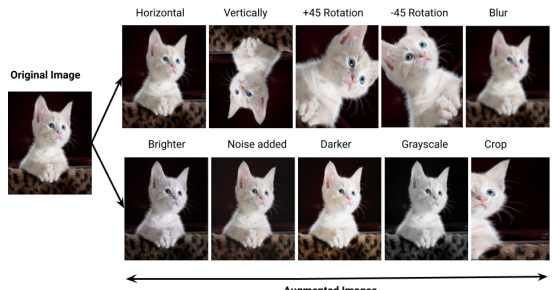


Image source [7]

Weight Initialization

Importance: avoid vanishing/exploding gradients.

Methods:

- **Zero Initialization:** All weights set to zero $\rightarrow$ not a good idea, gradient is the same for each
- **Random Initialization:** Small positive random numbers $\rightarrow$ not a good idea, on average each layer will add to the signal
- **Xavier/Glorot Initialization:** For sigmoid, tanh, softmax. Zero mean, variance $\frac{2}{n_{in}+n_{out}}$.
- **He Initialization:** For ReLU. Zero mean, variance $\frac{2}{n_{in}}$.

Batch Normalization

The key idea: Normalize the activations for a layer to have zero mean and unit variance.

- **stabilizes the training** and allows for higher learning rates
- invented in 2015

Normalization Process

Step 1: Compute Mean and Variance

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} h_i, \quad \text{(Mean)}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(h_i - \mu_B)^2, \quad \text{(Variance)}$$

Step 2: Normalize the Activations

$$\hat{h}_i = \frac{h_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

where ϵ is a small constant added to prevent division by zero.

Scale and Shift

- After normalization, the scale and shift parameters γ and β are introduced to restore the representational power of the network.

$$h_i^{\text{BN}} = \gamma \hat{h}_i + \beta$$

Learned Parameters

- γ controls the scaling of activations.
- β controls the shifting of activations.
- Both are learned through backpropagation.

Batch Normalization in Training

- During training, BN uses the mini-batch statistics for normalization.
- The mean and variance are computed for each batch and updated over time.

Running Mean and Variance

$$\mu_{\text{running}} = 0.9 \cdot \mu_{\text{running}} + 0.1 \cdot \mu_B$$
$$\sigma^2_{\text{running}} = 0.9 \cdot \sigma^2_{\text{running}} + 0.1 \cdot \sigma^2_B$$

These running averages are used during inference.

Summary Batch Normalization

- normalizes activations to have **zero mean** and **unit variance** and then **scales and shifts** them using learned parameters
- reduces internal **covariate shift**
- **regularization effect** due to coupling of examples in the mini-batch $\rightarrow$ less prone to overfitting

Regularization

- **L1 regularization**: $L' = L + \lambda \sum_i |w_i|$ Penalizes absolute values of weights. Encourages sparsity.
- **L2 regularization**: $L' = L + \lambda \sum_i w_i^2$ Penalizes squared values of weights. Prevents overfitting.
- **Dropout**: Randomly drop units during training to prevent reliance on any single neuron.
- **Batch normalization**: Normalizes activations (mean and variance) in a layer for each mini-batch during training.

Hyperparameter Tuning

Hyperparameters: everything that's not parameters (i.e. weights and biases)

- **Number of Layers and Units:** Depth and width of the neural network
- **Learning Rate:** Determines the step size during optimization
- **Batch Size:** Number of samples used in each iteration
- **Activation Functions:** Non-linear transformations applied to layers
- **Dropout Rate:** Fraction of neurons to ignore during training
- **Weight Initialization:** Strategy to set initial values of weights
- **Optimizers:** Algorithm to adjust model parameters (e.g., SGD, Adam)

Tuning hyperparameters is as much an art as it is a science. Iterative testing and validation are key.
Use also **ensembling**: Combine multiple models with different hyperparameters

Agenda

Autoencoder models

- Architecture
- Training
- Applications

Background and Motivation

- **What are autoencoders?**
  - Self-supervised learning models
  - Aim to learn a compressed representation of data
- **What are they good for?**
  - Dimensionality reduction / Data compression
  - Feature learning
  - Denoising data
  - Super resolution
  - Anomaly detection
  - Generative models

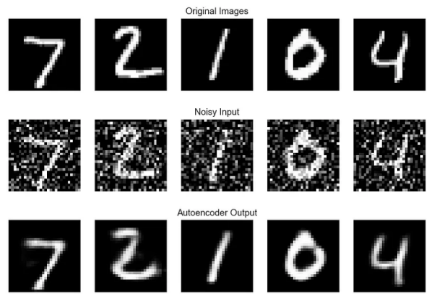


Image source [1]

Basic Concepts

- **Autoencoder Structure**
  - Composed of two main parts: encoder and decoder
  - Encoder compresses the input into a lower-dimensional code
  - Decoder reconstructs the input from the code
- **Learning Objective**
  - To minimize the difference between the input and its reconstruction
  - Achieved by using a loss function, such as Mean Squared Error (MSE)
- **Encoder and Decoder Functions**
  - Typically implemented as neural networks

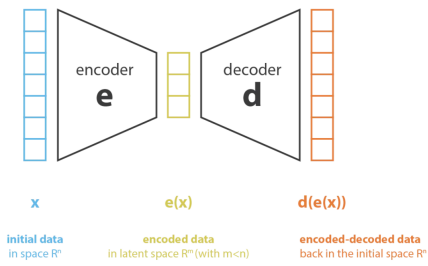


Image source [2]

Encoder

Shares the structure of the deep neural networks discussed previously

- Maps input data to a lower-dimensional latent space.
- Learns the most salient features of the data.
- Acts like PCA but with non-linearity, and captures more complex patterns than linear methods.

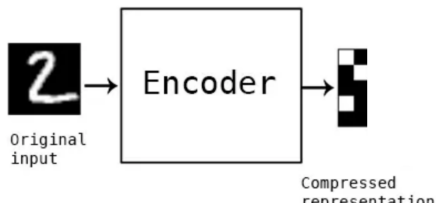


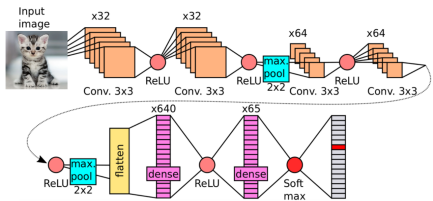
Image source [3]

Encoder

Shares the structure of the deep neural networks discussed previously

- Maps input data to a lower-dimensional latent space.
- Learns the most salient features of the data.
- Acts like PCA but with non-linearity, and captures more complex patterns than linear methods.

Image source [3]

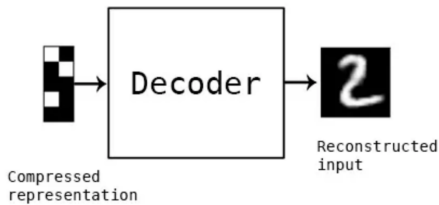


for comparison

Decoder

Reconstruction Process:

- Mirrors the encoder in structure, reversing the process.
- Transforms the code back into a representation of the original input.
- Loss function can guide the learning towards better reconstruction (e.g., MSE, Cross-Entropy)
- Typical layers: dense, transposed convolutions, upsampling



Decoder and encoder are trained together.

Image source [3]

The bottleneck or latent space

Latent Space Representation or the “heart of the autoencoder”:

- Represents the compressed knowledge of the input data.
- The dimensionality determines the level of compression.
- Captures the essential features needed for reconstruction.
- The "code" from which the input is reconstructed during decoding.

Importance of Dimensionality:

- Too small: may not capture all relevant features.
- Too large: may lead to overfitting and not enforce a useful representation.

Visualization of the latent space (2D) for MNIST dataset

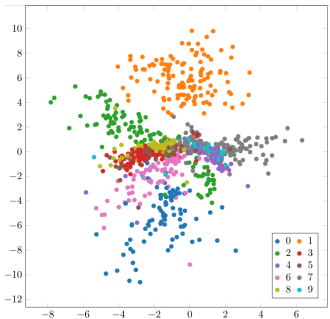


Image source [4]

Applications - Denoising Autoencoders

- Autoencoders can be trained to remove noise from the input data.
- The model learns to reconstruct the clean signal from the corrupted one.

Training Process:

- Input data is deliberately corrupted with noise during training.
- The target data for reconstruction remains the original, uncorrupted data.

Image source [3]

Applications - Super-Resolution Imaging

Autoencoders can be trained to increase the resolution of images.

Training Process:

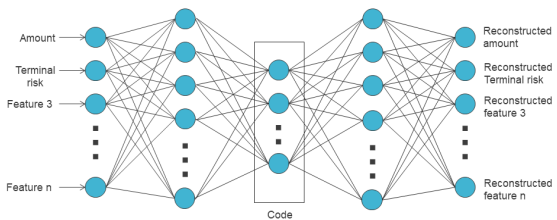
- Resolution of input data is deliberately reduced during training.
- The target data for reconstruction remains the original.

Denoising and Super-Resolution work only for images of the domain of the training database.

Image source [5]

Applications - Anomaly detection

Anomalies will typically have a higher reconstruction error during testing since they deviate from the learned patterns.



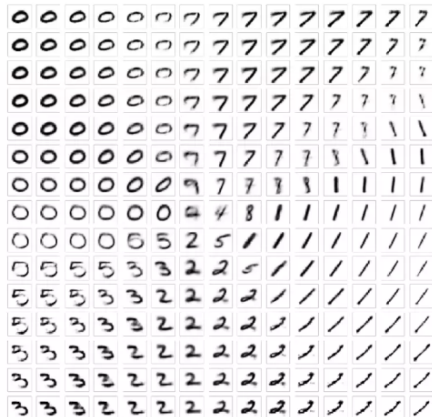
Examples: credit card fraud detection, industrial defect detection, network intrusion detection

Image source [6]

Applications - Image generation

Objective: "Fake" data that look realistic.

Idea : Create a random vector in the latent space and run it through the decoder.

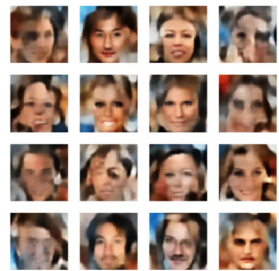


For sharper images, GANs and diffusion models are state-of-the-art; VAEs provide principled latent spaces.

Applications - Image generation

Objective: "Fake" data that look realistic.

Idea : Create a random vector in the latent space and run it through the decoder.



For sharper images, GANs and diffusion models are state-of-the-art; VAEs provide principled latent spaces.