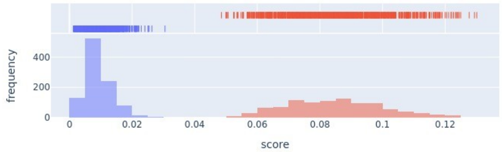
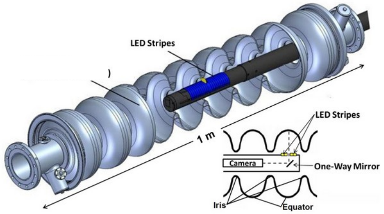
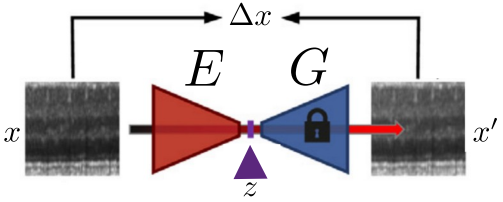
Preparatory Project



Automized Anomaly Detection
via
Unsupervised Machine Learning
Trained on the OBACHT Dataset





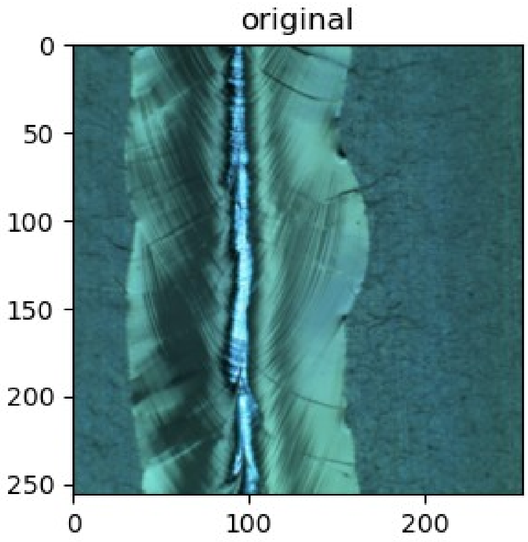
Overview

- Goals
- Database - OBACHT
- Anomaly Detection with Autoencoders
- Application
- Outview

Goals

1 find visual defects **automatically**

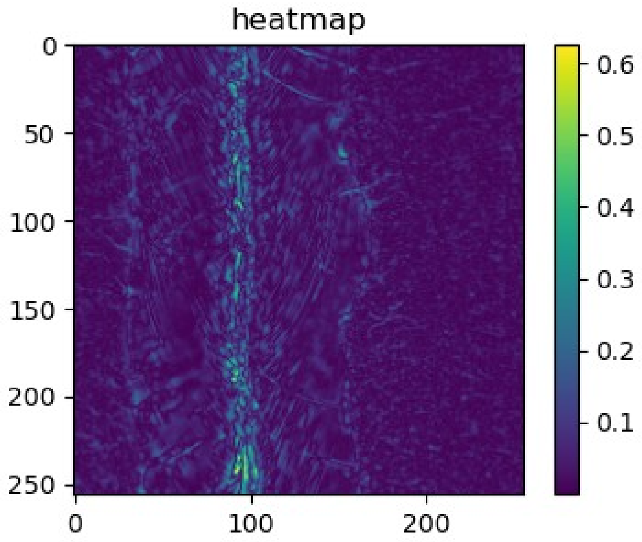
2 find correlations with cavity performance



ML

$\longrightarrow$

1



Analysis

$\longrightarrow$

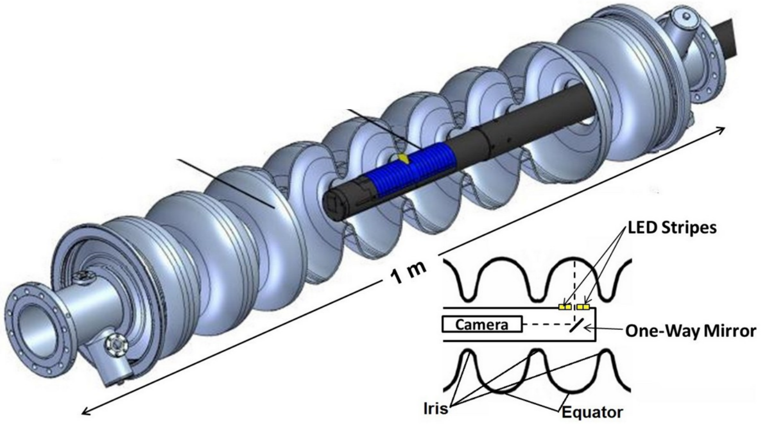
2

$$Q, E_{max}, \ldots$$

Database - OBACHT

- **large database**
  - ~350,000 source images
- **high quality**
  - resolves structures up to 4µm
  - 3 color channels 3,488x2,816 pixel each
- **high variety**
  - multiple stages of **chemical preprocessing**
  - multiple **vendors**
  - 9 cell / 1 cell
- **caveat**
  - unsystematic scanns (e.g. some vendors are more reserved)

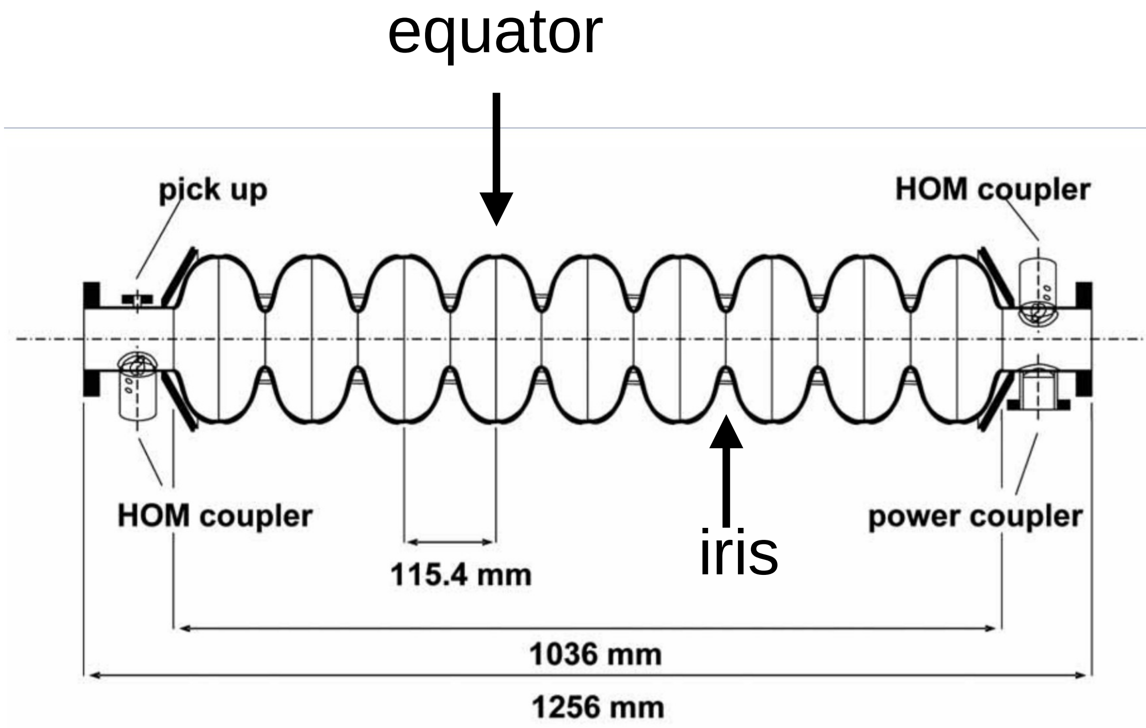
Wenskat 2019



images of cavity **inner** surface!

Focus

- **visual** anomalies only
- equatorial region only
  - highest performance impact expected (highest surface mag. field)
  - 120,000 images left



DESY Report 2006–097 p. 67

ML Approach:
~~GAN~~
Autoencoder

Detection Principle

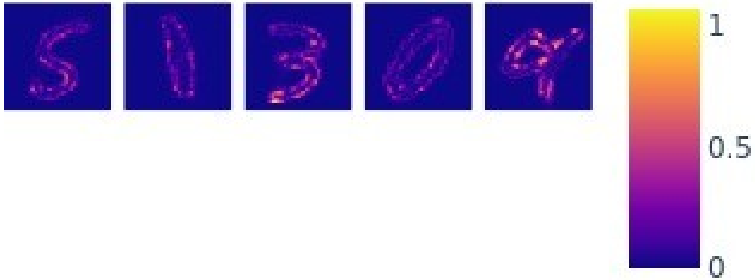
original

ML

reconstructed

normal/common → no change

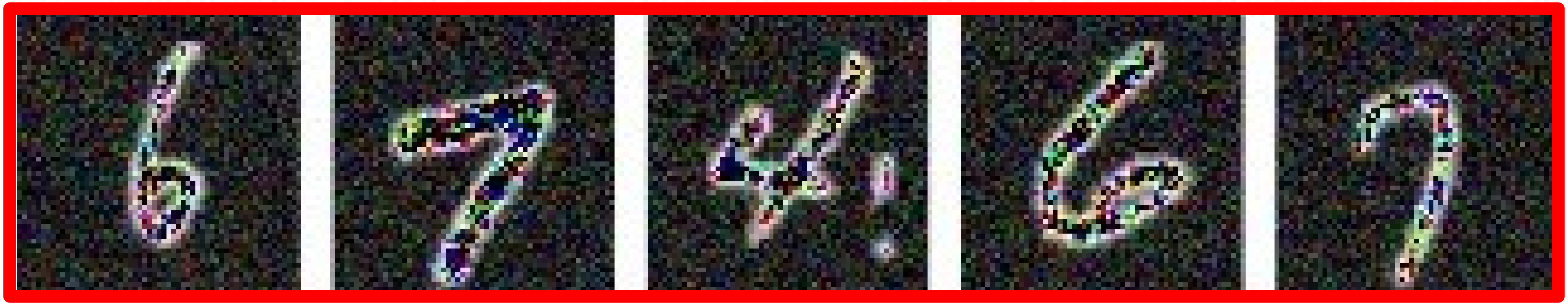
difference



a.u.

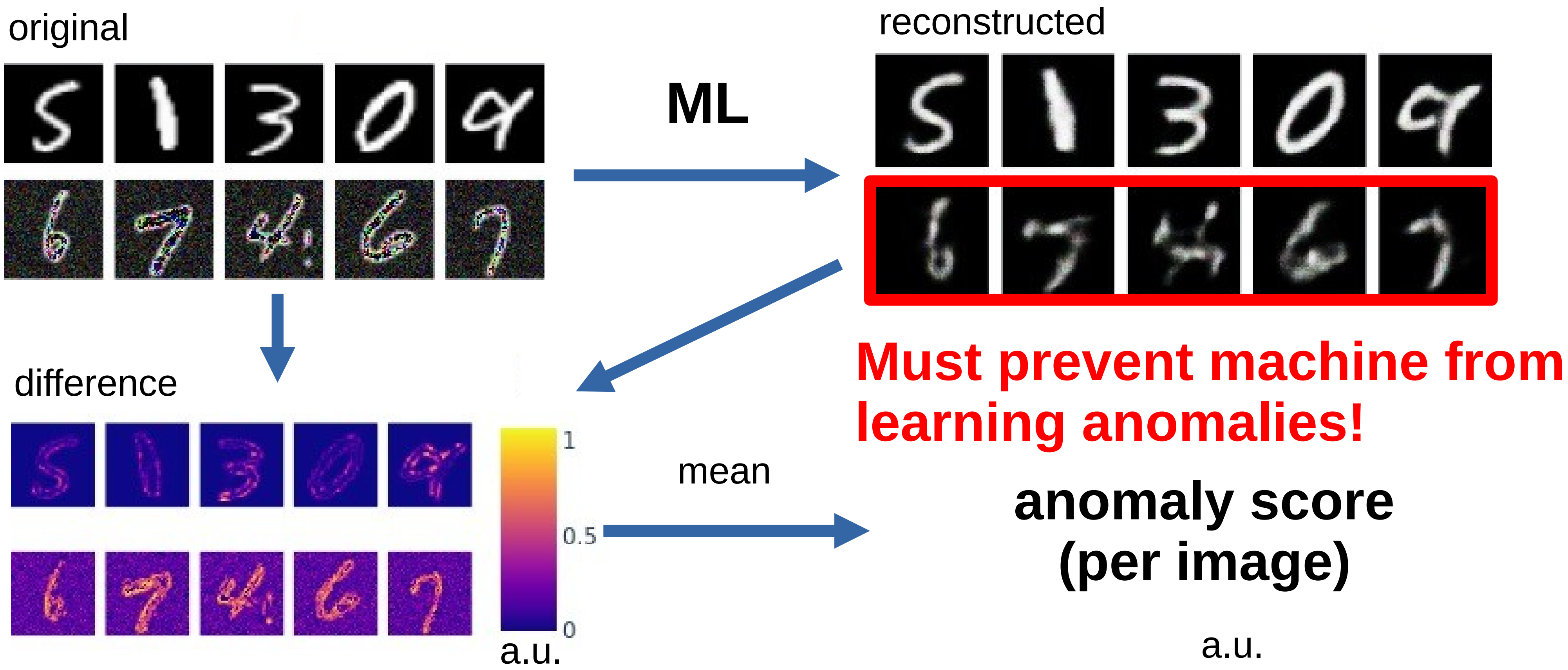
original images from MNIST

Anomaly!



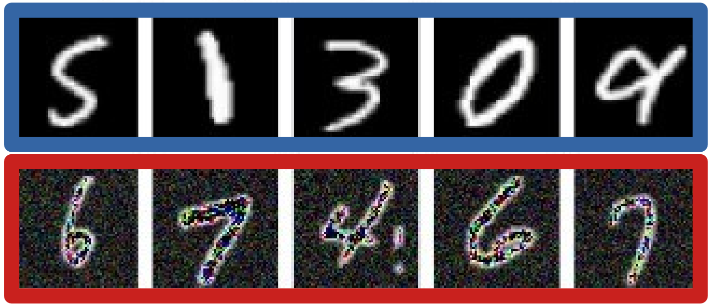
unseen/uncommon structures

Detection Principle



original images from MNIST

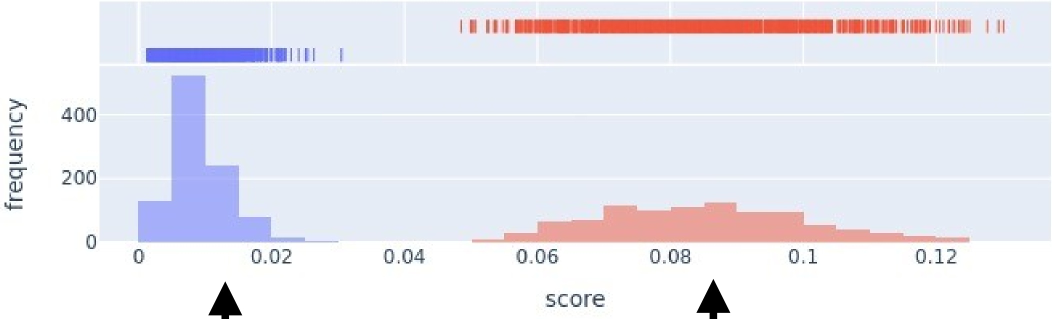
Usage



normal

anomalous

anomaly score (per image)



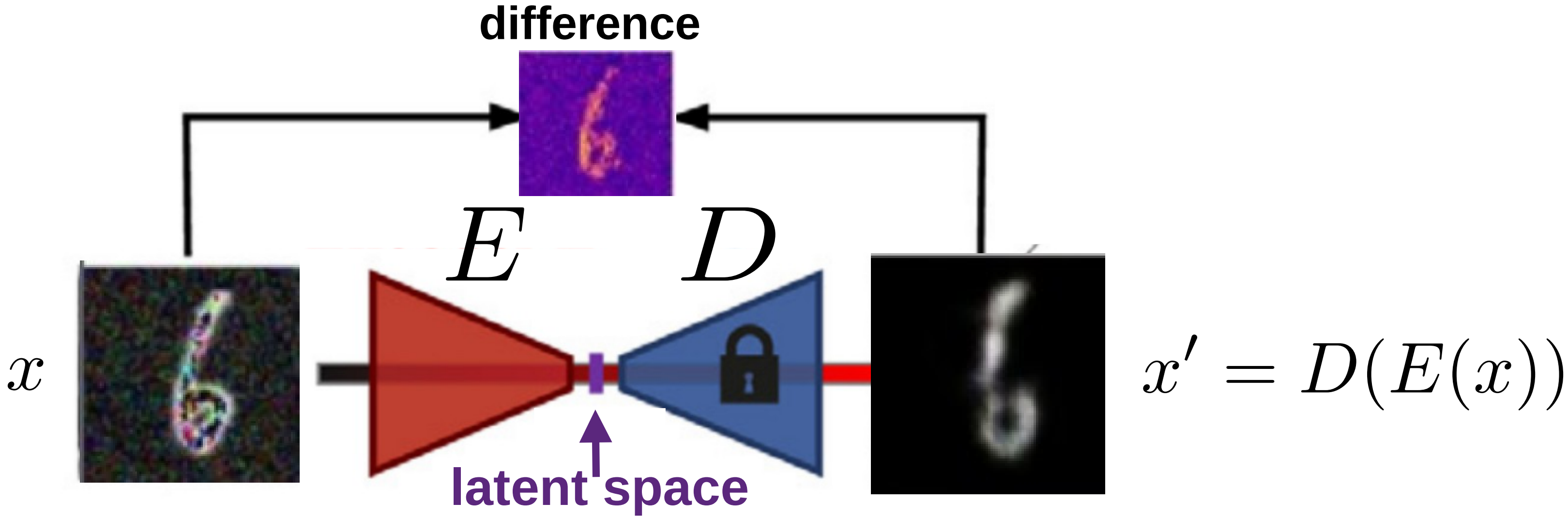
bunched by category!

correlate this with Q, Emax, ...

Autoencoder Principle

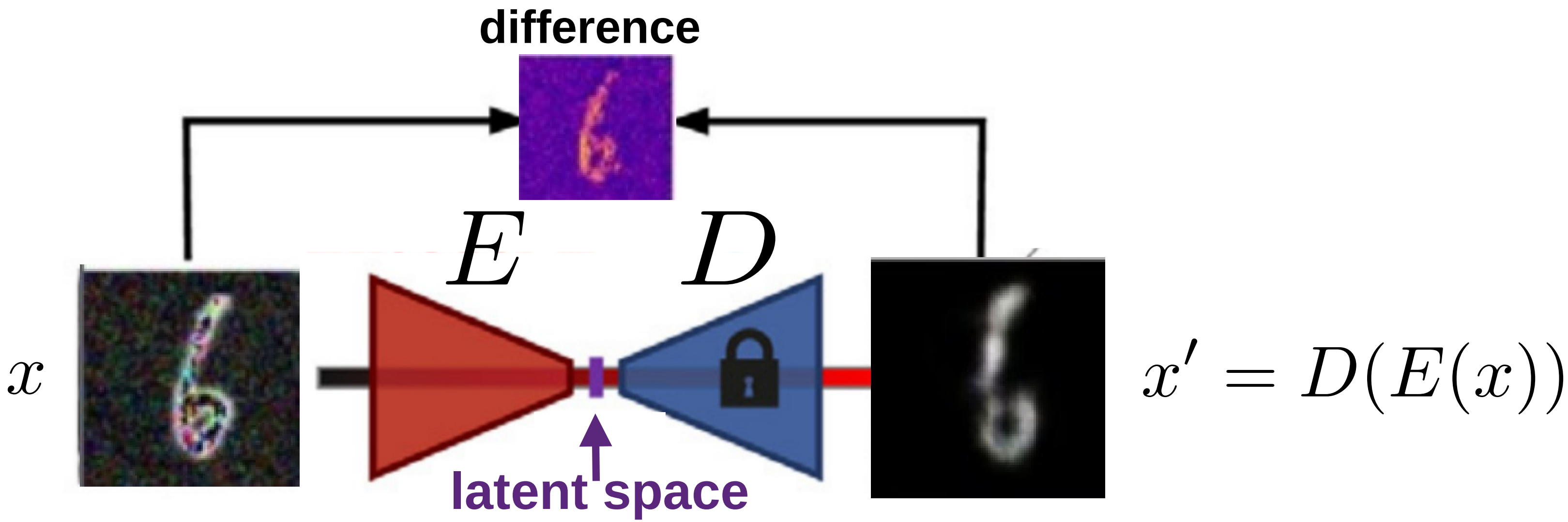
1 **lossy!** image compression via **encoder** E

2 image reconstruction via **decoder** D

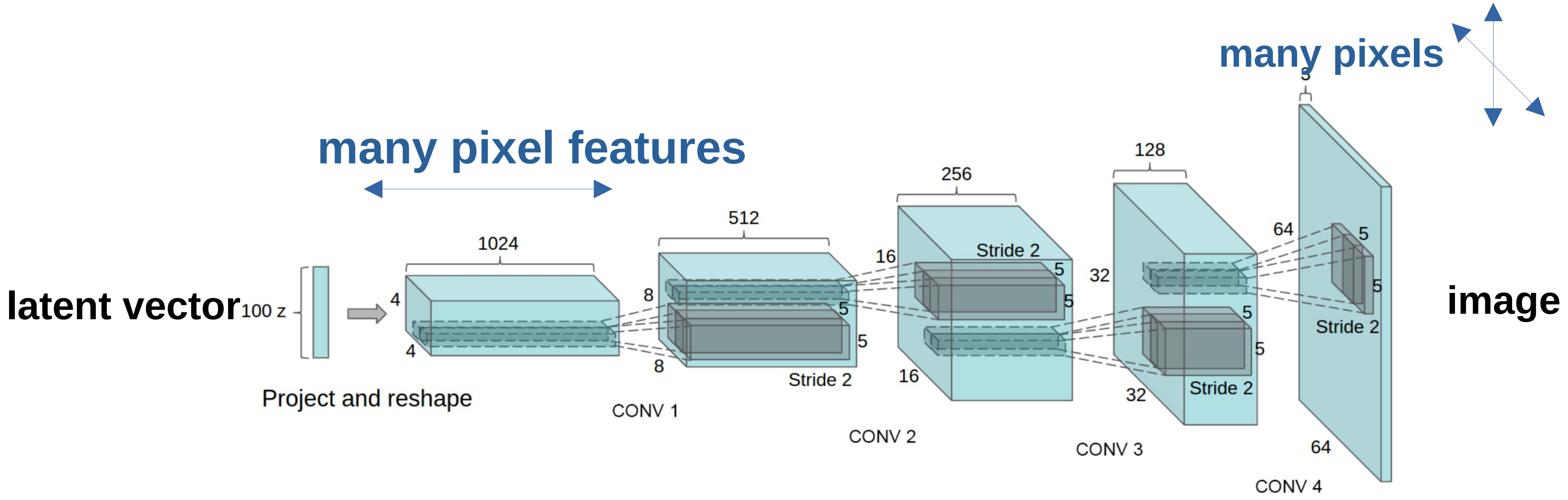


Reconstruction Principle

- **preserve** `normal` features
- **forget** `anomalous` features



Decoder Architecture: CNN

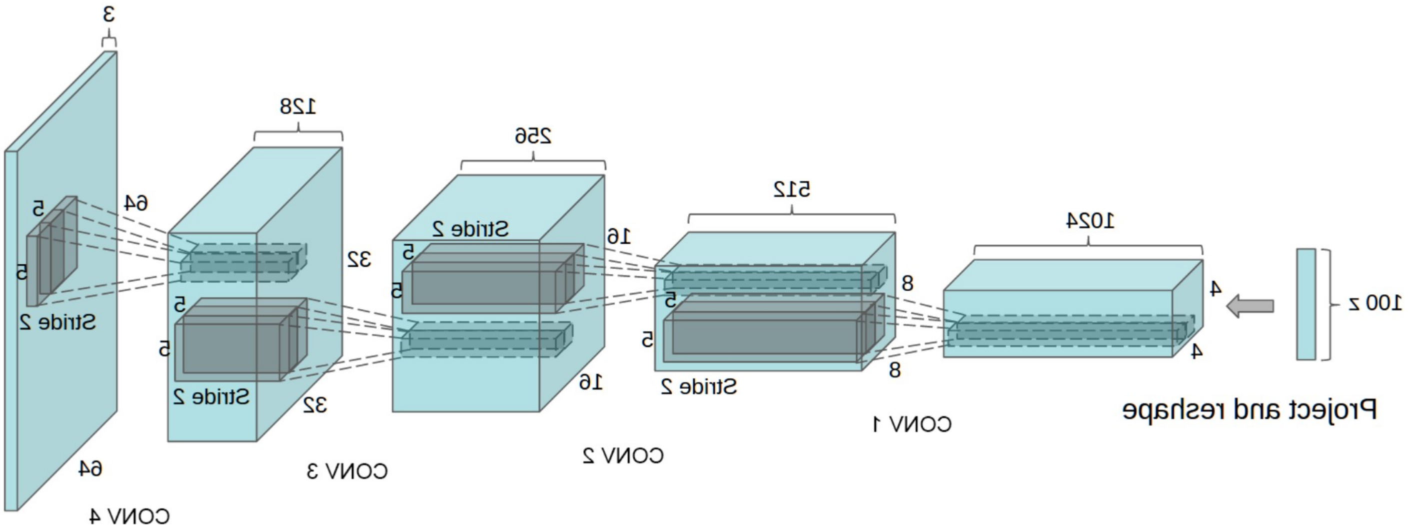


Radford et al. 2016

Convolutional **N**eural **N**etwork

Encoder Architecture: **reversed**/transposed CNN

image



latent vector

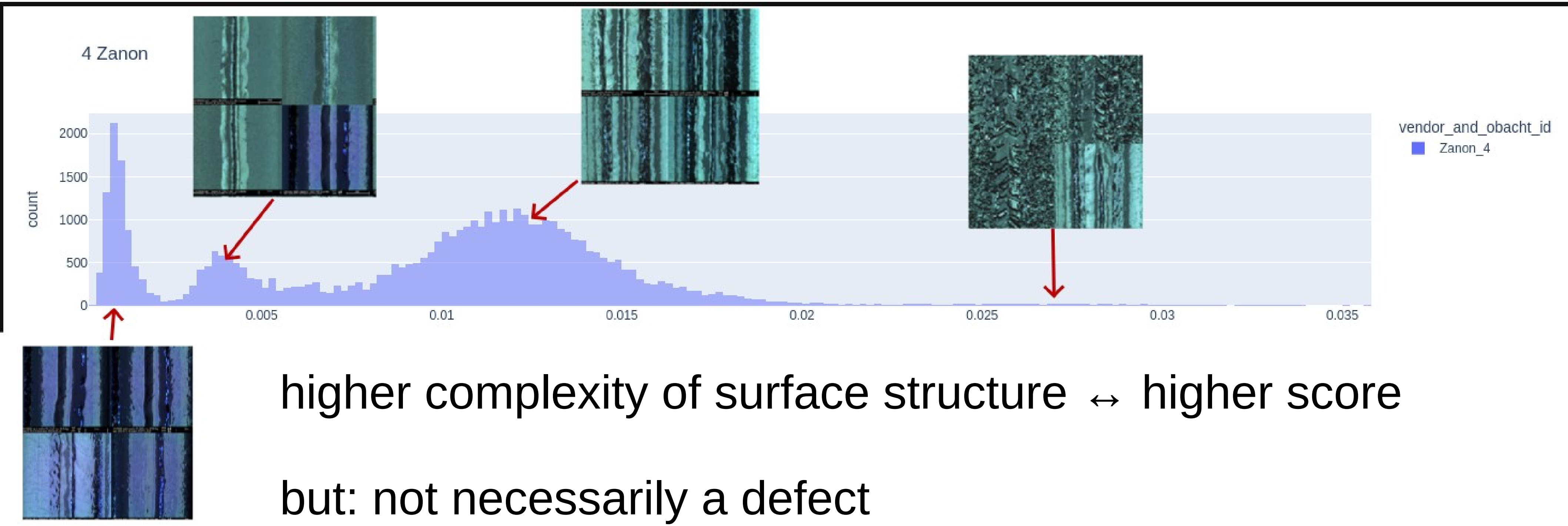
Application

Training Data

- **manually selected** ~25,000 ‚**normal**’ images
- spans across
  - multiple vendors
  - multiple processing stages
  - cell numbers per cavity (9 & 1)

→ ‚**representative**’

Anomaly Score Histogram



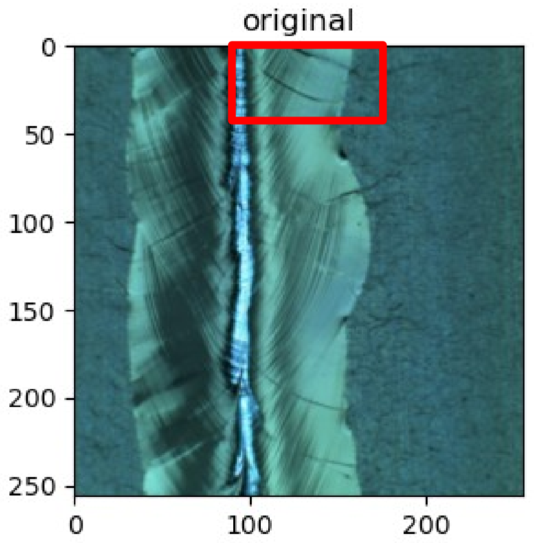
higher complexity of surface structure ↔ higher score

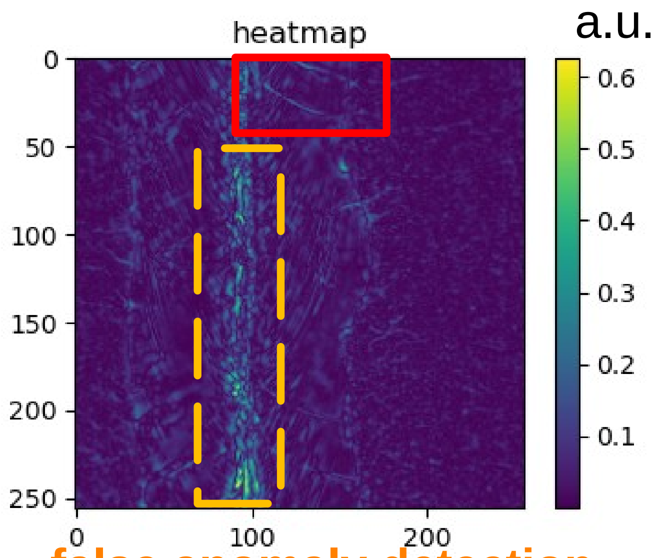
but: not necessarily a defect

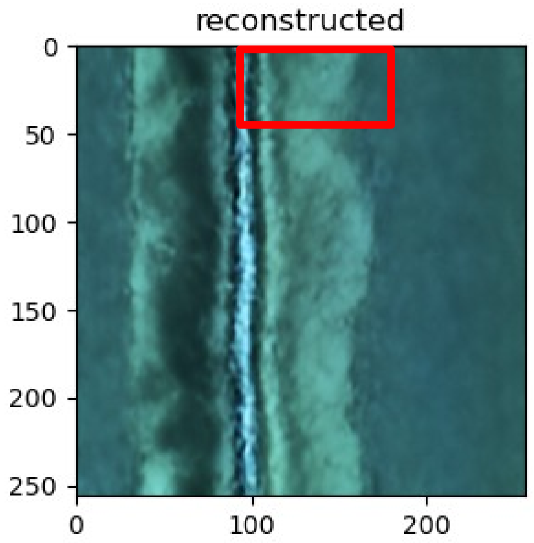
Heatmap

mostly normal
with some **anomalies**

anomaly not reconstructed







false anomaly detection

heatmap = difference

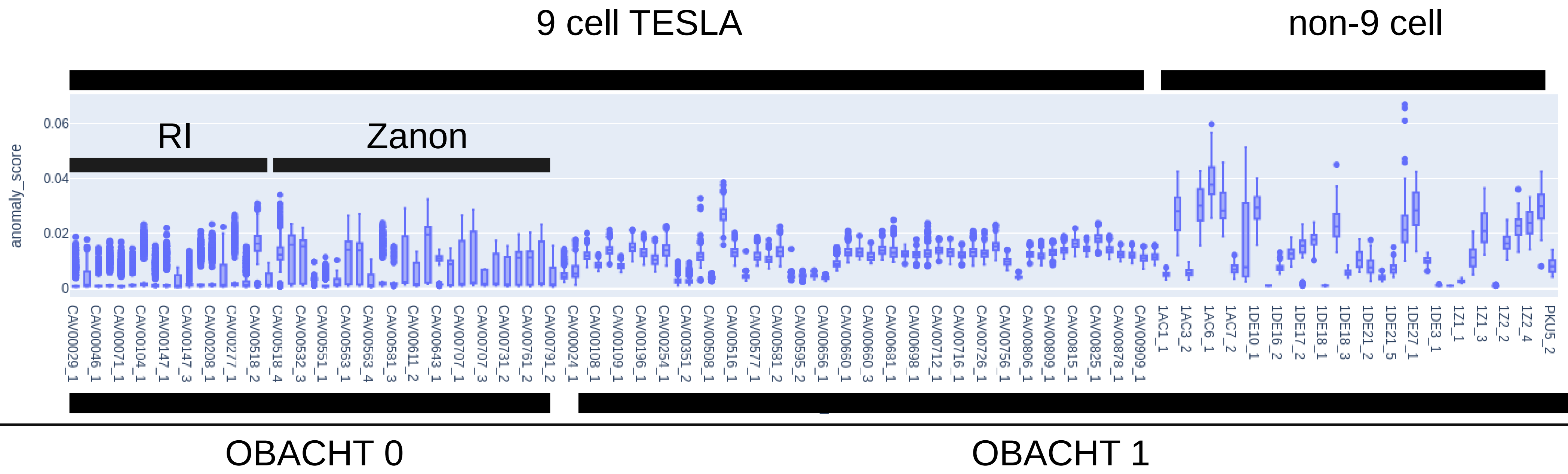
score = average of heatmap

false positive	true positive
e.g. **welding seam**	**anomaly**
→ has a lot of ‚character‘	→‘unknown’
→ fails to reconstruct	→ fails to reconstruct
→ high score	→ high score

Challenges

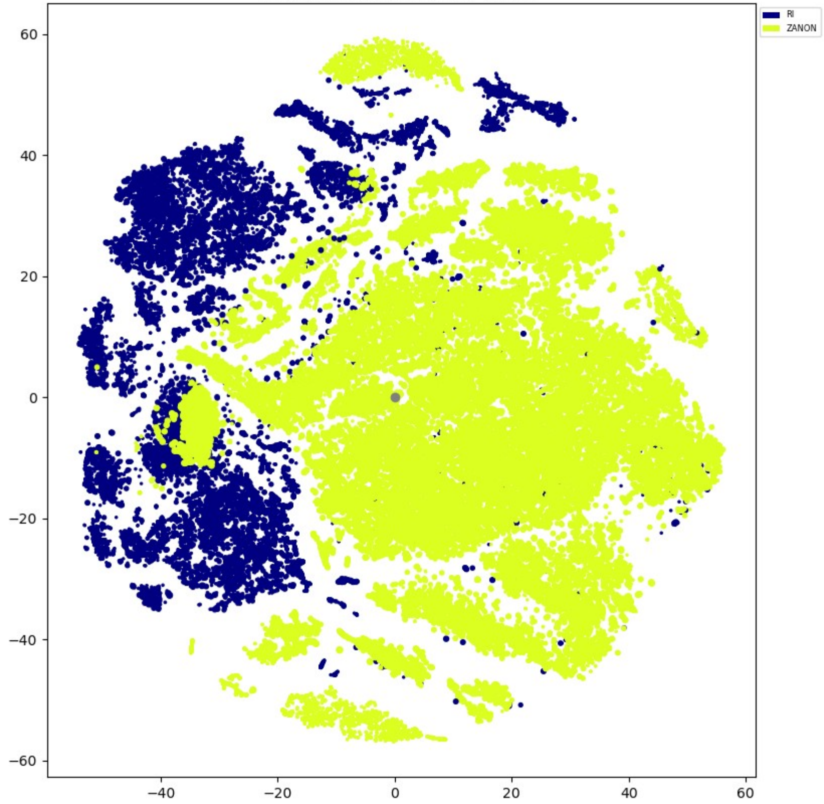
- **correlations** with global properties of cavities
  - due to aggregation (difficult to find ***the*** spot)
  - global vs. local
- *usable* **heatmap**
  - pinpoint defects inside an image

Anomaly Scores Per Scan



Deep Dive: Latent Space

By Vendor



TSNE: projection* 2048D → 2D

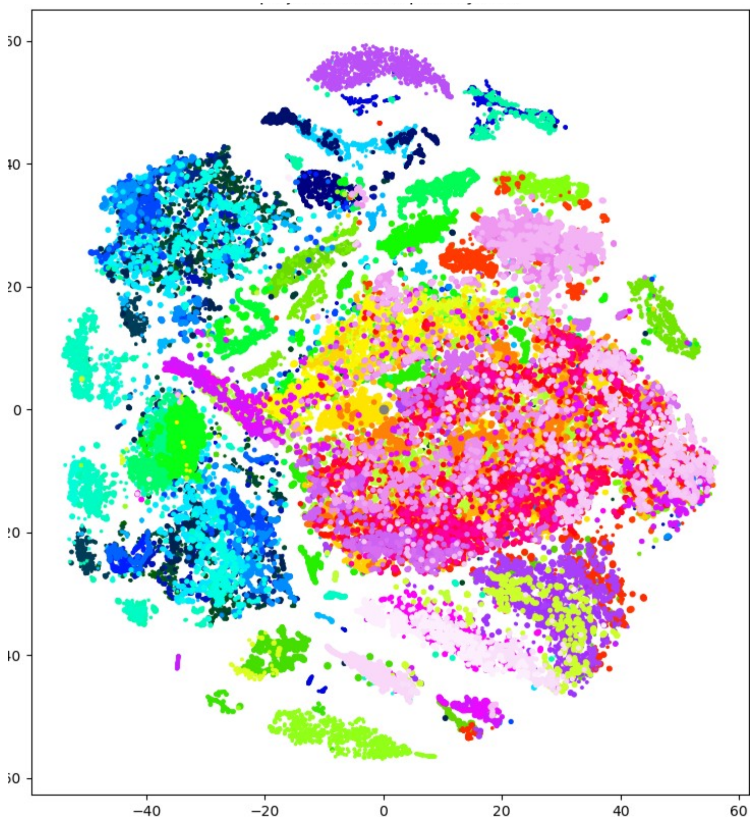
color = **vendor**

clustering evolved **unsupervised**

* Does not strictly represent distance.
  Mapping is statistical.

Deep Dive: Latent Space

By Cavity Scan



TSNE: projection 2048D → 2D

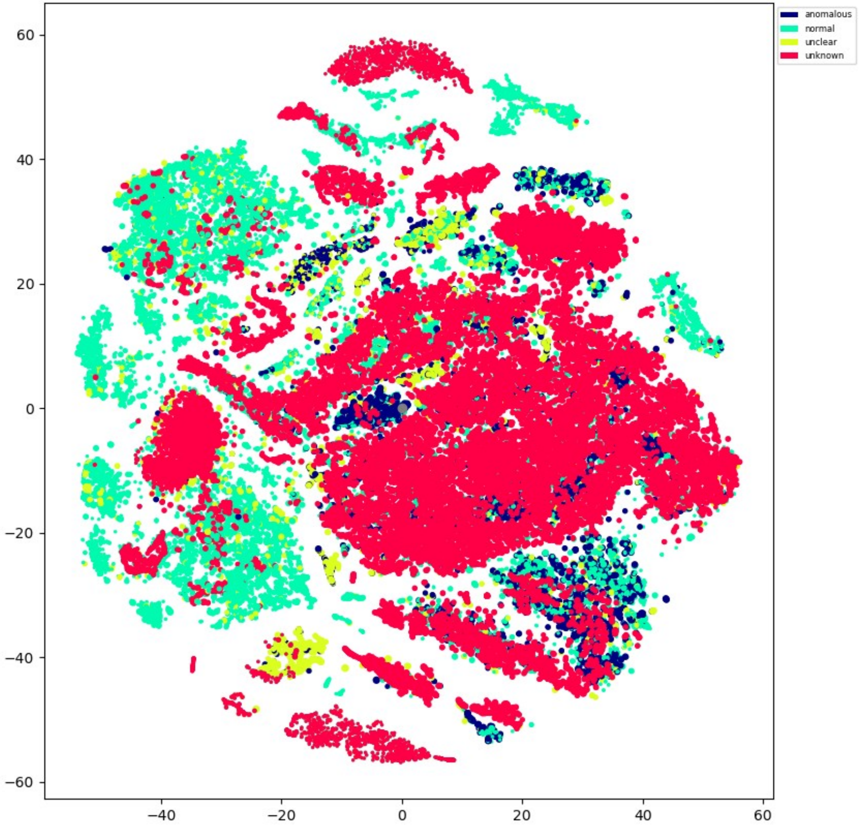
color = **cavity scan**

clustering evolved **unsupervised**

note: AE is **not variational**

Deep Dive: Latent Space

By Class



TSNE: projection 2048D → 2D

color = normal/anomalous/unclear/unclassified

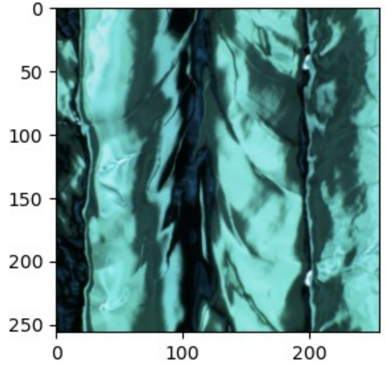
as expected: anomalous is mapped as if normal

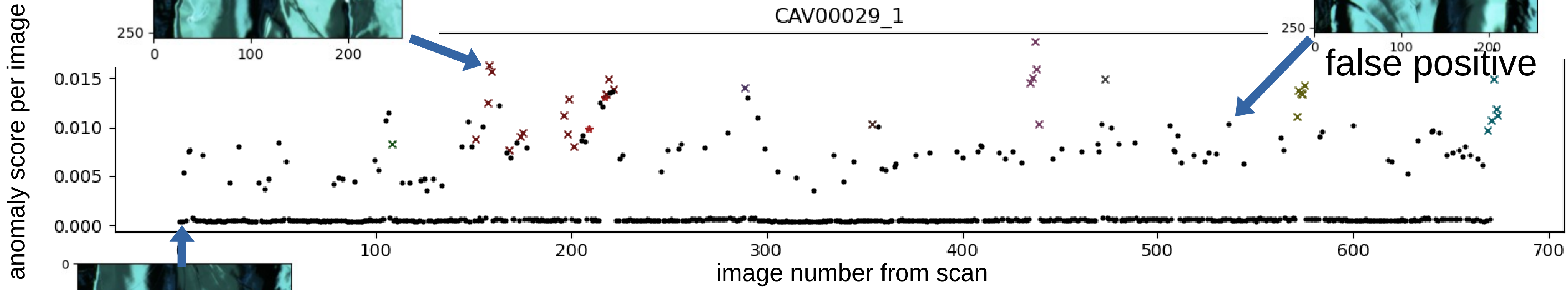
normal = training data

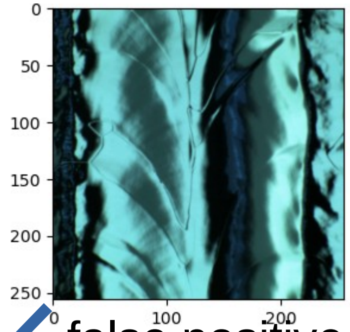
In Detail: Score per Scan

example for a ‚**good**‘ cavity
(high max. field + few visual anomalies)

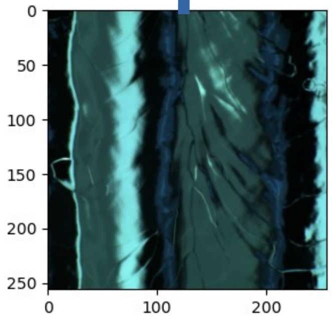
true positive





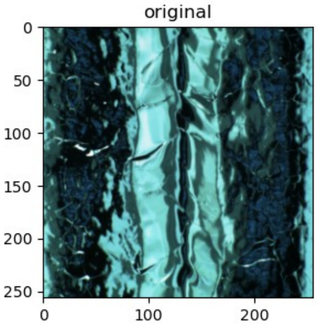


false positive

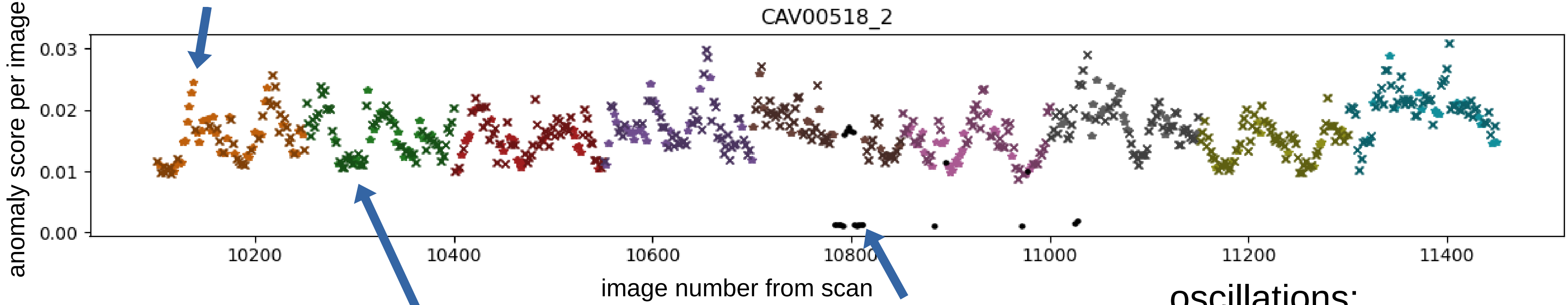


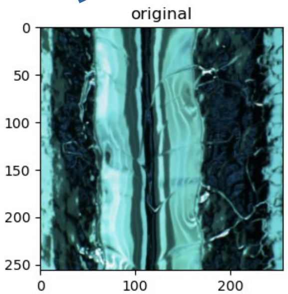
true negative

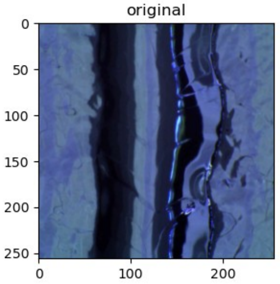
In Detail: Score per Scan



example for a ‚**bad**‘ cavity
(many visual anomalies + low max. field)







oscillations:
2-3 periods per cell

origin:
- manufaturing
- camera missalignment

Autoencoder

- works best for
  - finding images which have **some anomaly**
- does **not** work for
  - **pinpointing** exact location of anomaly

Correlations

data taken from the **Cavity DB** hosted by DESY

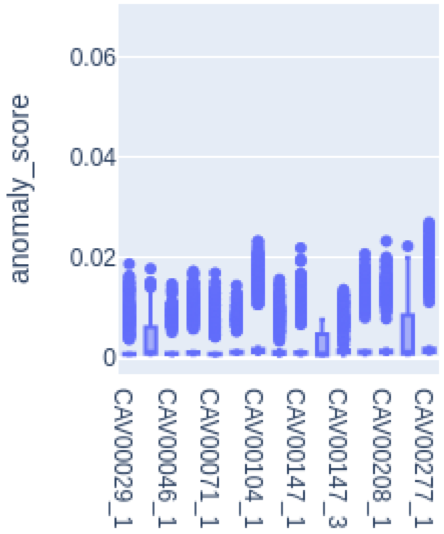
→ unfortunately **no correlations found** (yet)

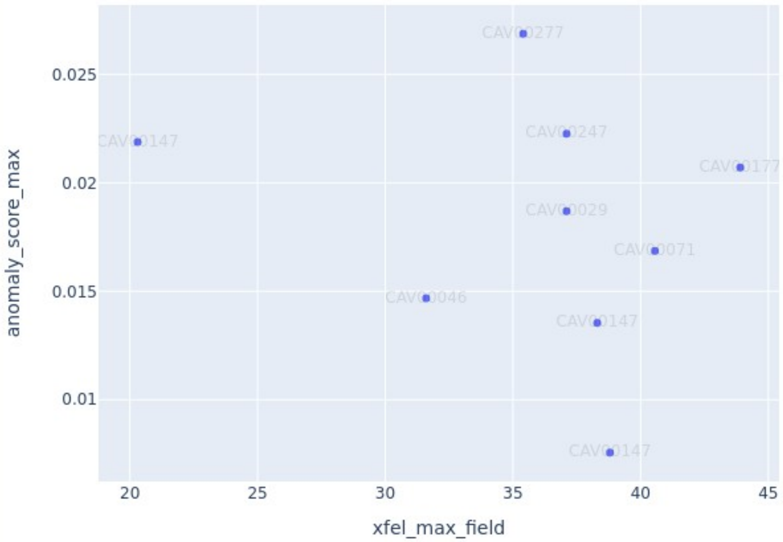
- challenges:
  - many **local** images **vs** few **global** cavity properties
  - unsystematic scans
  - traning data is across all stages of chemical preprocessing (or else not enough trainig data)

Correlations?

Example: OBACHT 0 RI only

aggregation: max





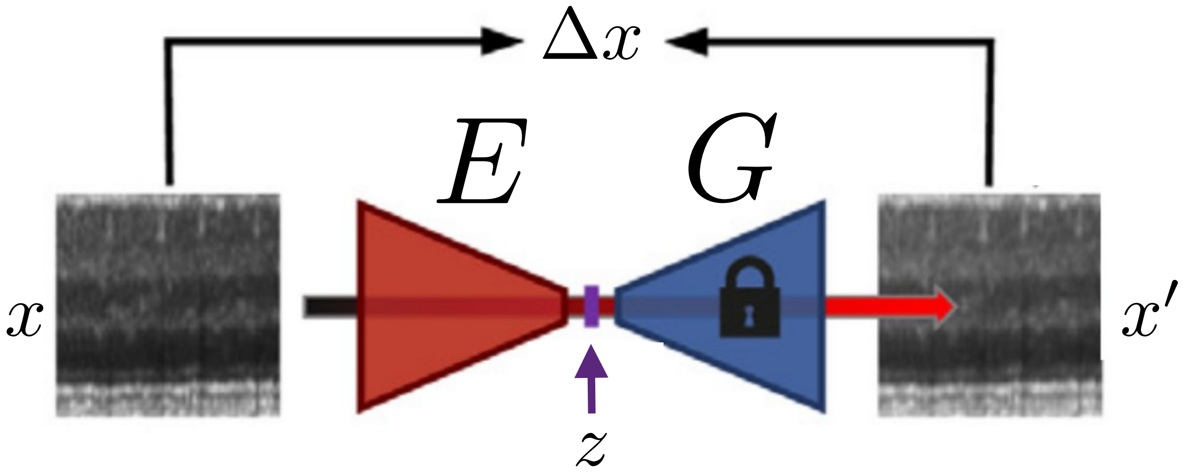
no correlations

Practical Advice

- the more **local** the **physical data** the better
  - e.g. single cell cavities
  - e.g. coloring the defects (per-pixel info; manually or temp. map) (see Schlegel et al 2019)
- ensure physical **data** and images are
  - **machine friendly** at all times
  - homogenious in shape
  - **systematically obtained**
- **GANs** produce images of *subjectively* better quality with ***worse* reconstruction errors**
- **VAEs** have wore **reconstruction errors** than AEs (preliminary result)

Summary

- **unsupervised** ML to detect anomalies
- Autoencoder ✅
- Correlations ?



Summary

- AE bad because anomalies are not localized enough
  - good for general score
  - bad for finding spot

Outview

- improve **aggregation** methods
- refine filtering of training data
  - e.g. focus on after chemical treatment only

Acknowledgements

Special thanks to:

- Antonín, Marc and Annika
- The OBACHT team
- The DESY Cavity Database Team

References

- Schlegel et al. 2019: DOI 10.1016/j.media.2019.01.010
- Radford et al. 2016: DOI 10.48550/arXiv.1511.06434
- Wenskat 2019: DOI 10.1088/1748-0221/14/06/P06021

Appendix

Intermezzo:

Some ML Recap

Neural Network

- sequence of functions (layers)
- applied in order
- **simple, differential** functions

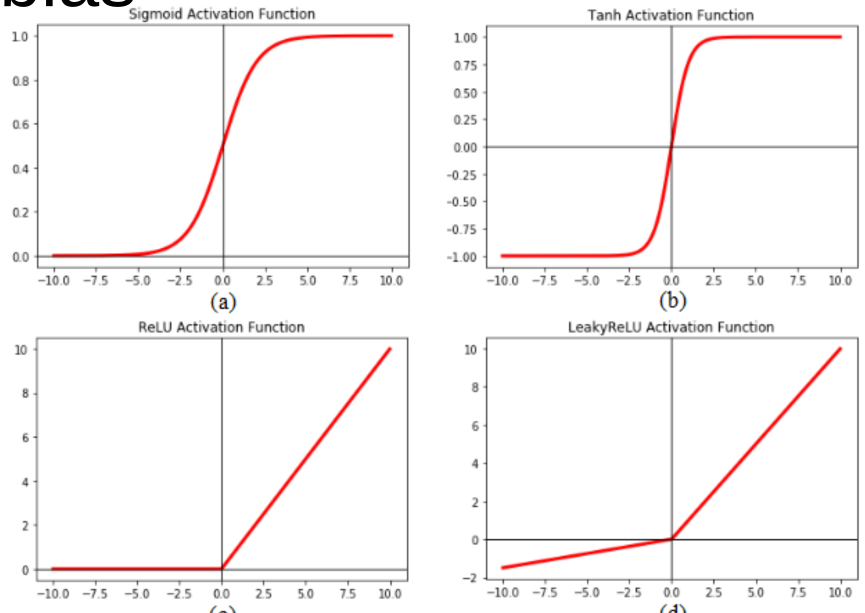
- → **fitting** the curve to data

Fundamental Layer Types

$$\mathrm{y}_i = \sum_j w_j x_{ij} + b_i$$

- **affine**:
  - weighted sum of input + bias

- **activation**:
  - leaky ReLU
  - tanh

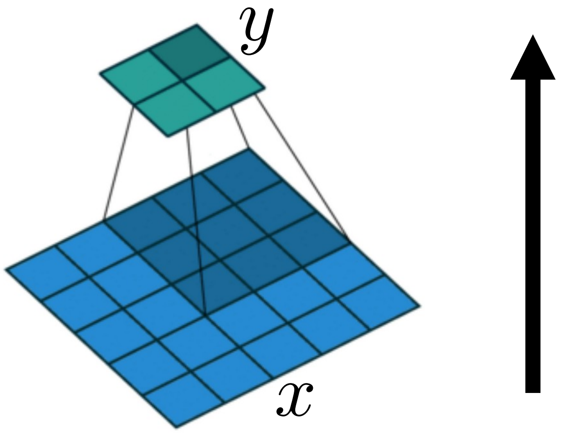


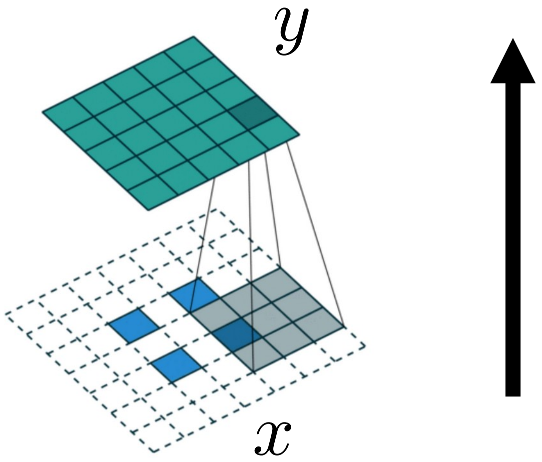
(a) (b) (c) (d)

Convolutional Layer

- convolution:
  - stride ↔ shrink
  - here: **compresses**

- transposed convolution:
  - stride ↔ grow
  - pseudo-inverse
  - here: **extrapolates**

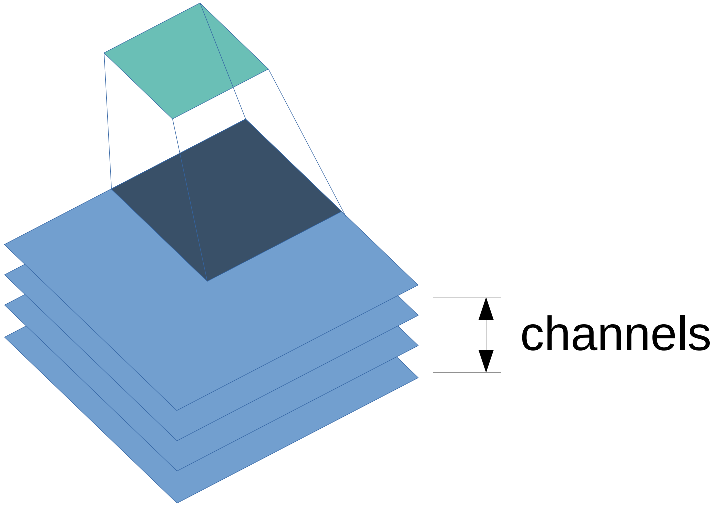




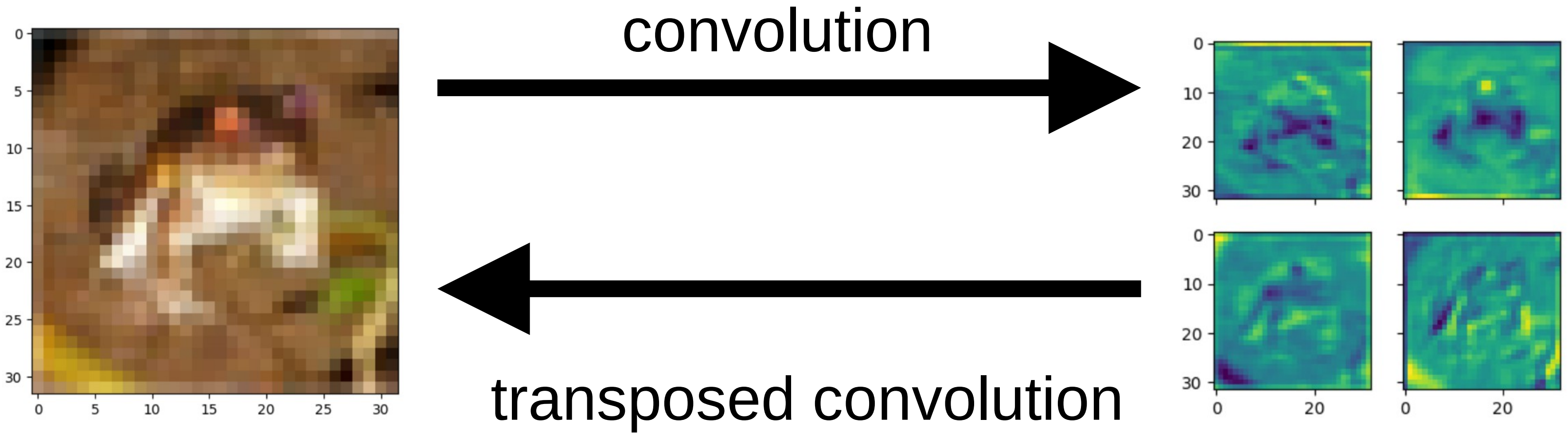
images: https://github.com/vdumoulin/conv_arithmetic

Side Note: Channels

- each pixel has multiple channels
  - e.g. Red Green Blue
- amount can change
  - e.g. convolution



ML Basics – Conv. Layer



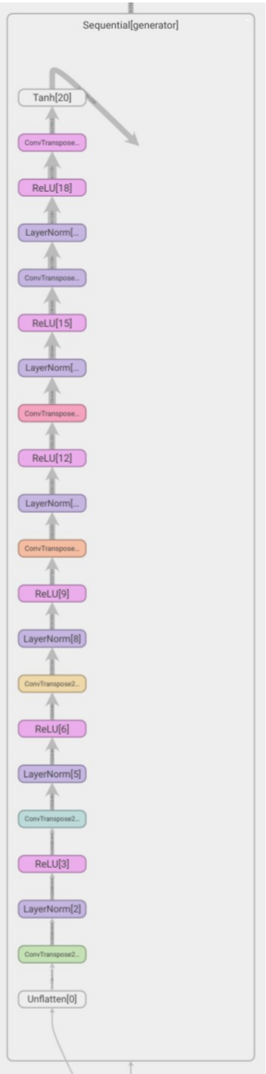
3 color/feature channels

4 out of n feature channels

image from CIFAR10

Continuation

D



Architecture Details

image_dim = 256x256
latent_vec_dim = 2048
batch_size = 128
leaky_slope = 0.2
drop_prob = 0.1
feature_map_depth = 64

GAN Losses – Wasserstein + GP

$$\mathcal{L}_C = \mathbb{E}_{x\sim\mathbb{P}_r, z\sim\mathcal{N}}[\underbrace{-C(x)}_{\text{real}} + \underbrace{C(G(z))}_{\text{fake}} + \texttt{gradientPenalty}]$$

$$\texttt{gradientPenalty} = \lambda\mathbb{E}_{\alpha\sim\mathcal{U}(0,1)}[\nabla_a\ C(a)|_{a=\alpha x+(1-\alpha)G(z)}]$$

$$\mathcal{L}_G = -\mathbb{E}_{z\sim\mathcal{N}}[\ \underbrace{C(G(z))}_{\text{fake as 'real'}}\ ]$$

$$\mathcal{L}_E = \mathbb{E}_{x\sim\mathbb{P}_r}[\texttt{MSE}(G(E(x)), x)]$$

Schlegl et al. 2019

Losses

Gradient

Mean Critic Scores

Adam GAN

---

input : γ (lr), β_1, β_2 (betas), θ_0 (params), $f(\theta)$ (objective)
λ (weight decay), *amsgrad*, *maximize*

initialize : $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment), $\widehat{v_0}^{max} \leftarrow 0$

---

for $t = 1$ **to** $\ldots$ **do**
  if *maximize* :
    $g_t \leftarrow -\nabla_\theta f_t(\theta_{t-1})$
  else
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  if $\lambda \neq 0$
    $g_t \leftarrow g_t + \lambda\theta_{t-1}$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  $\widehat{m_t} \leftarrow m_t / \big(1 - \beta_1^t\big)$
  $\widehat{v_t} \leftarrow v_t / \big(1 - \beta_2^t\big)$
  if *amsgrad*
    $\widehat{v_t}^{max} \leftarrow \max(\widehat{v_t}^{max}, \widehat{v_t})$
    $\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m_t} / \big(\sqrt{\widehat{v_t}^{max}} + \epsilon\big)$
  else
    $\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m_t} / \big(\sqrt{\widehat{v_t}} + \epsilon\big)$

---

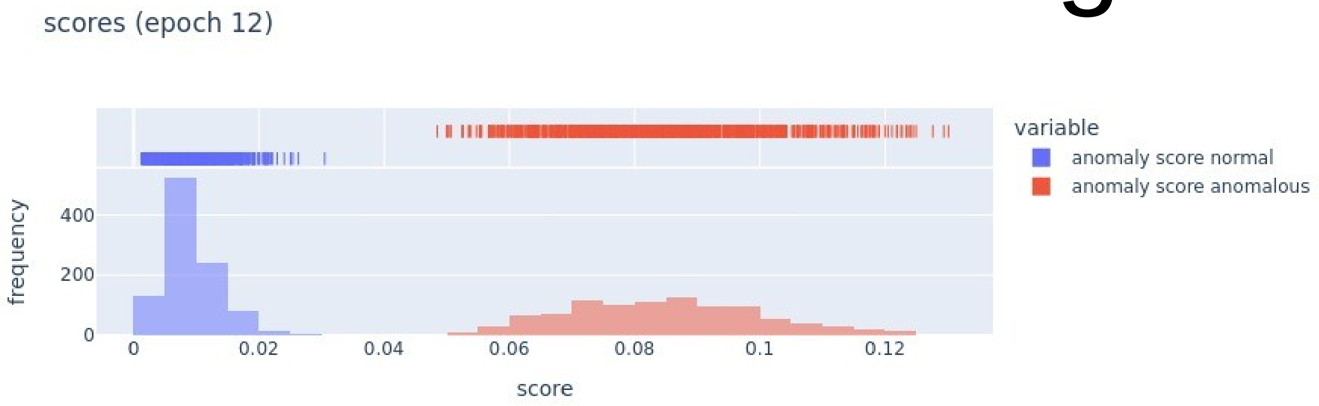
return θ_t

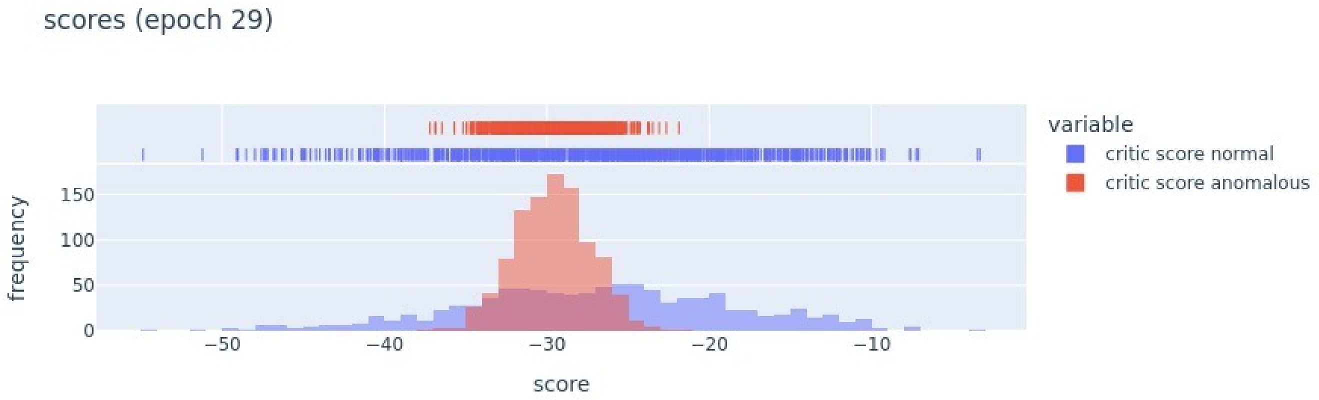
$$\gamma = 10^{-5}, \beta_1 = 0.25, \beta_2 = 0.999, \lambda = 0$$

Preprocessing

- center crop, resize & normalize images
- bundle as few files
  → reduce IO overhead
- preserve meta data
  → trace back to origin

MNIST Scores Histogram





Current State - Scores

Technical Details - Dataset

- dataset of OBACHT-0/E
  - ~20GB per crop cycle
  - ~**600GB** due to 30 croppings per image
- memory map large datasets
  → data > **RAM** (still fast due to OS caching)

Technical Details - Networks

- for images 256x256x3
  - **X M** trainable parameters
  - **VRAM** ~Y **GB**

Technical Details

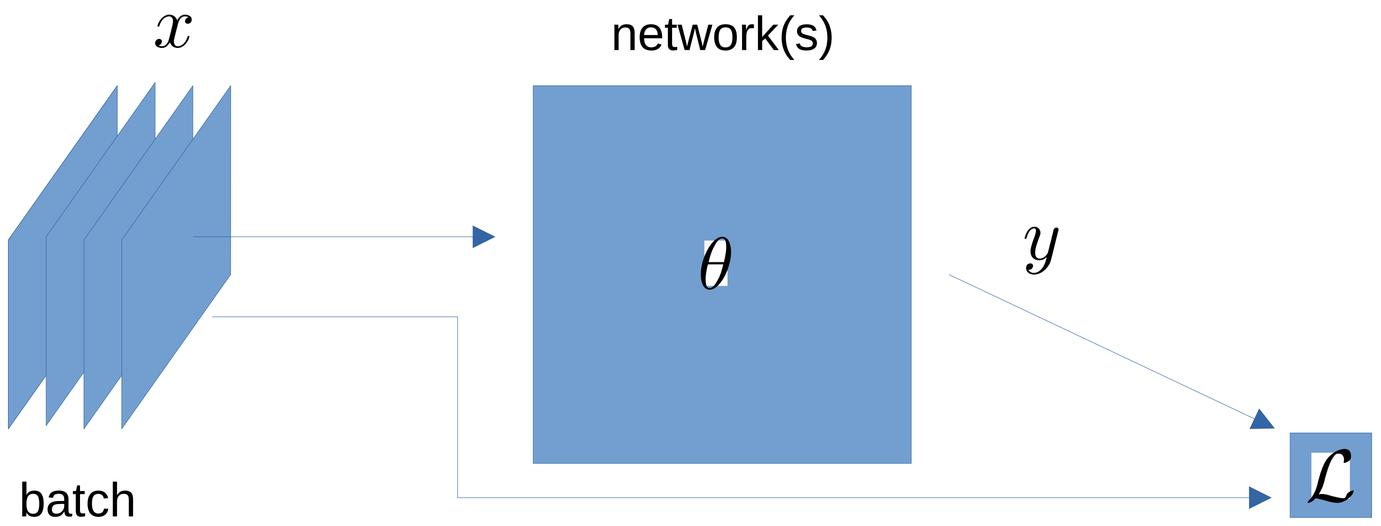
- Throughput using an NVIDIA A100 40GB
  - ~5min per 1,000 batches à 128 samples
    → 100,000 iterations take ~**8h**
  - ca. 400,000 iterations needed for generator
    → total training time in the order of few days

Some Machine Learning Basics

ML Basics – Training - Idea

- quanitify output of network(s) as loss (number)
- loss $\leftrightarrow$ ‚quality’
- incrementaly update network parameters to optimize loss (e.g. find minimum)

ML Basics – Training - Principle



$$g = \frac{d\mathcal{L}}{d\theta} \; : \; \text{gradient}$$

$$\theta \leftarrow \theta + \text{optim}(\nabla g)$$

Training Algorithm - Principle

per iteration:

- randomly sample a batch of images
- apply networks
- calculate mean loss + gradient
- update network parameters

ML Basics – Special Layers

- normalization:
  - batch
  - layer
- drop

→ resilience

Latent Space by OBACHT batch