A Low-cost Strategic Monitoring Approach for Scalable and Interpretable Error Detection in Deep Neural Networks

Florian Geissler[1], Syed Sha Qutub[1], Michael Paulitsch[1], and Karthik Pattabiraman[2]

[1] Intel Labs, Germany
[2] University of British Columbia, Vancouver, Canada

intel

SafeComp 2023, Toulouse, France

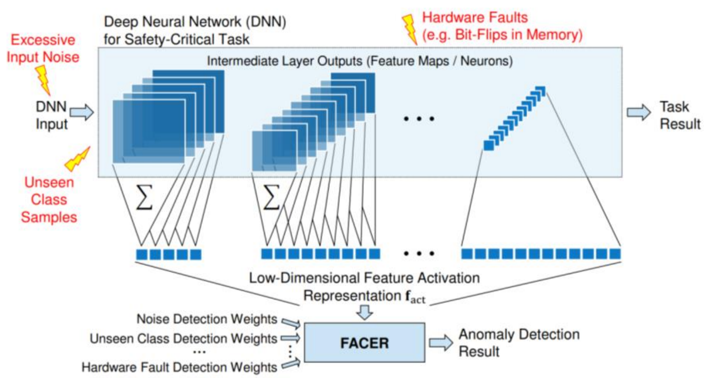
Problem Statement

TP: True positives (green), **FP**: False positives (orange), **FN**: False negatives (blue)

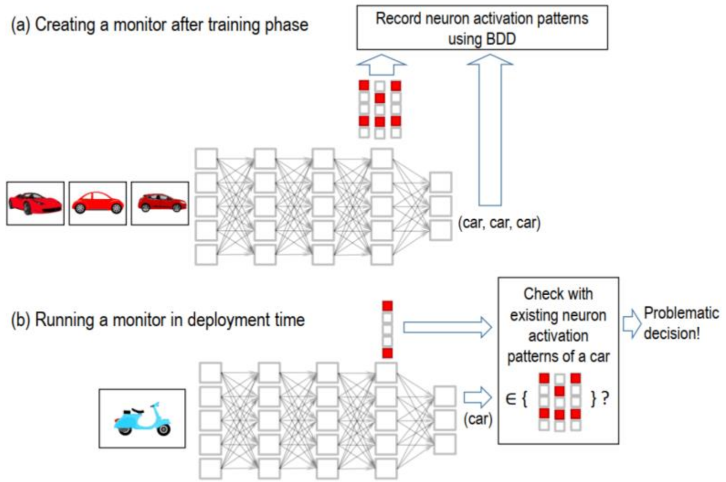
- Complex DNNs are known to be sensitive to silent data corruption (SDC) under specific faults (noise, hardware, etc.)
- Need to protect the DNN at runtime against diverse critical faults (while ignoring non-critical faults)
- Concept: Monitor intermediate activations to classify error patterns
- Interpret the patterns to find best correction method and increment user trust

State of the Art

[1] Cheng et al., 2018
[2] Ahuja et al., 2019
[3] Chen et al., 2020
[4] Hoang et al., 2019
[5] Schorn et al., 2018
[6] Schorn et al., 2020
[7] Huang et al., 2018



Schorn et al, 2018, 2020



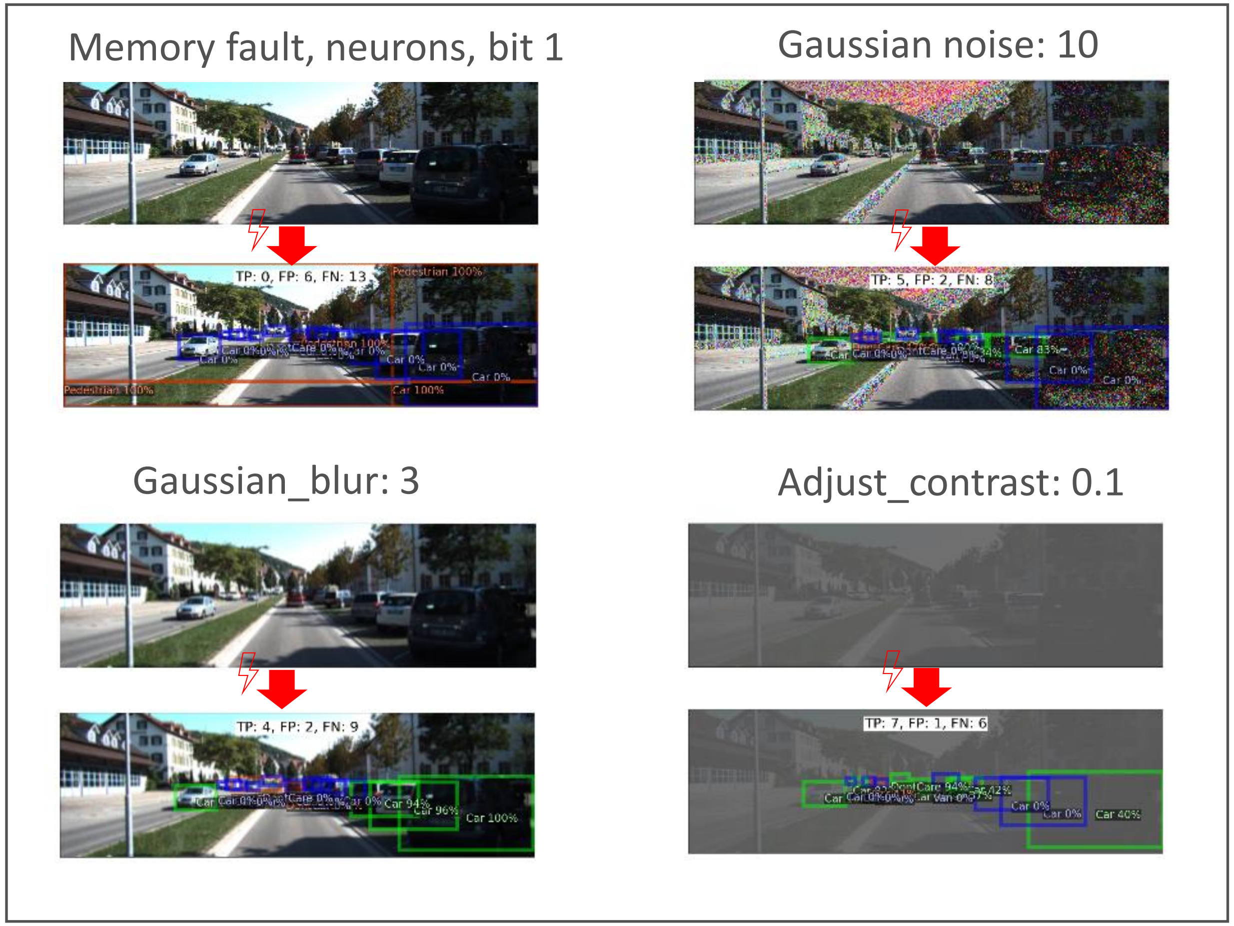
Cheng et al, 2018

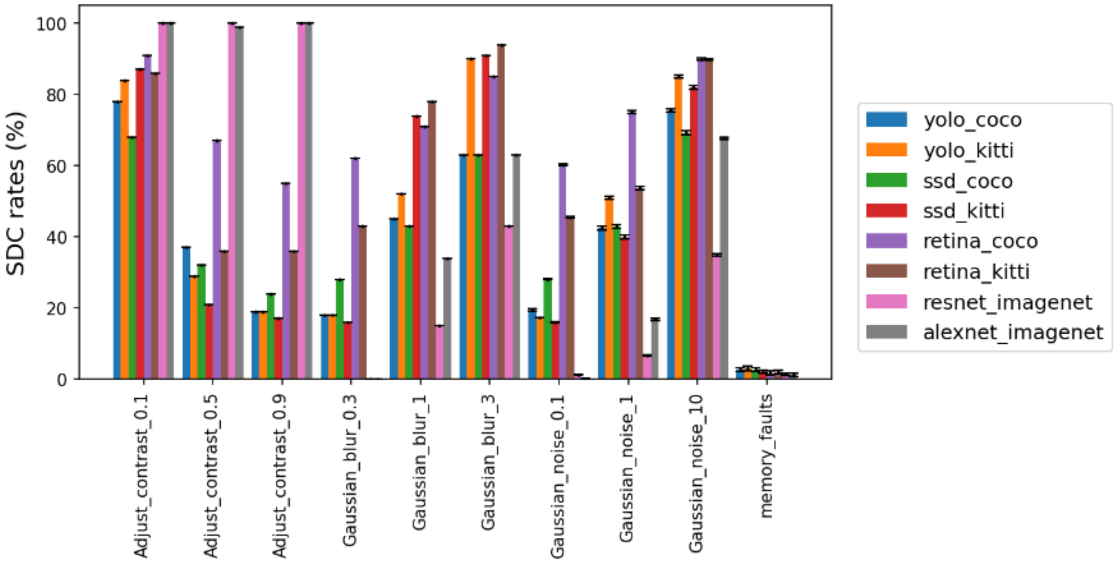
<u>Goals of our method:</u>

<u>Current methods have at least one of the following shortcomings:</u>

- Use methods that are limited to single-label problems (e.g., associate error pattern with specific outcome class) [1,2] — Universally applicable
- Use global thresholds for error classification that omit more subtle faults [3,4] — Capture subtle and outlier fault
- Have in-transparent detector methods so that error patterns are not interpretable [5,6] — Explainability
- Only address a single fault mode [7] — Address diverse fault patterns with a single detector
- Require significant compute or memory overhead [2, 5,6] — Efficient in compute and memory

Failure Modes and Models





- 3 input faults with 3 different magnitudes each + memory faults (average neurons and weights) = **10** fault modes
- SDC rate depends on model and dataset (**8** different computer vision setups tested)
- Calibrate experiments to get equal statistical samples

intel.

Example: Yolo + Kitti

Our key observation

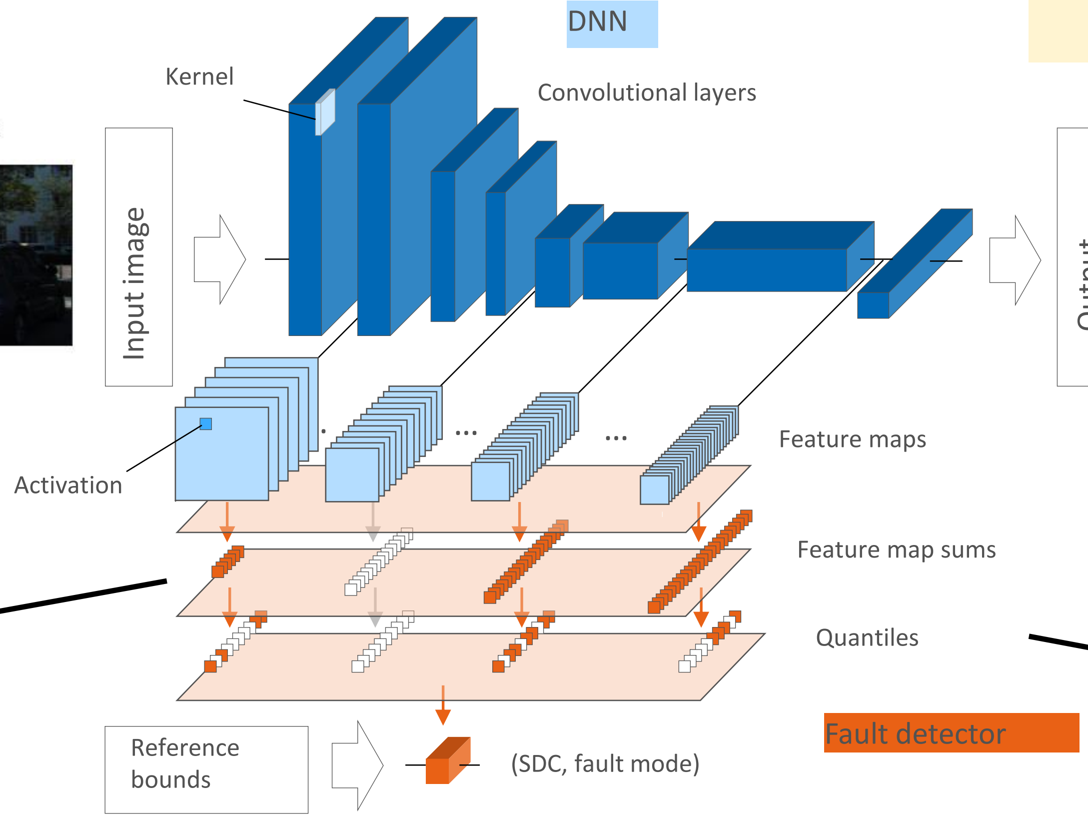
When SDC occurs, the topology of the feature maps in the affected layer typically changes as:

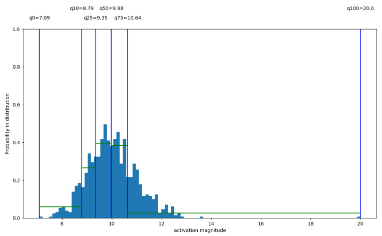
1. A few individual values get changed a lot = **peak shift** (seen for <u>memory faults</u>, large deviations in all subsequent layers)
2. Many (all) values get changed slightly = **bulk shift** (seen for <u>input faults</u>, can become large deviations in subsequent layers)

→ Quantile markers capture **both** effects in a unified way

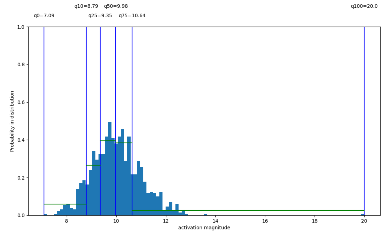
The method: Quantile Extraction

Note: Quantile extraction will
- Capture both peak and bulk shifts in a unified way
- Massively reduce the number of features to handle (and monitor)





Example fmap sums: distribution of ~**28K** values

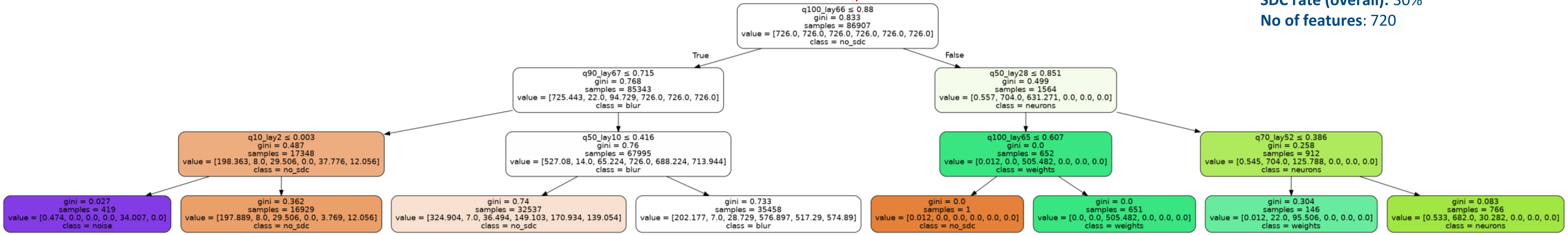


Example Quantiles: **720** values (only ~10 check points to represent one layer)

Detector - Decision Tree

Model: Yolov3
Data: Coco (100 images)
Epochs: 100
Fault model: Gaussian Blur, Gaussian Noise, Adjust Contrast, Neurons, Weights
Fault mode: 1 fault per image
SDC rate (overall): 30%
No of features: 720

Decision node inspecting q100_lay66



Decision tree graph (depth=3)

Prediction

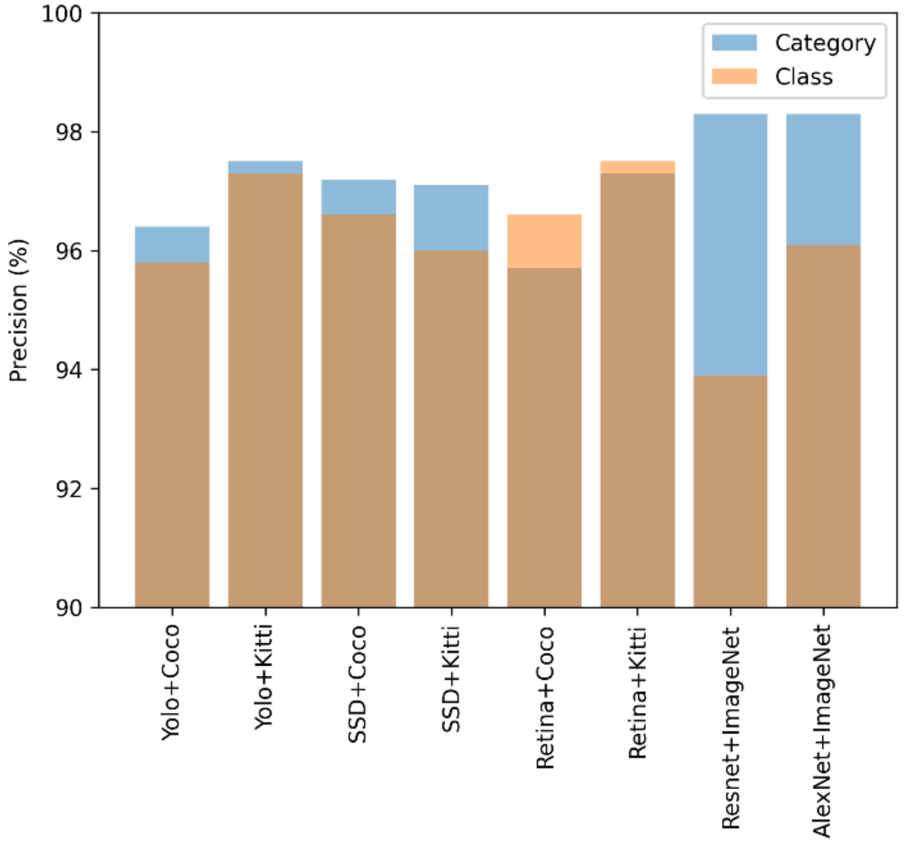
Ground truth

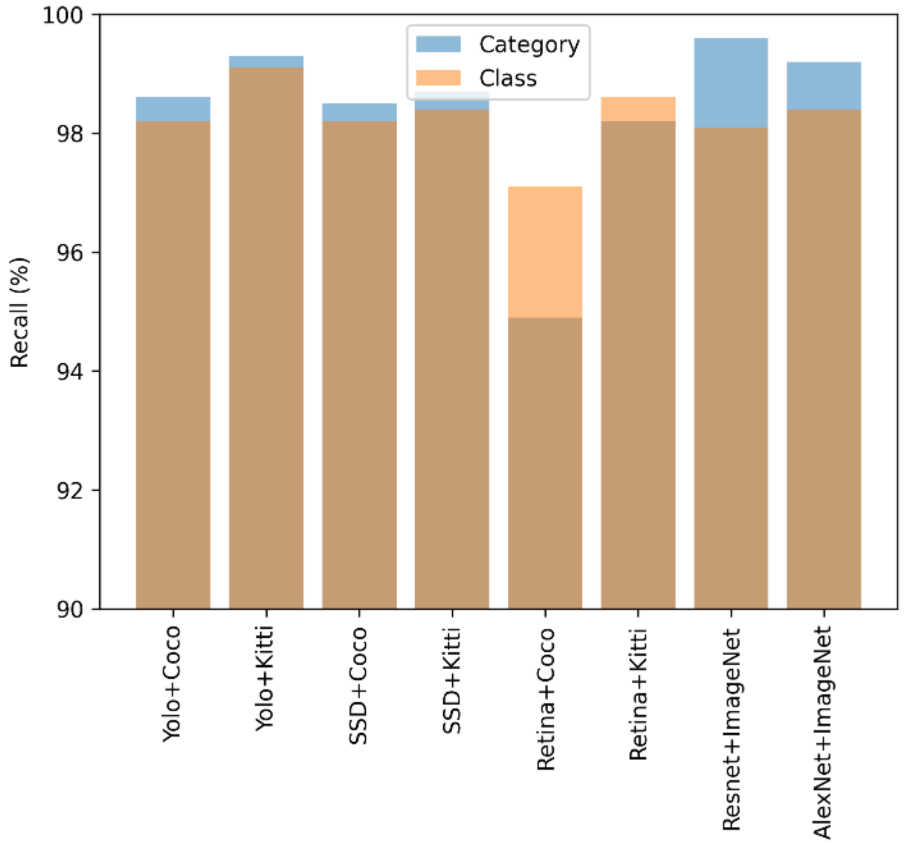
Real\Predicted		No SDC	SDC				
		No SDC	Neurons	Weights	Blur	Noise	Contrast
No SDC	No SDC	**26790**	140	52	2950	85	0
SDC	Neurons	12	**330**	6	2	0	0
	Weights	17	7	**448**	2	0	0
	Blur	0	0	0	**3534**	0	317
	Noise	37	2	0	2848	**1235**	0
	Contrast	0	0	0	3314	0	**677**

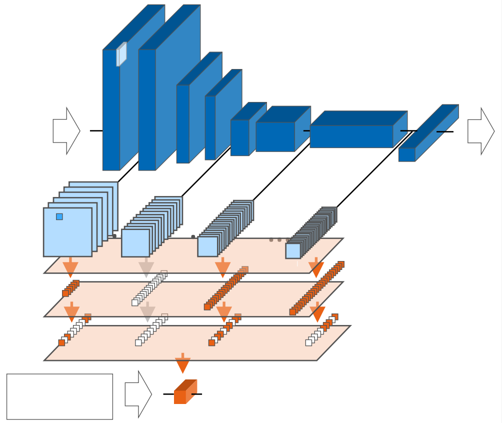
Tree confusion matrix (depth=10)

- **Decisions** based on quantile values are fully **transparent**
- Error classification improves with **tree depth**
- Most confusions happen **within input fault classes or input faults vs no SDC**
- Error detection success **metric** can be varied depending on use case

Results – Precision and Recall

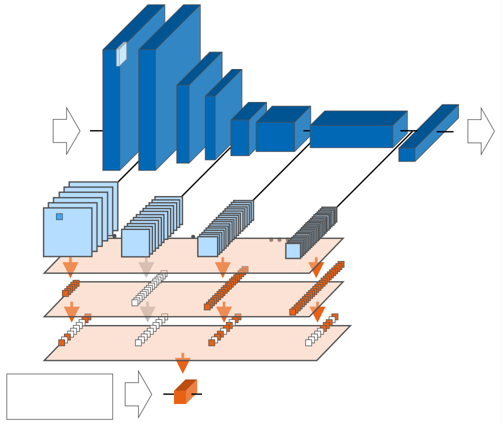


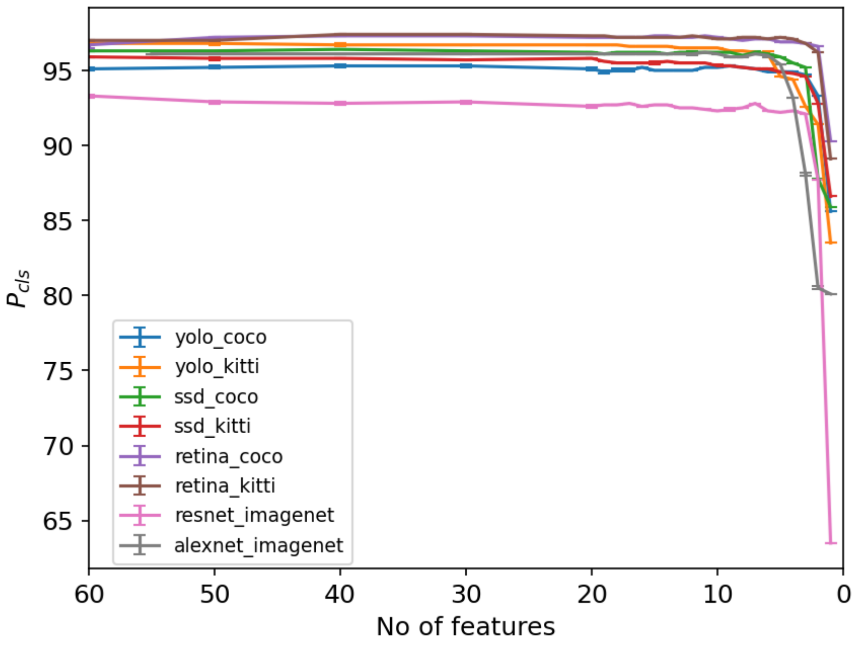


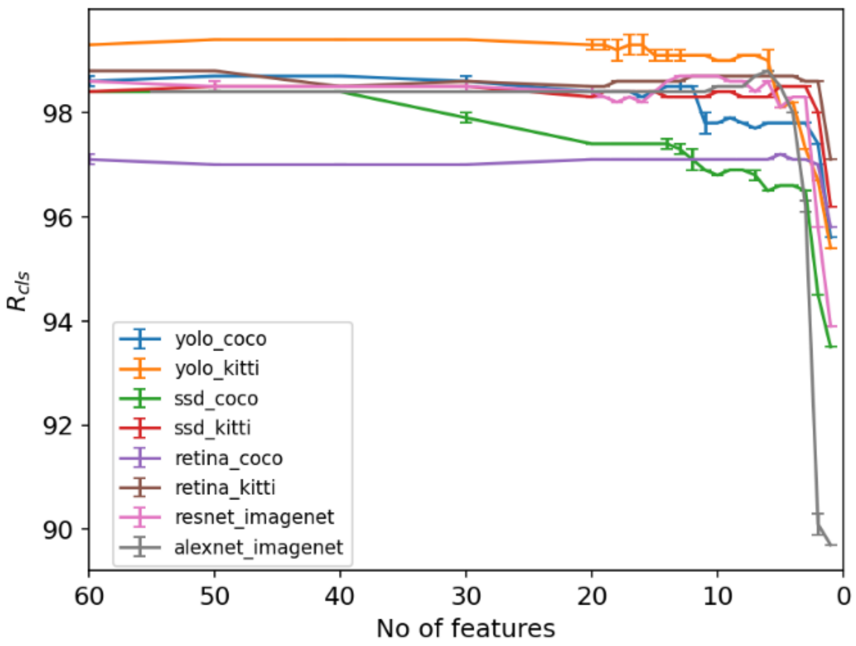


- Results are averages across all failure modes, per model
- We find **P >= 94%, R>97% for class-wise metric** (all confusions except exact class get penalized)
- We find **P > 95%, R>=95% for category-wise metric** (confusions within the same fault category *input/memory/no_sdc* do not get penalized)

Results - Feature Reduction

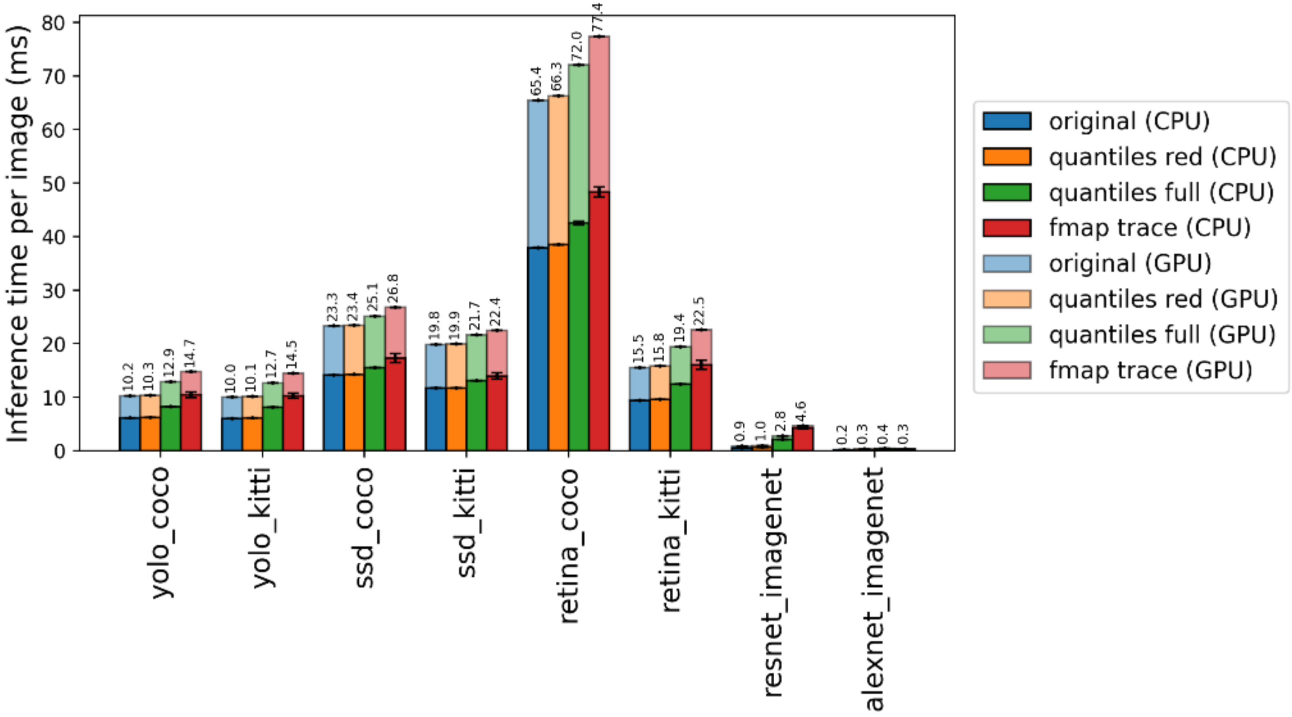






- We can **reduce** the number of monitored quantiles and layers significantly and still get very good results for precision/recall
- In most cases, 2-3 quantiles from **2-3 layers** are enough to reach **>95%** of the original performance (with all quantiles and layers) → Save a lot of compute and memory!
- **Quantile markers**: Typically, a strategic marker in a late layer and one in the first half of the DNN works best.

Results - Overhead



Model	N_{act}	N_{fm}	N_{ft} (full)	$\min(N_{ft})$ (red)
Yolo+Coco	39.2M	27.1K	825	2
Yolo+Kitti	38.4M	26.4K	825	2
SSD+Coco	75.4M	11.6K	429	2
SSD+Kitti	73.7M	9.1K	429	2
RetinaNet+Coco	296.6M	31.4K	781	2
RetinaNet+Kitti	69.7M	30.8K	781	2
ResNet+Imagenet	2.4M	26.6K	583	2
AlexNet+Imagenet	82.4K	1.2K	55	4

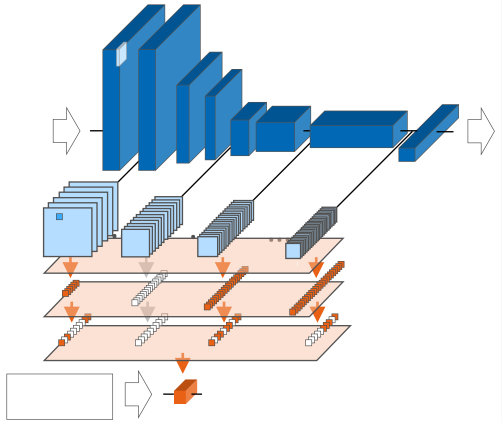
68-9445 x

21-46 x

13-412 x

- **Quantile monitoring is faster than feature map tracing**: Additional quantile operation but do not need to store large tensors
- Reducing to minimal model saves more computation. **Only 0.3%-1.6% inference time overhead** for object detection models
- **Information compression ratio**: Quantile operation compresses data by a factor of >20 x, feature reduction by another factor of >10-400 x.

Summary

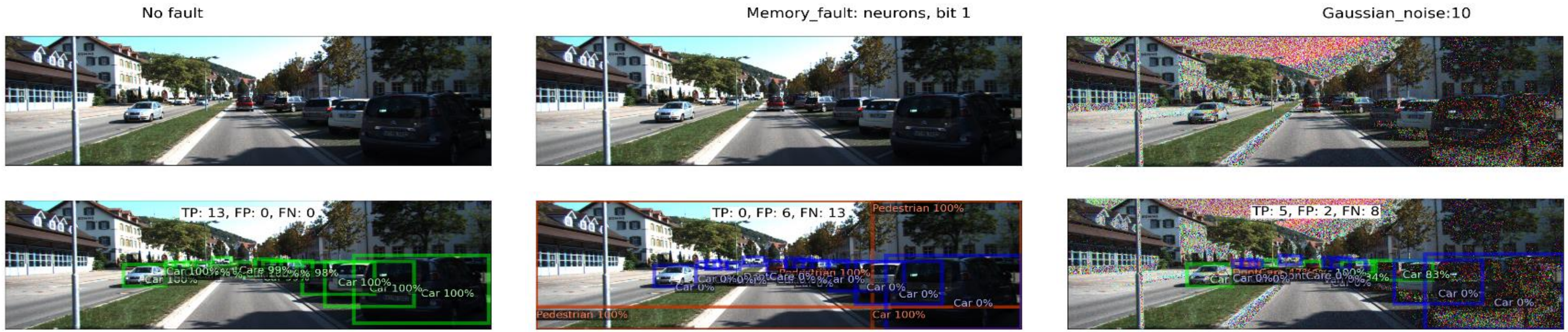


- First method to address **both hardware faults and input faults in a unified way**
- Input and memory faults are associated with **bulk and peak activation shifts**, giving a unifying perspective on the dependability of DNNs
- Even for complex object detection networks, **errors can efficiently be detected** (P up to ~97%, R up to ~98%) even with quantile shifts in **only a few layers** (down to 2 layers)
- Method is **low-cost,** as high information compression incurs only low overhead (down to ~+0.3% in inference time)
- We identify **minimal sets of relevant features** for monitoring across models
- Detection with **algorithmically transparent components** such as decision trees

intel®

Problem statement

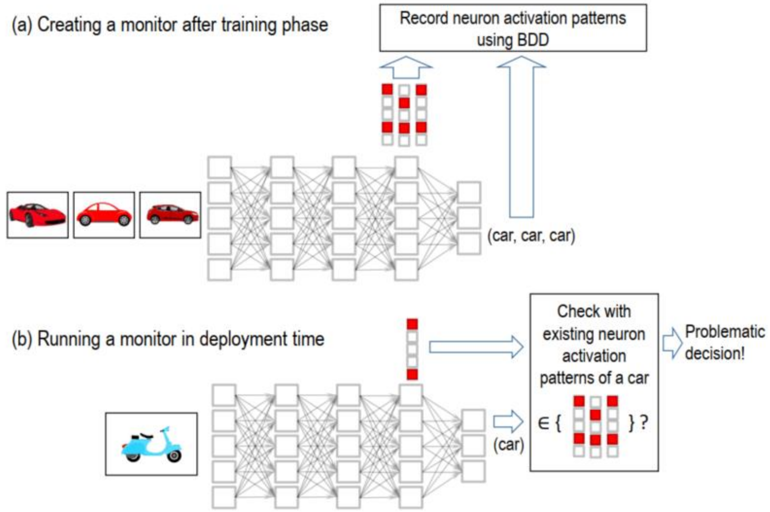
- Even complex DNNs are known to have robustness issues under specific faults (noise, hardware faults, etc.).
- Most critical are silent data corruption (SDC) errors!



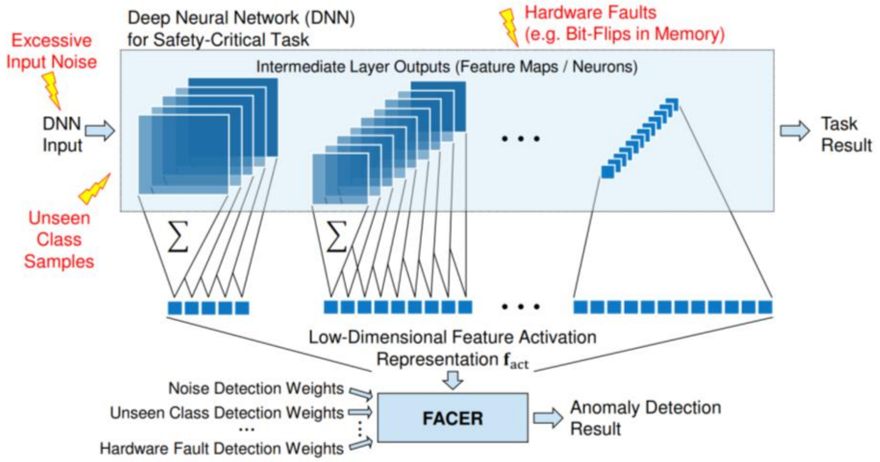
TP: True positives, **FP**: False positives, **FN**: False negatives

- Goal: Error detection in two steps: **1)** Monitor activation patterns, **2)** Anomaly detection.
- Challenges: Design DNN error detectors that are
  - Efficient in performance and memory footprint
  - Can reliably identify SDC, and differentiate fault modes
  - Transparent to foster model explainability

State of the art - examples



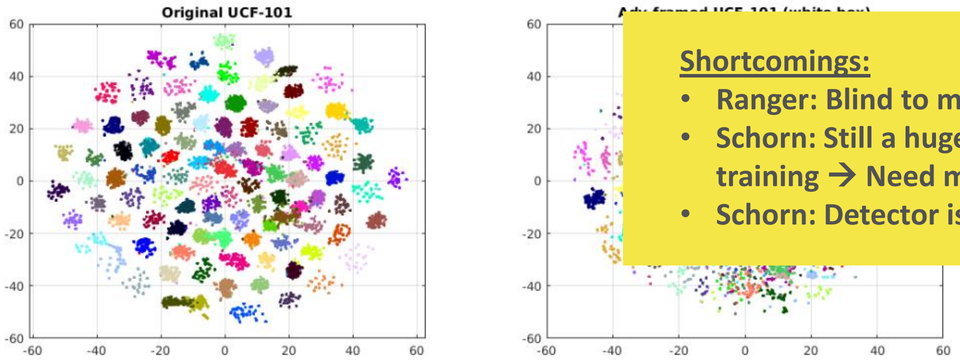
Cheng et al, 2018: class activation vectors



Schorn et al, 2018, 2020: FACER (Feature activation consistency checker)

Shortcomings:
Assumes discrete outcome classes to form cluster patterns, does not generalize to any ML problem.

!



Shortcomings:
- Ranger: Blind to more subtle faults below maximum.
- Schorn: Still a huge amount of features to be extracted, used for classifier training → Need more efficient monitoring
- Schorn: Detector is again a black box → no transparency

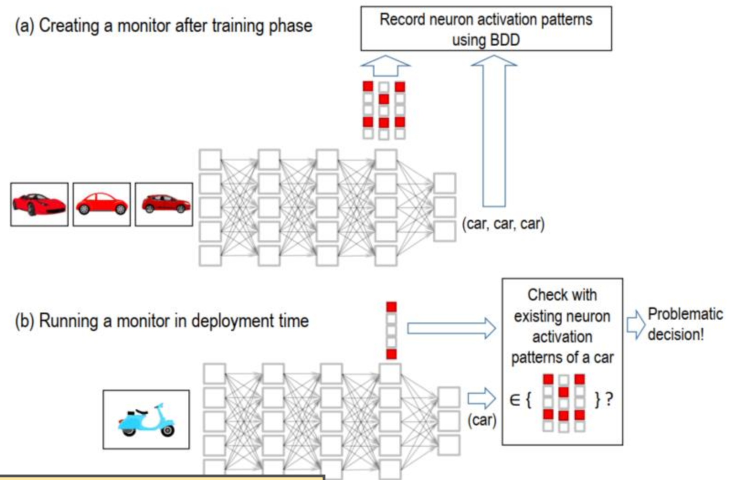
!

Figure 4: t-SNE visualization of (spatio-temporal) feature embeddings for UCF101 using C3D Resnet101 (Layer 1).

Ahuja et al, 2019: Deep feature modeling

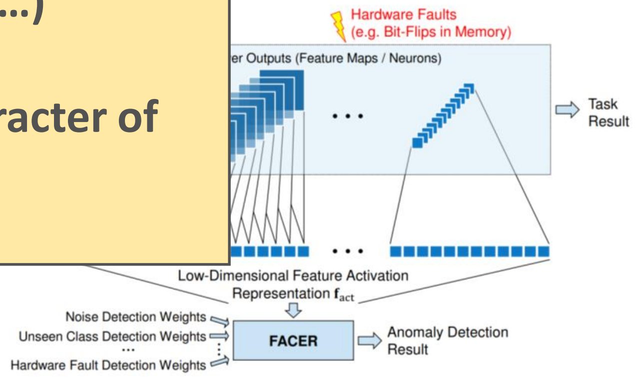
State of the Art

Category	Example methods	Shortcomings (for our goal)
Associate observed activation patterns with class outcome clusters	Class activation vectors [1], deep feature modeling [2]	Only for single-label problem. What about multi-label scenarios like object detection?
Compare observed activation patterns to predefined global rule		
Detect errors from ob activation pattern wit attached secondary n		
Image-level error dete		



et al, 2018

<u>We see need for better methods that achieve all of the following:</u>
- **Highly accurate SDC detection**
- **Discrimination of different types of faults (input, memory faults, …)**
- **High efficiency in compute and memory footprint**
- **Algorithmic transparency in the detector, to tackle black-box character of DNN**



Schorn et al, 2018, 2020

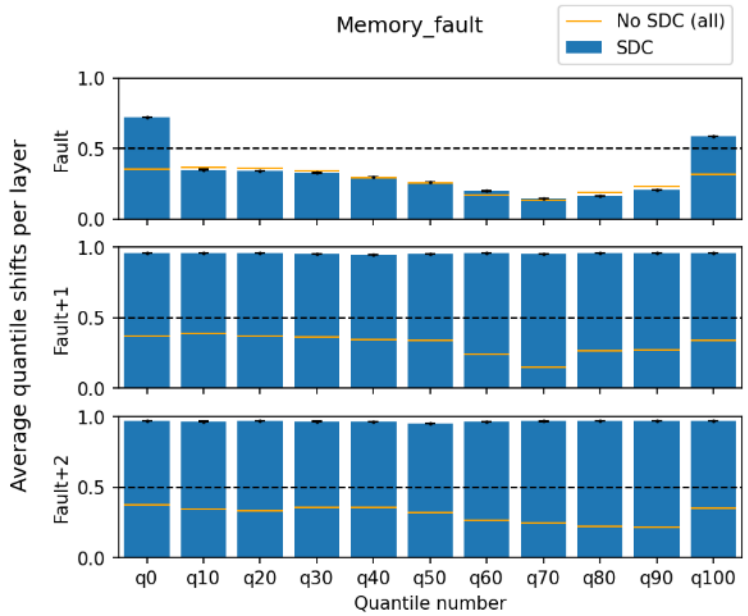
[1] Cheng et al., 2018
[2] Ahuja et al., 2019
[3] Chen et al., 2020
[4] Hoang et al., 2019
[5] Schorn et al., 2018
[6] Schorn et al., 2020
[7] Huang et al., 2018

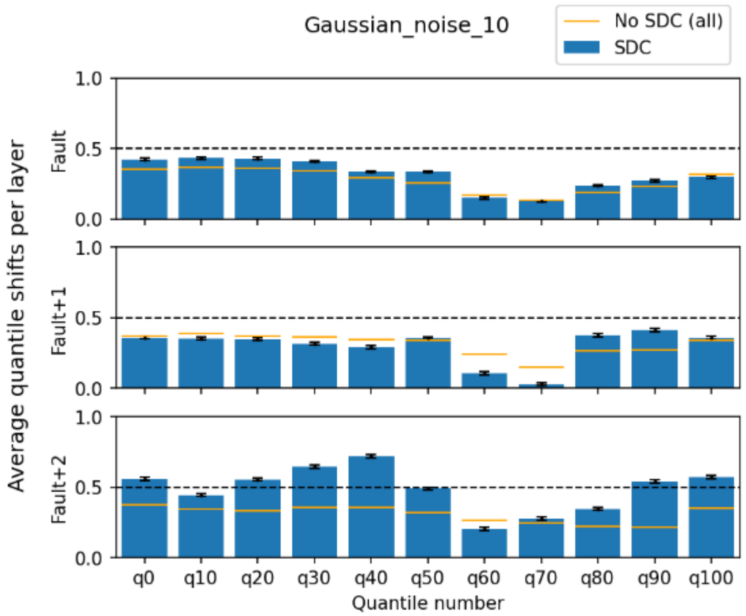
Quantile Shifts

Network layer (conv)

Quantile marker (10 percentiles)

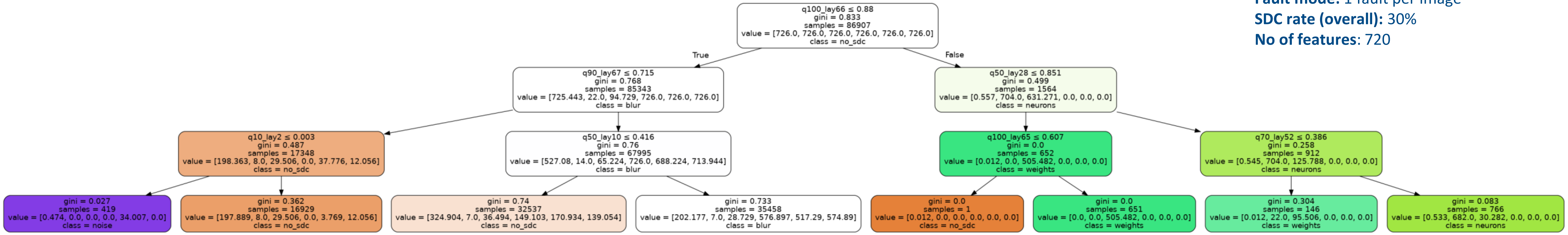
- Full quantile vector to find error patterns is $\vec{q} = [q_0^1, q_0^2, \dots q_0^L, q_{10}^1, \dots q_{100}^L]$
- Gets normalized to range (0,1) with **1**: large out of bound values (pos or neg), **0.5**: approx. the bounds, **0**: within bounds and close to lower bound
- Confirm intuition in affected and following layers:
  - Memory fault → min/max quantiles out-of-bound (= **peak shift**), escalates to all quantiles quickly
  - Input faults → All quantiles changed slightly and in-bound (= **bulk shift**), escalates slowly towards out of bound quantiles





Detector - Decision Tree

Model: Yolov3
Data: Coco (100 images)
Epochs: 100
Fault model: Gaussian Blur, Gaussian Noise, Adjust Contrast, Neurons, Weights
Fault mode: 1 fault per image
SDC rate (overall): 30%
No of features: 720



Decision tree graph (depth=3)

Tree Depth	P (SDC/NoSDC)	R (SDC/NoSDC)
3	0.31	0.60
5	0.44	0.95
10	0.66	0.99
No limit	0.98	0.99

Tree precision and recall

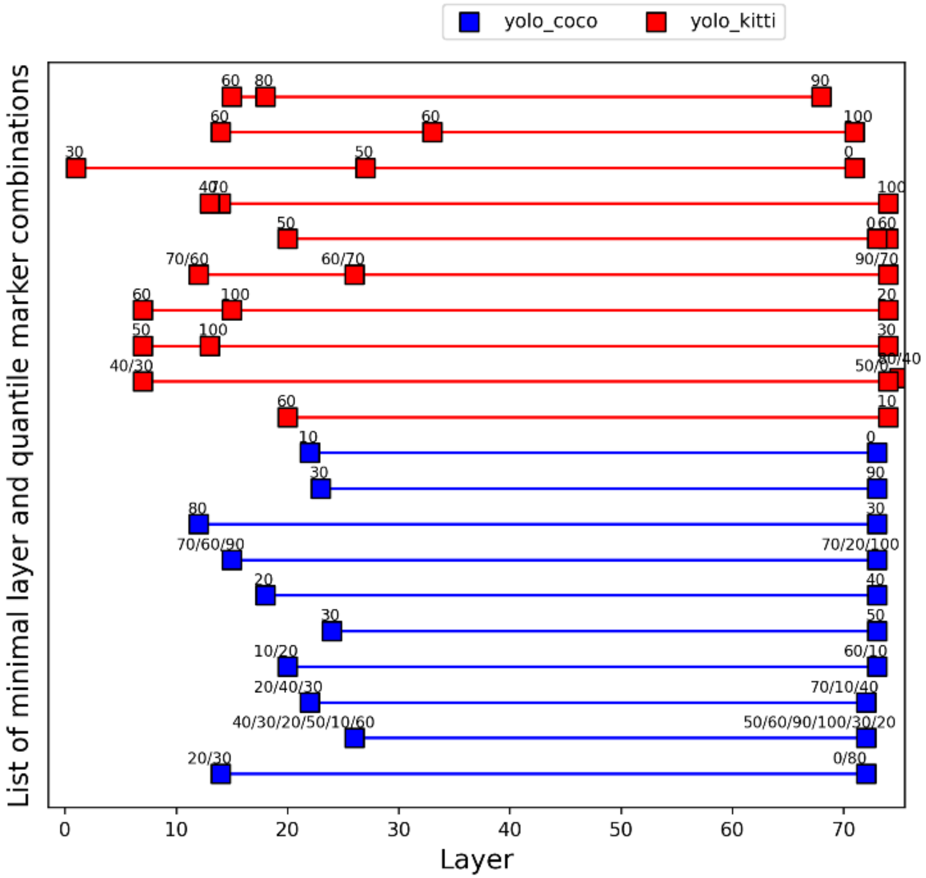
Prediction

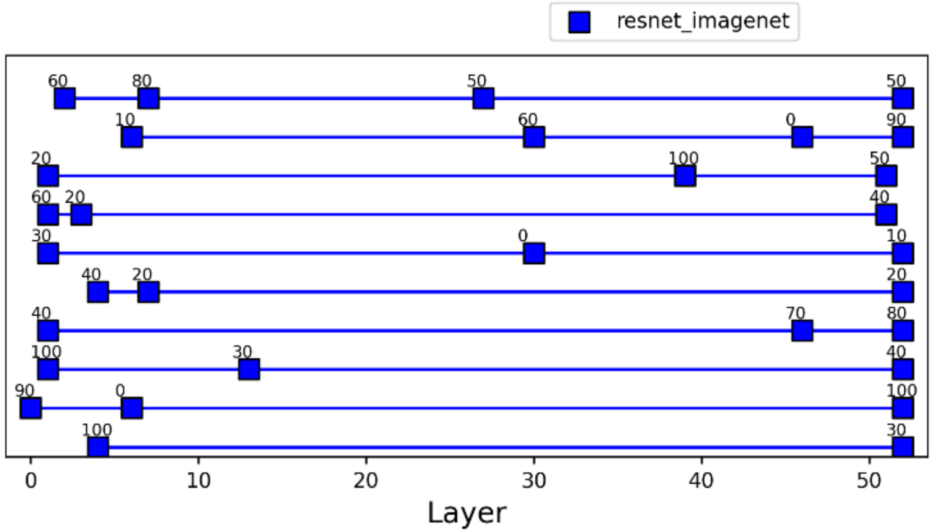
Ground truth	Real\Predicted	No SDC	SDC				
		No SDC	Neurons	Weights	Blur	Noise	Contrast
No SDC	No SDC	**26790**	140	52	2950	85	0
SDC	Neurons	12	**330**	6	2	0	0
	Weights	17	7	**448**	2	0	0
	Blur	0	0	0	**3534**	0	317
	Noise	37	2	0	2848	**1235**	0
	Contrast	0	0	0	3314	0	**677**

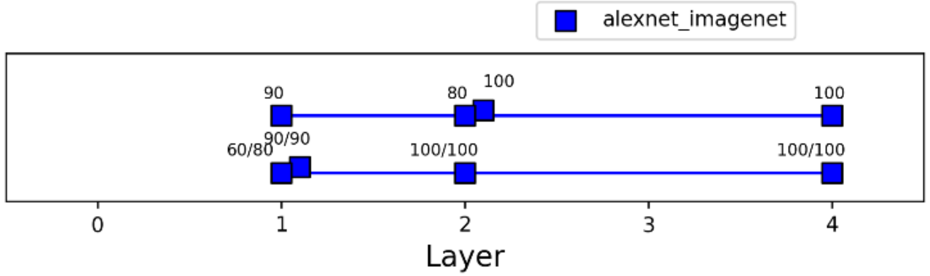
Tree confusion matrix (depth=10)

intel.

Feature reduction







Results

Legend:
P: Precision
R: Recall
Cls, cat, sdc: detector metrics for class-wise, category-wise (input/memory), or sdc-only classificaiton
Nft: Number of monitored features (quantiles x layers)
Nl: Number of monitored layers
Full: detector model using all features
Red (avg): Reduced detector model (averaged)

- 3 input faults with 3 different magnitudes each + memory faults (average neurons and weights) = **10** fault modes
- SDC rate depends on model and dataset (**8** different computer vision setups tested)
- Calibrate experiments to get equal statistical samples

Model	**P(%)**			**R(%)**			**DT**
	P_{cls}	P_{cat}	P_{sdc}	R_{cls}	R_{cat}	R_{sdc}	N_{ft}/N_{l}
Yolo+Coco							
full	95.8	96.4	96.1	98.2	98.6	98.4	825/75
red (avg)	93.3	94.6	93.4	97.4	96.3	96.7	2/2
Yolo+Kitti							
full	97.3	97.5	97.4	**99.1**	99.3	99.2	825/75
red (avg)	92.6	92.1	92.0	97.3	96.4	96.8	3/2
SSD+Coco							
full	96.6	97.2	96.6	98.2	98.5	98.3	429/39
red (avg)	95.2	96.3	94.9	96.5	94.5	95.9	3/3
SSD+Kitti							
full	96.0	97.1	96.2	98.4	98.7	98.6	429/39
red (avg)	92.8	94.6	92.1	98.0	97.7	98.2	2/2
RetinaNet+Coco							
full	96.6	95.7	96.9	97.1	94.9	98.0	781/71
red (avg)	**96.6**	96.6	96.5	97.0	94.6	98.2	2/2
RetinaNet+Kitti							
full	**97.5**	97.3	97.5	98.6	98.2	98.7	781/71
red (avg)	96.2	96.6	95.9	**98.6**	97.8	98.9	2/2
ResNet+Imagenet							
full	93.9	**98.3**	**97.6**	98.1	**99.6**	**99.4**	583/53
red (avg)	92.1	**97.6**	**96.7**	98.3	**99.6**	**99.5**	3/3
AlexNet+Imagenet							
full	96.1	**98.3**	97.3	98.4	99.2	99.0	55/5
red (avg)	93.2	96.8	95.0	98.0	99.0	98.8	4/3

Quantiles (I)

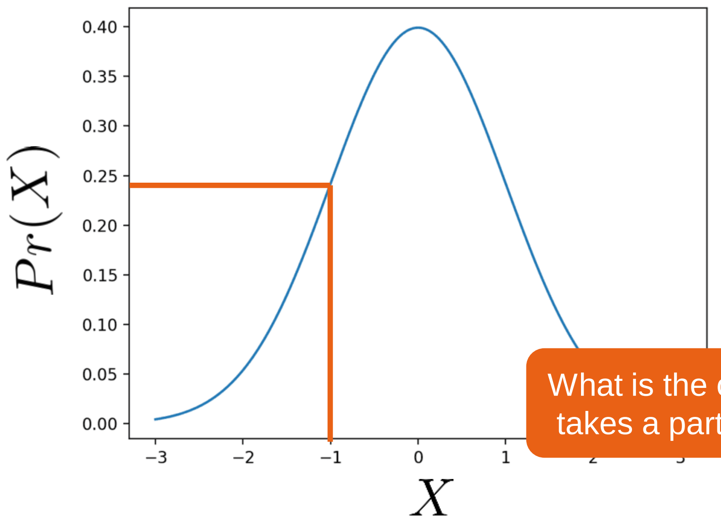
- The quantile function* is the **inverse** cumulative distribution function**, $Q = F^{-1}$

$$F_X(x) = Pr(X \leq x) = p$$
$$Q(p) = F_X^{-1}(p) = x$$

- „**Quantiles**“ are discrete evaluations of the quantile function (Q(p) = “p-quantile”)

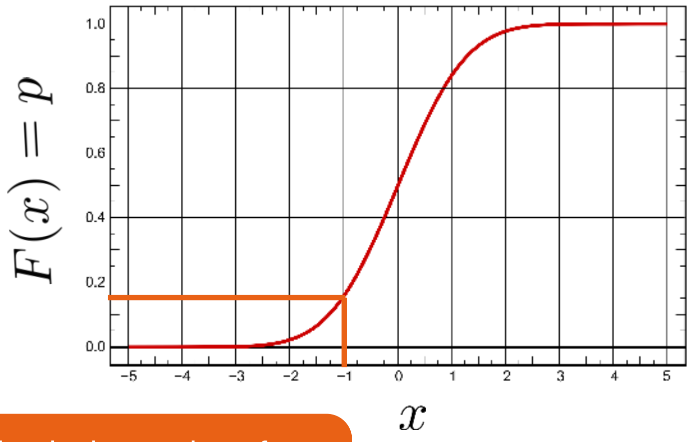
- Quantiles can discretize information about how a variable is distributed

Probability density function (PDF)

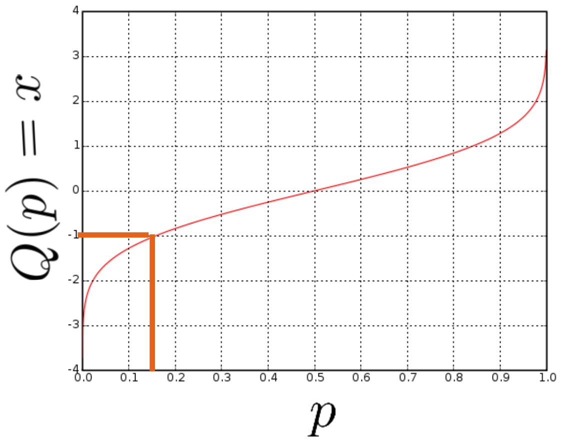


What is the chance that X takes a particular value?

Cumulative distribution function (CDF)



Quantile function



What is the portion of outcomes where X takes any value below a theshold of x?

Given a chance p, what is the corresponding threshold x so that p is the chance that X is below x.

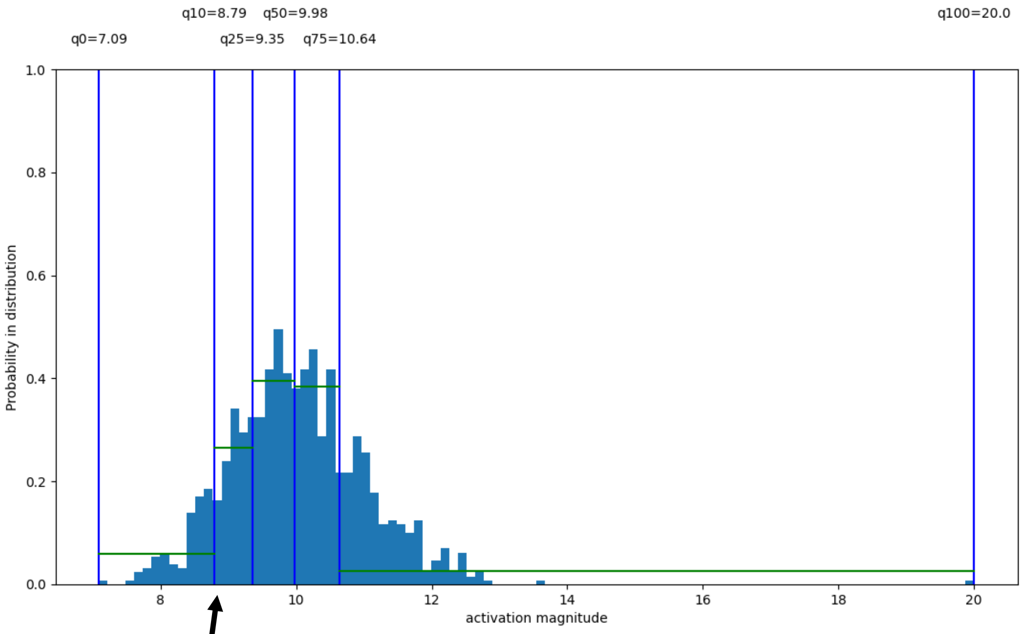
X: random
*associate
variable
**assumption: continuous variables and functions

Quantiles (II)

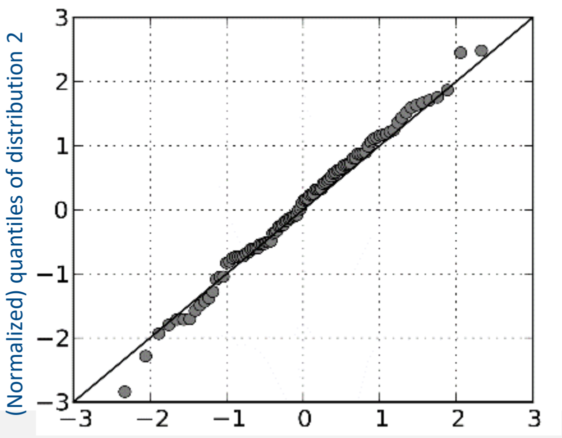
- Quantiles can discretize information about how the variable is distributed
- We can use the discretization to reconstruct an estimate distribution

$$Pr(q_m \leq X \leq q_n) \approx \frac{n-m}{q_n - q_m}$$

- Can be used to compare similarity of two distributions, e.g. Q-Q-Plot



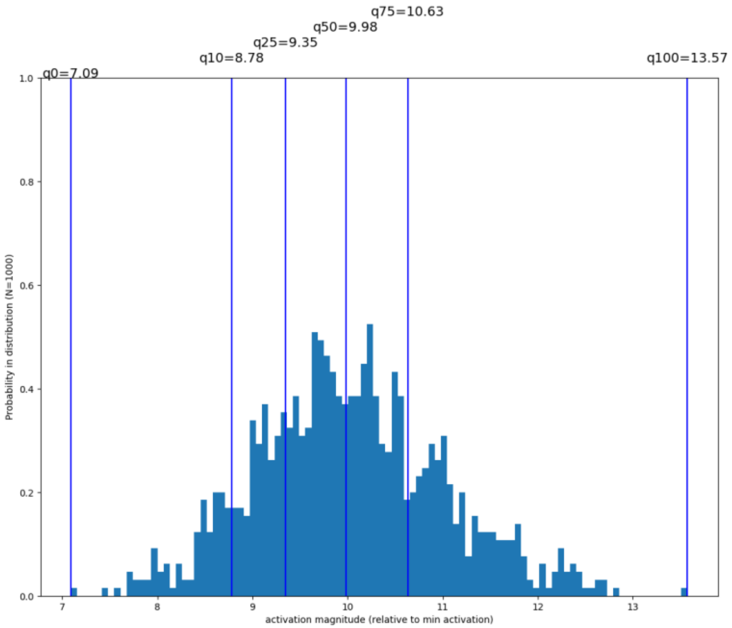
10% quantile: 8.79 is the threshold so that10% of all data points are below that



Note:
Q0 = minimum
Q50 = median
Q100 = maximum

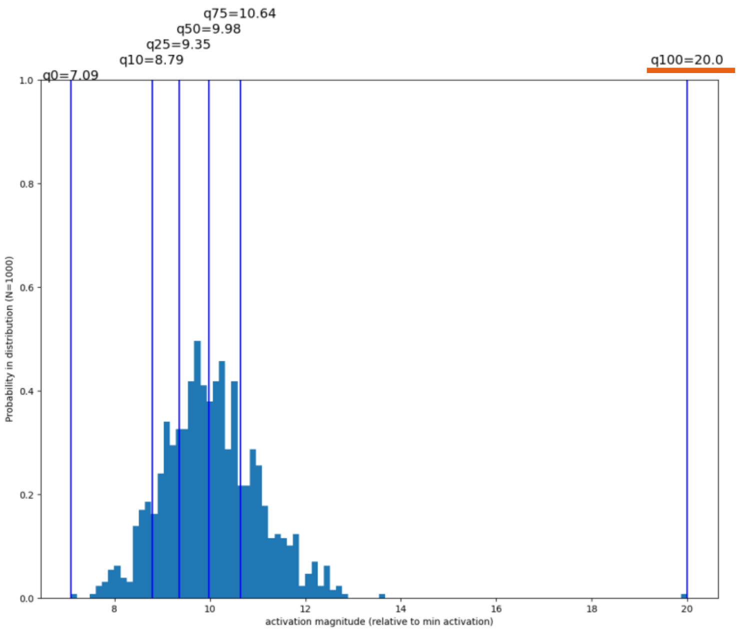
Generic example:

- 1000 random samples
- drawn from Gaussian distribution with μ=10, σ=1



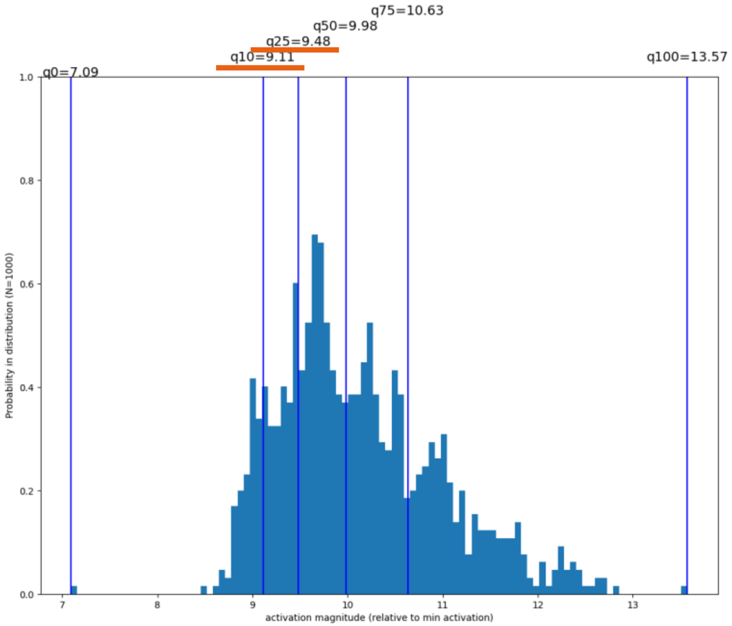
Peak shift

(e.g. here 1 element → 20)



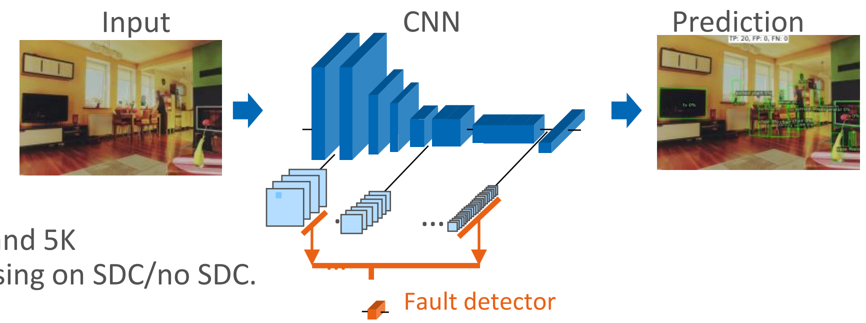
Bulk shift

(e.g. here first 100 elements +1)



SDC?

Findings



- Performance
  - Detector in example setup achieves about **P:95%, R:95%** with fault trees, **P:90%, R:90%** with LR units and 5K training samples per fault mode. The confusion rate was found to be <3% when only focusing on SDC/no SDC.
- Efficiency
  - Feature space is significantly reduced (~400 features in Yolov3) compared to Schorn approach with fmap sums (~26Kfeatures). That also means that much less data is required to train the detector network. Fcc approach gives low P,R for given data.
  - Quantile monitoring is slower than activation sum (~10x), but can be compensated by above.
  - It appears that only the supervision of very few layers (~5 for Yolov3 from >70) is sufficient to achieve decent performance (P~80%, R~95%). This could be used to hook only some selected „**symptom layers**“.
- Transparency
  - Detector is inherently transparent ML component: Human can understand decision based on symptoms
  - Acquire understanding about fault patterns in different parts of the network, e.g. emphasis on later layers
- Novelty
  - Use new way to condense features in much smaller network/tree for better efficiency in inference and detector training. Can be only specific layers.
  - Transparent monitoring, i.e. reasoning for fault detection is traceable and can be interpreted by a human
  - More generalized use case demonstrated (object detection). Method is architecture-independent.