Anomaly Detection on MVTec2D: PatchCore - A density-based method approach

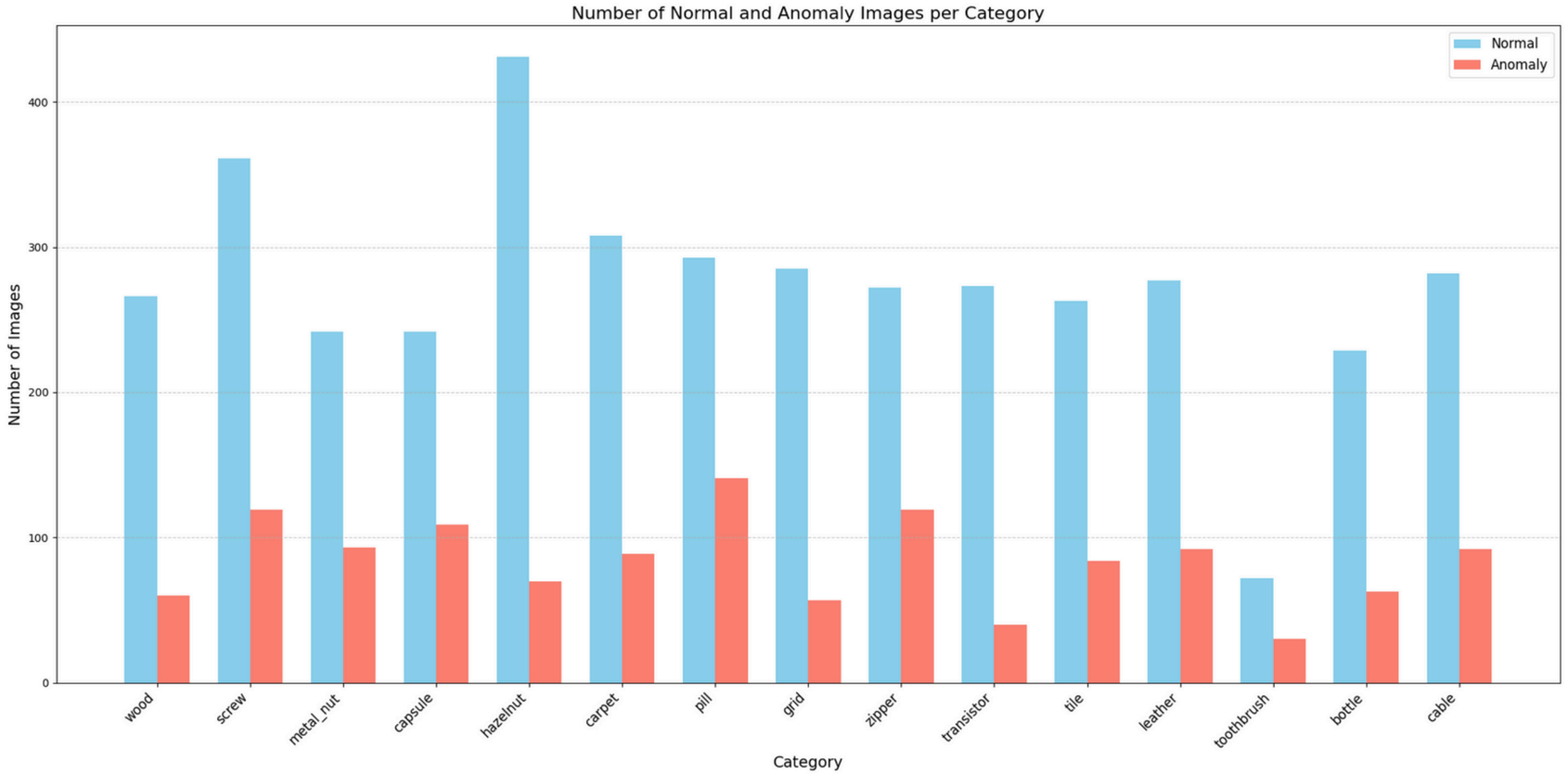
Ly Nguyen Thuy Linh Tran Huu Loc Tran Tuan Khoa

University of Information Technology

16th December 2024

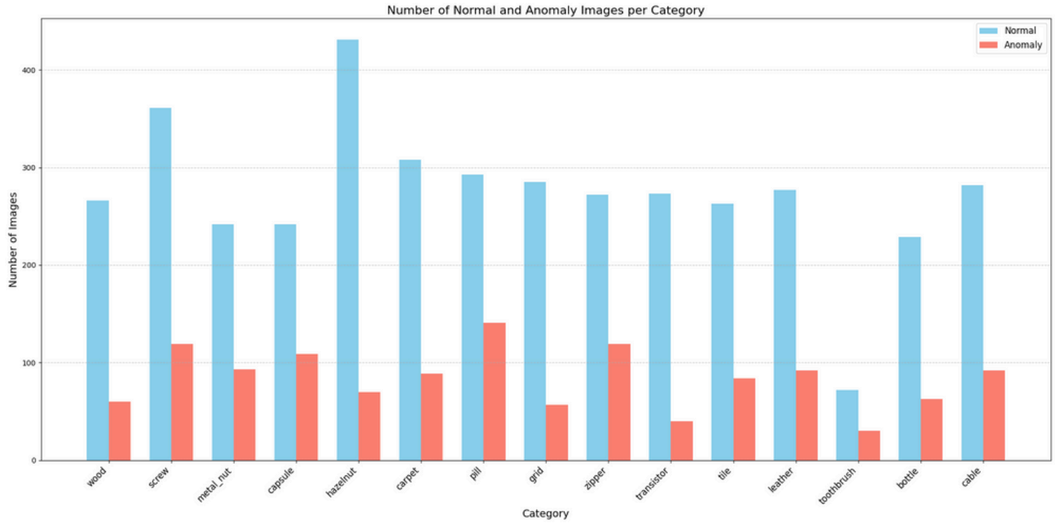
Anomaly Detection Scenario

- Assume that it is practically impossible to sufficiently secure abnormal data, so they learn only from normal data.

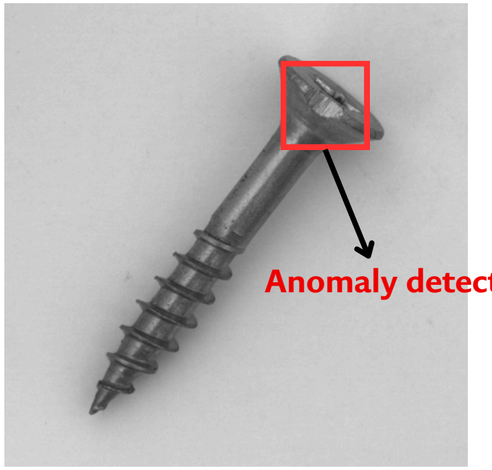


Anomaly Detection Scenario

- Assume that it is practically impossible to sufficiently secure abnormal data, so they learn only from normal data.
- Based on the characteristics of normal data, when a different result occurs, it is detected as abnormal data.

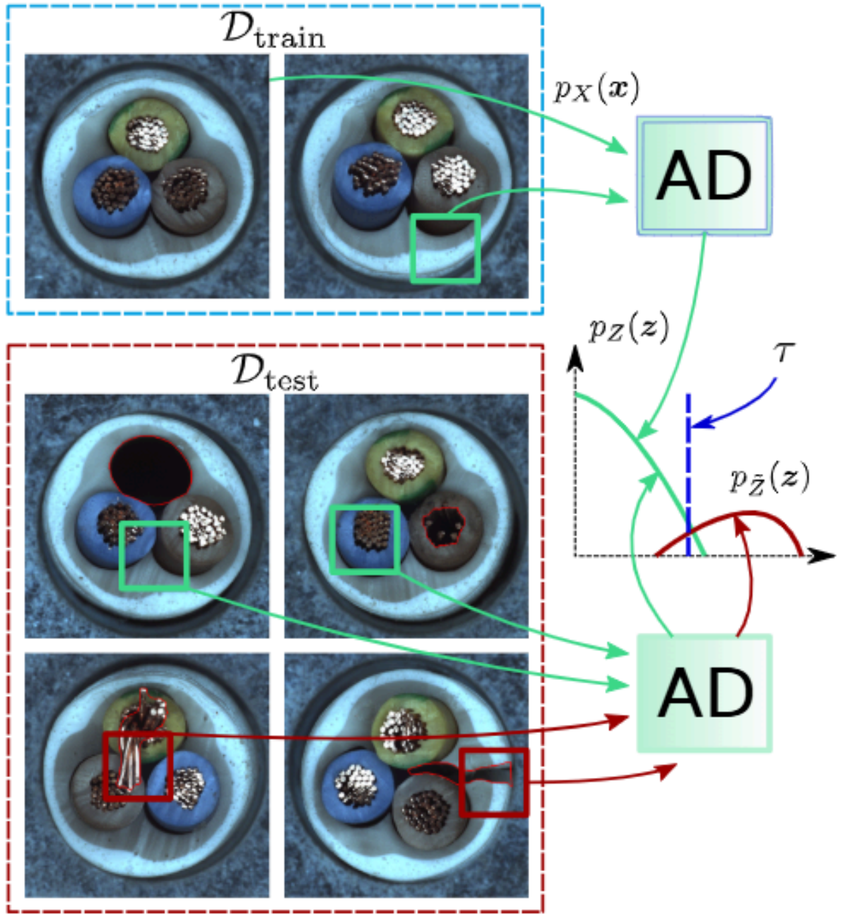


Normal images as training data



Anomaly images as testing data

Anomaly Detection Scenario

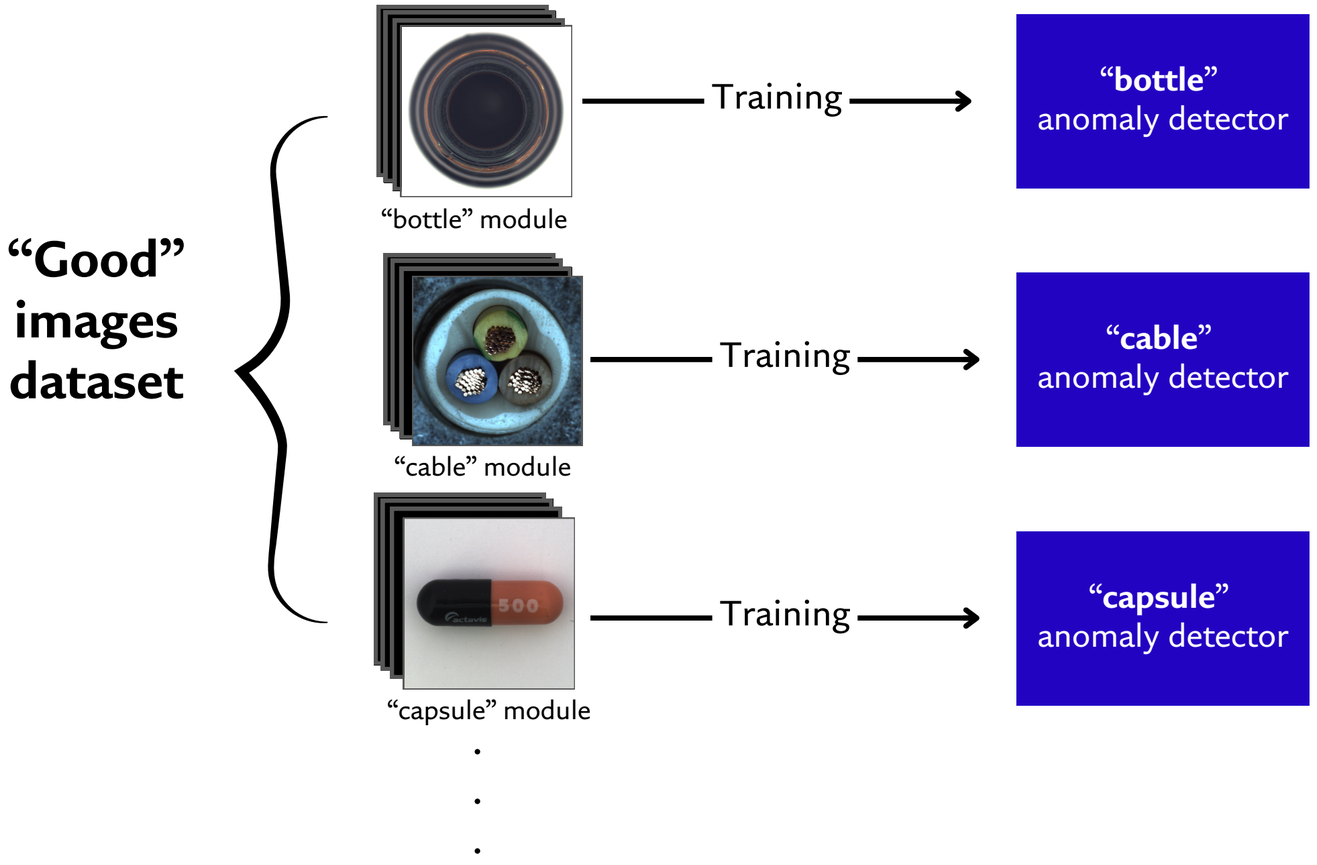


- The Anomaly Detector learns the distribution of normal images and transforms it into a Gaussian distribution.

CFLOW-AD: Real-Time Unsupervised Anomaly Detection with Localization via Conditional Normalizing Flows

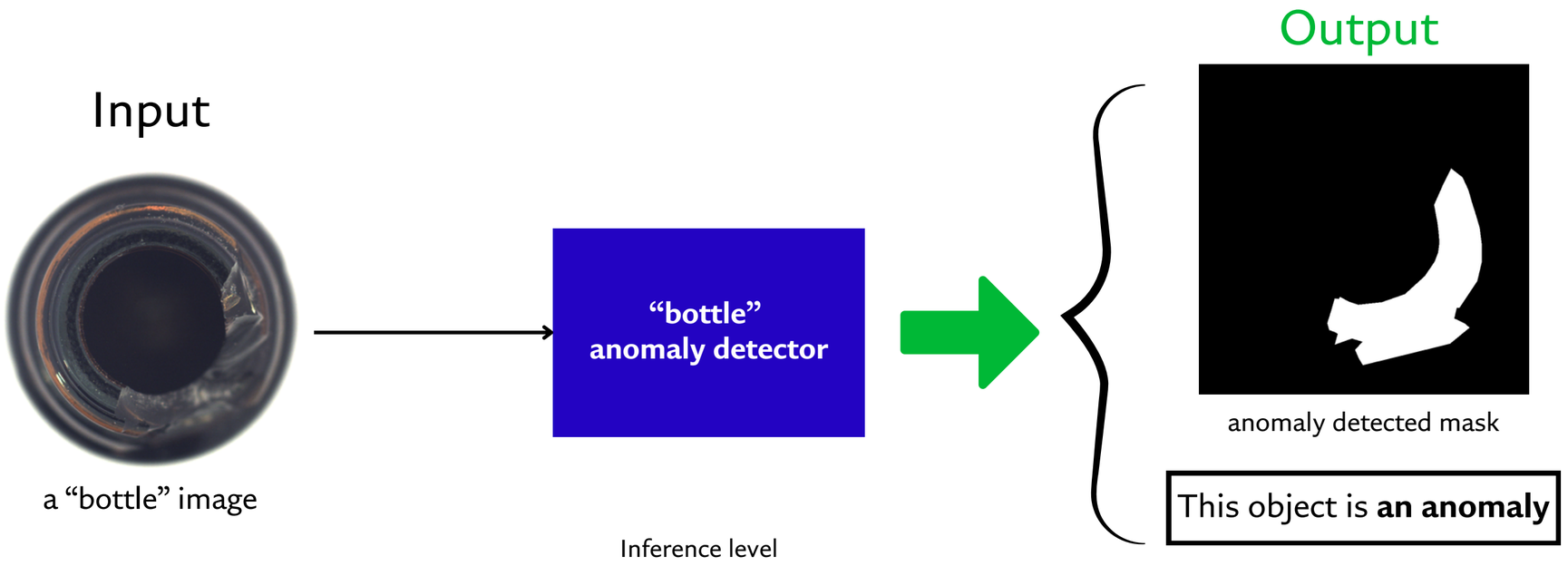
Problem Statement

- Training level.
  - **Input:** MVTec **modules** including **industrial objects/textures.**
  - **Output:** The trained modules corresponding with the data module input.



Problem Statement

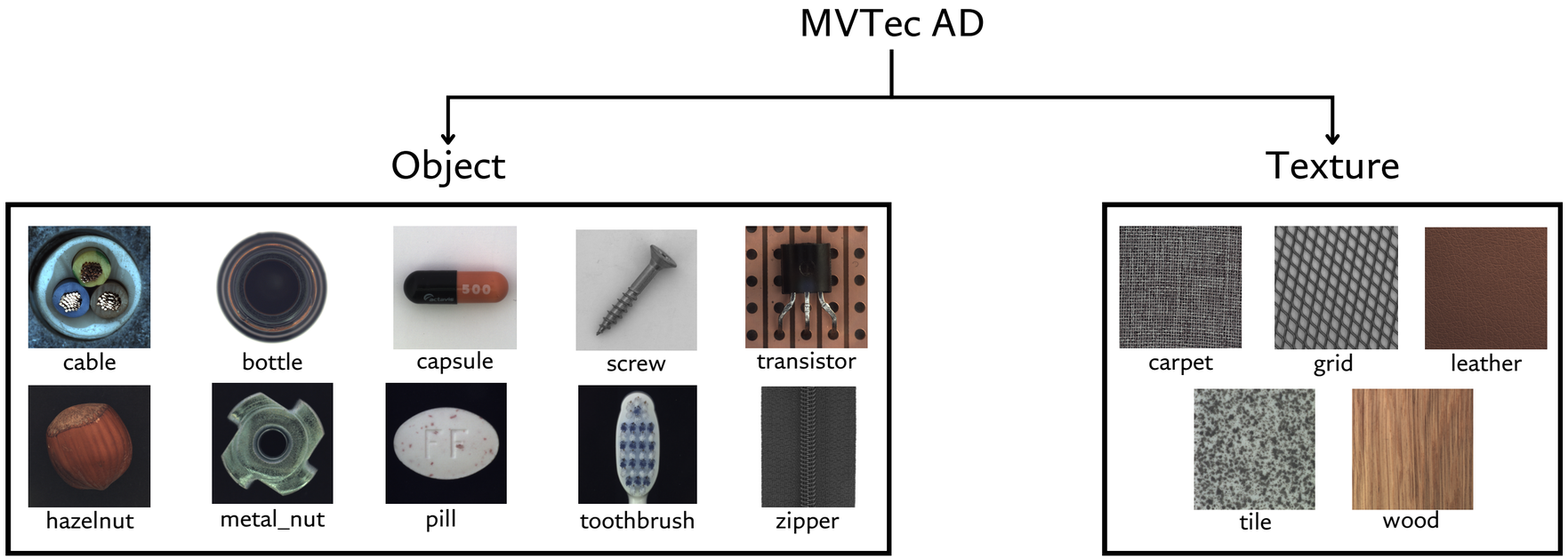
- Training level.
  - **Input:** MVTec **modules** including **industrial objects/textures.**
  - **Output:** The trained modules corresponding with the data module input.
- Inference level.
  - **Input:** an object/texture images from a module that has been trained.
  - **Output:** a mask showing the anomaly position and classification result.



Inference level

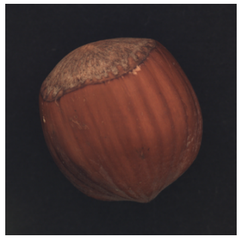
MVTec AD Dataset

- bottle
  - ground_truth
  - test
    - broken_large
    - broken_small
    - contamination
    - good
  - train
  - license.txt
  - readme.txt
- cable
- capsule



MVTec AD Dataset

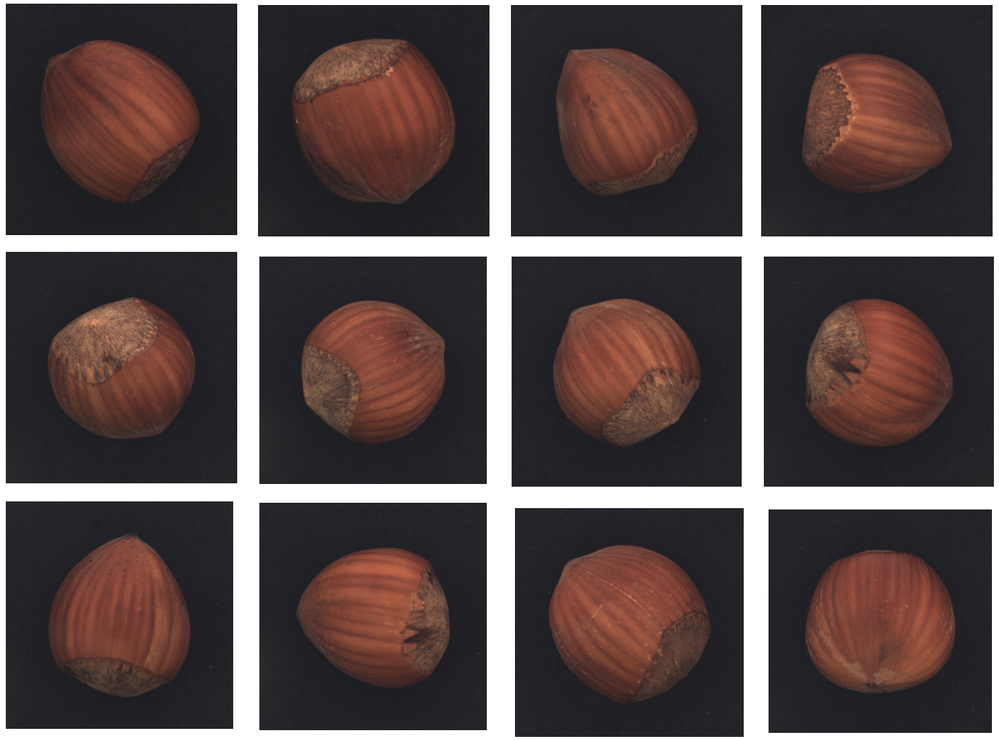
normal



abnormal

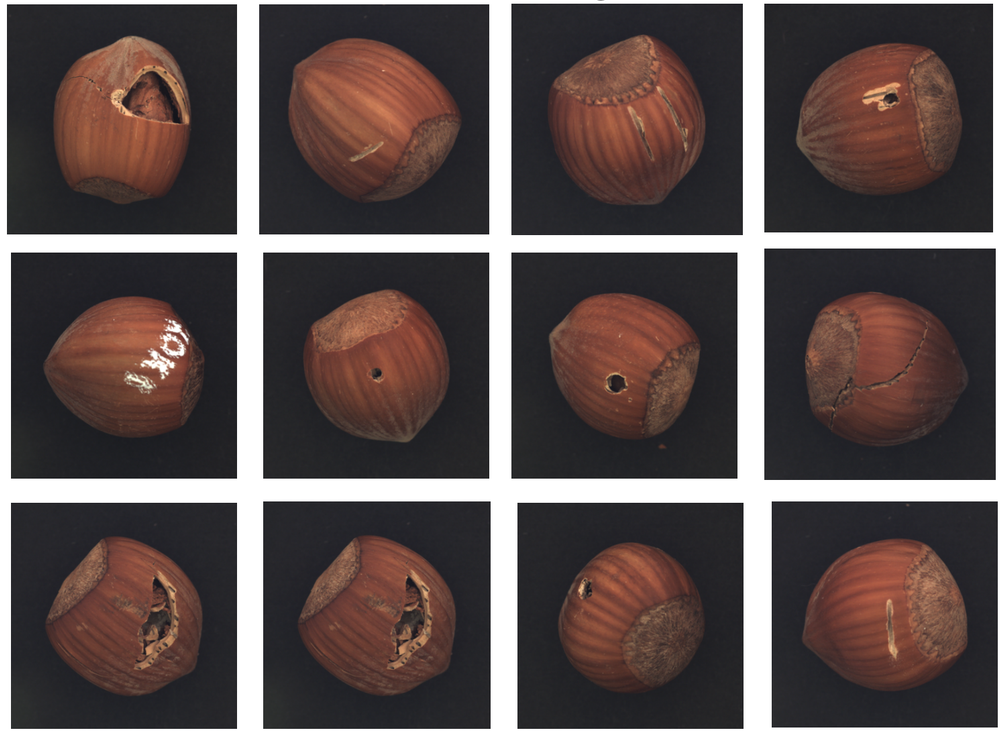
ground truth

normal images



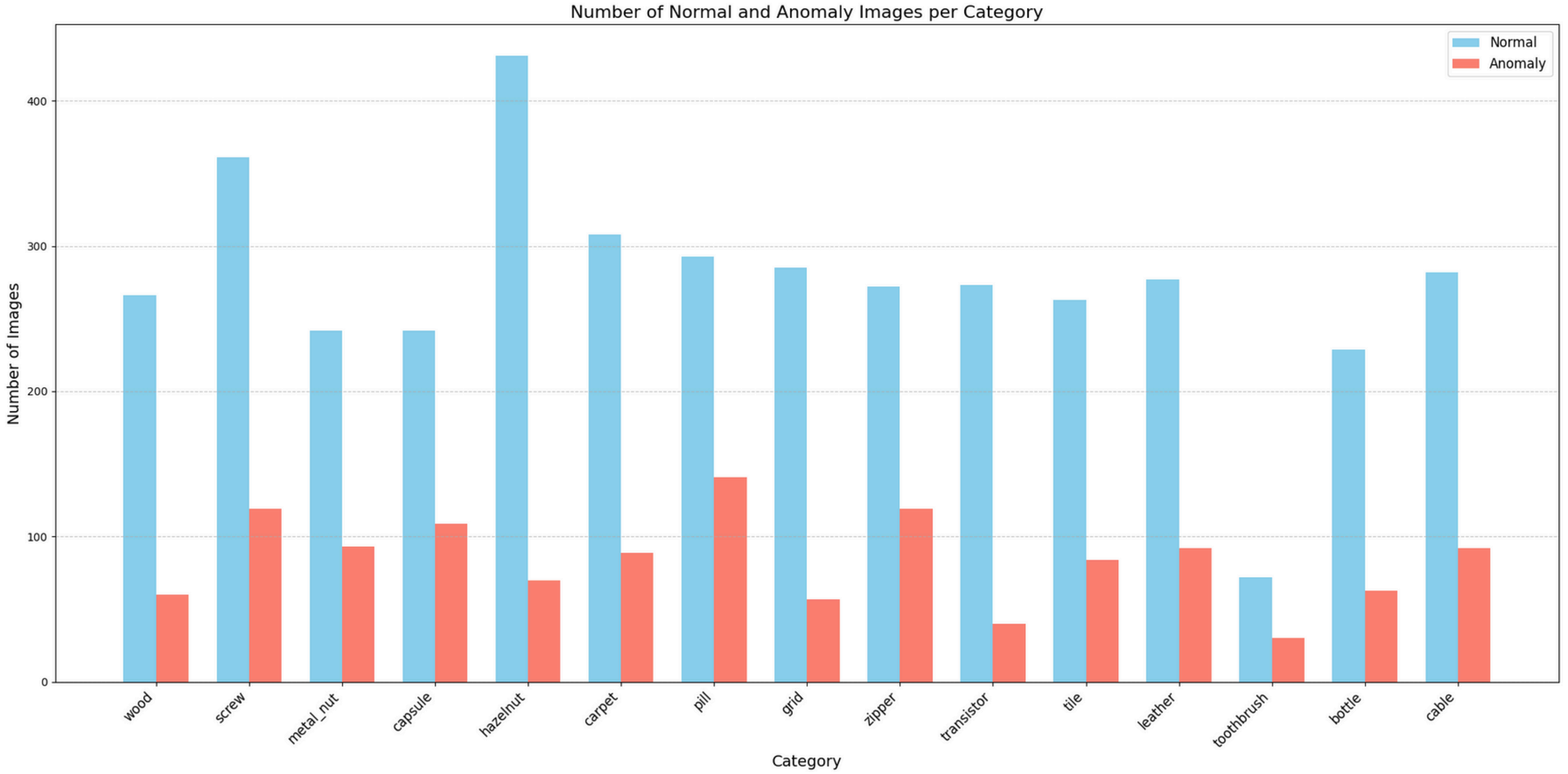
...

abnormal images

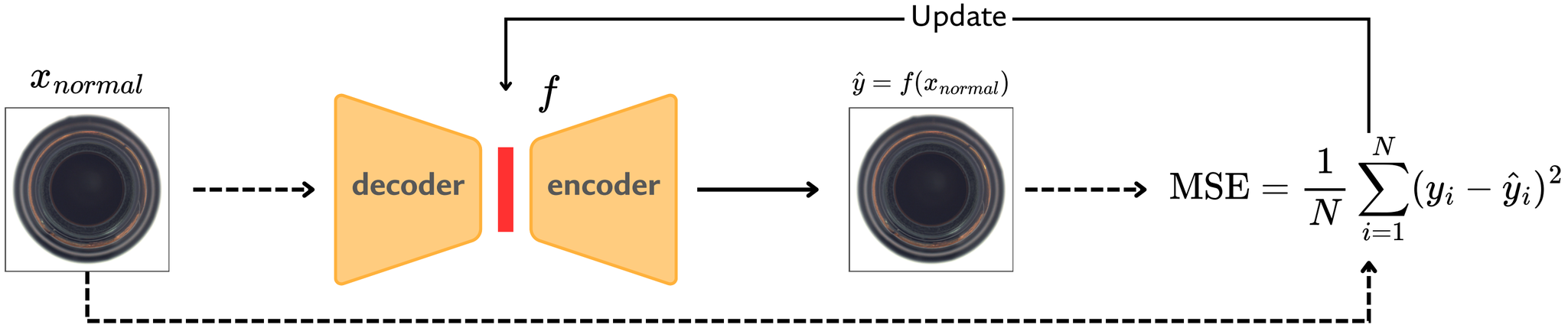


...

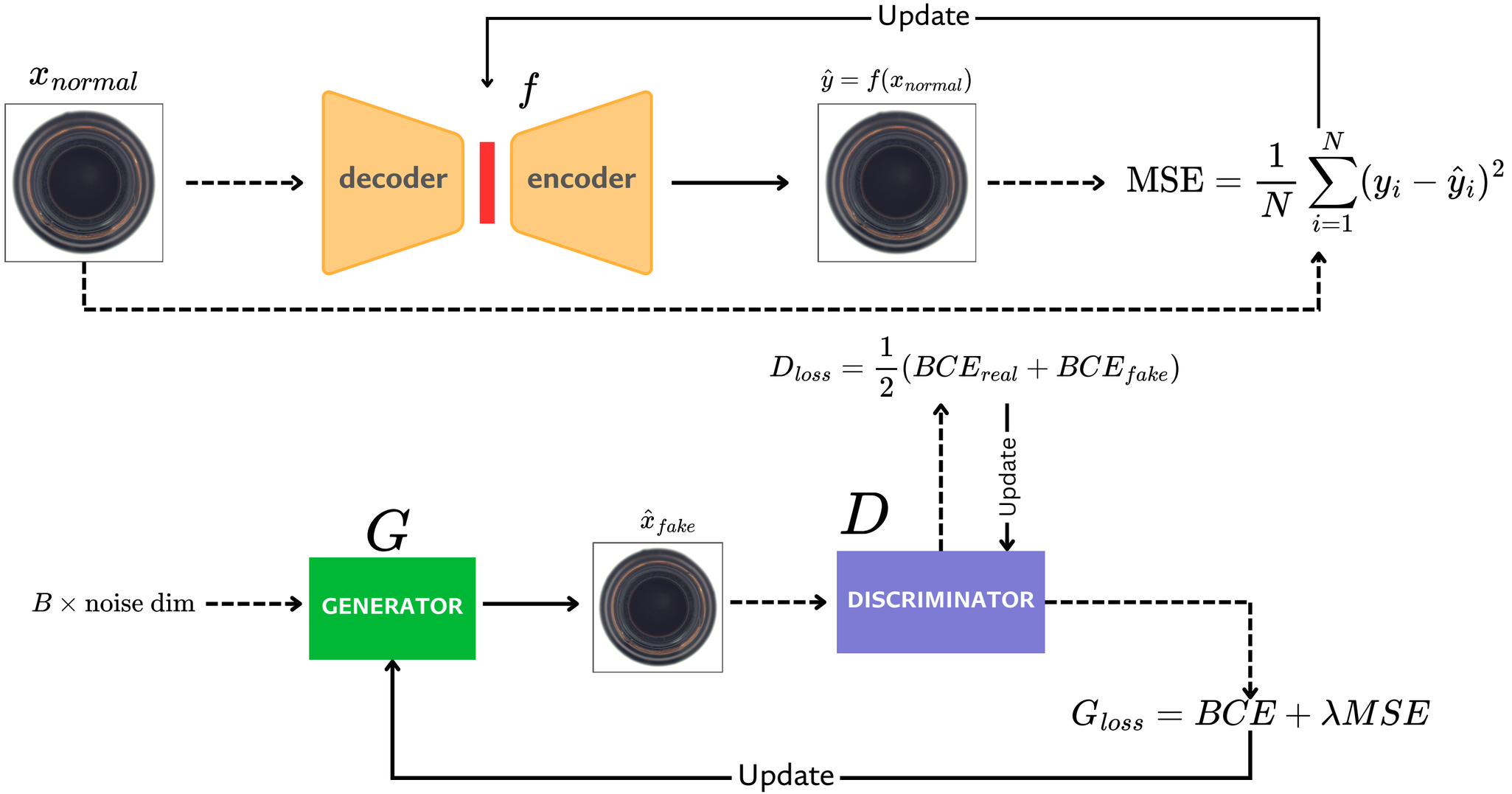
MVTec AD Dataset



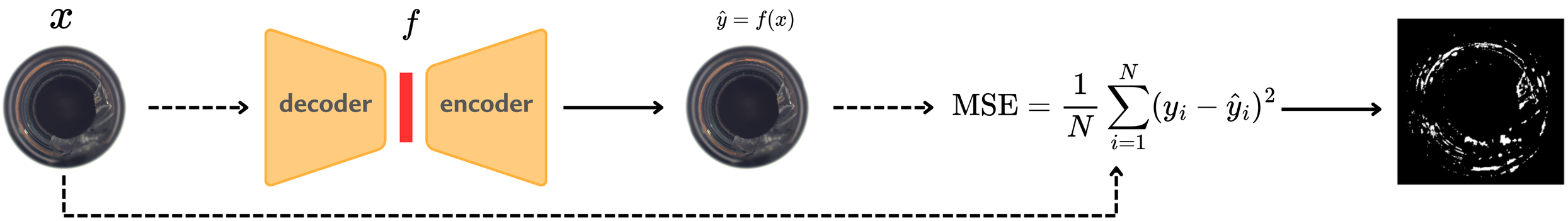
Naive Reconstruction Approach: Pipeline train

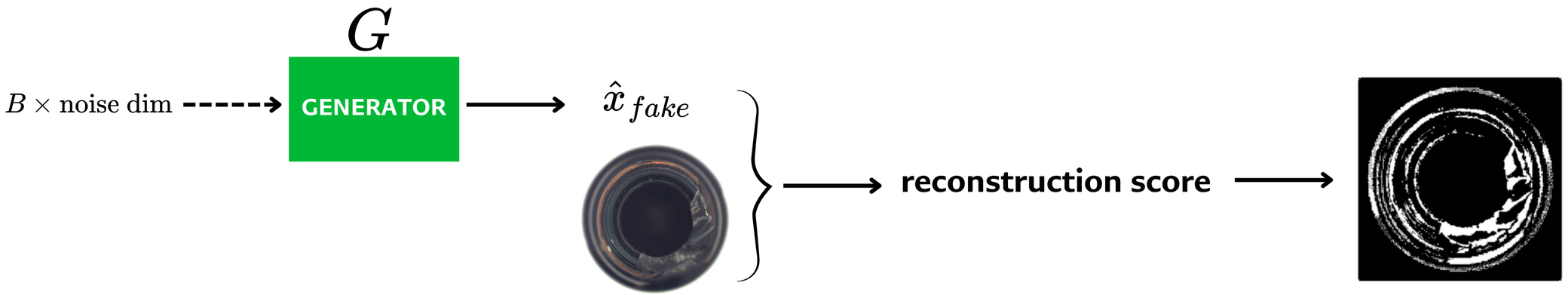


Naive Reconstruction Approach: Pipeline train



Naive Reconstruction Approach: Pipeline inference



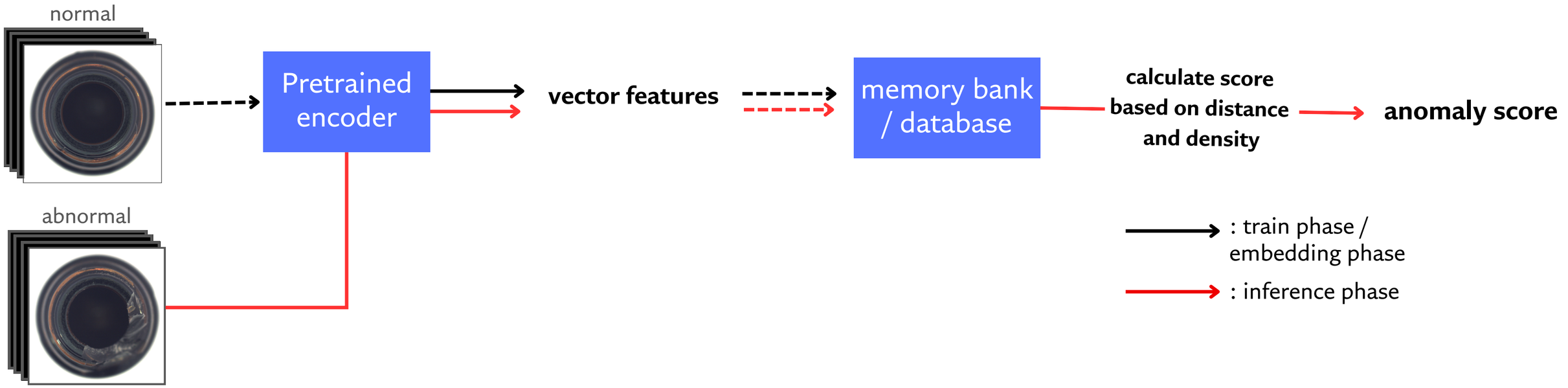


Density-based Approach: Patchcore

- Density-based method: detecting abnormal data based on the distribution of normal data.

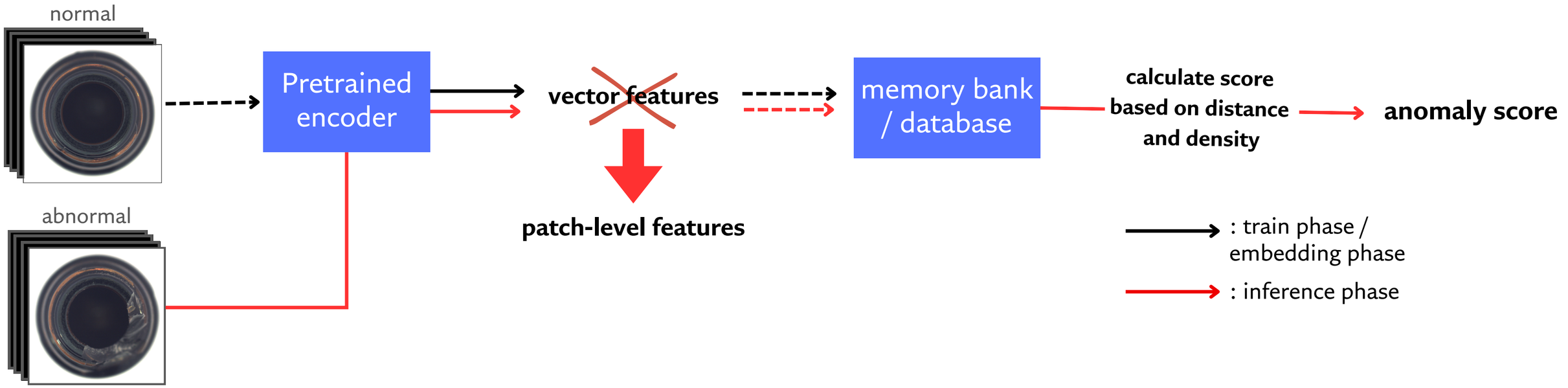
Density-based Approach: Patchcore

- Density-based method: detecting abnormal data based on the distribution of normal data.



Density-based Approach: Patchcore

- Density-based method: detecting abnormal data based on the distribution of normal data.

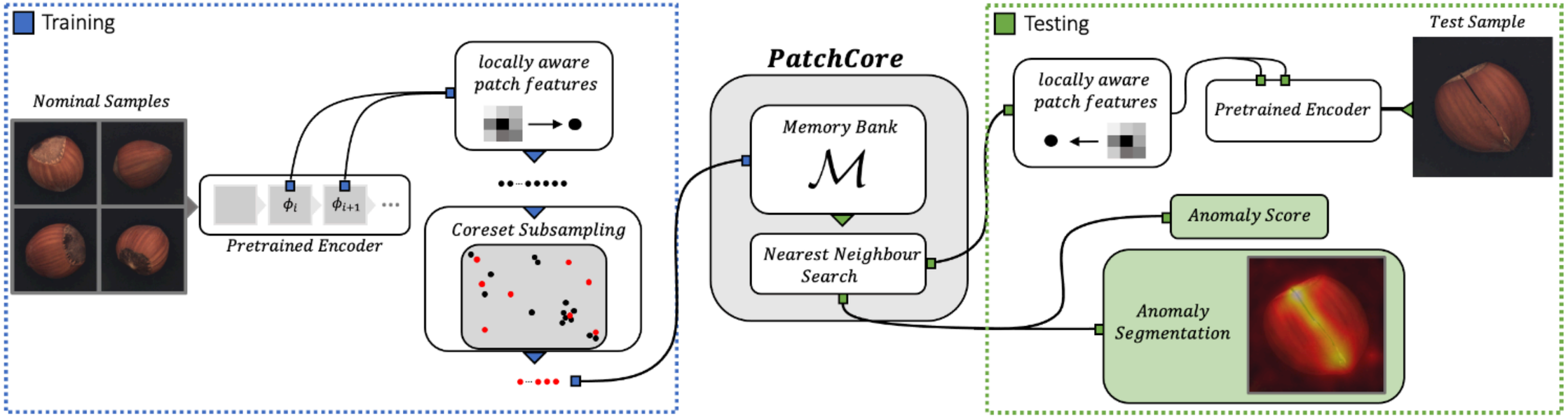


Local search problem

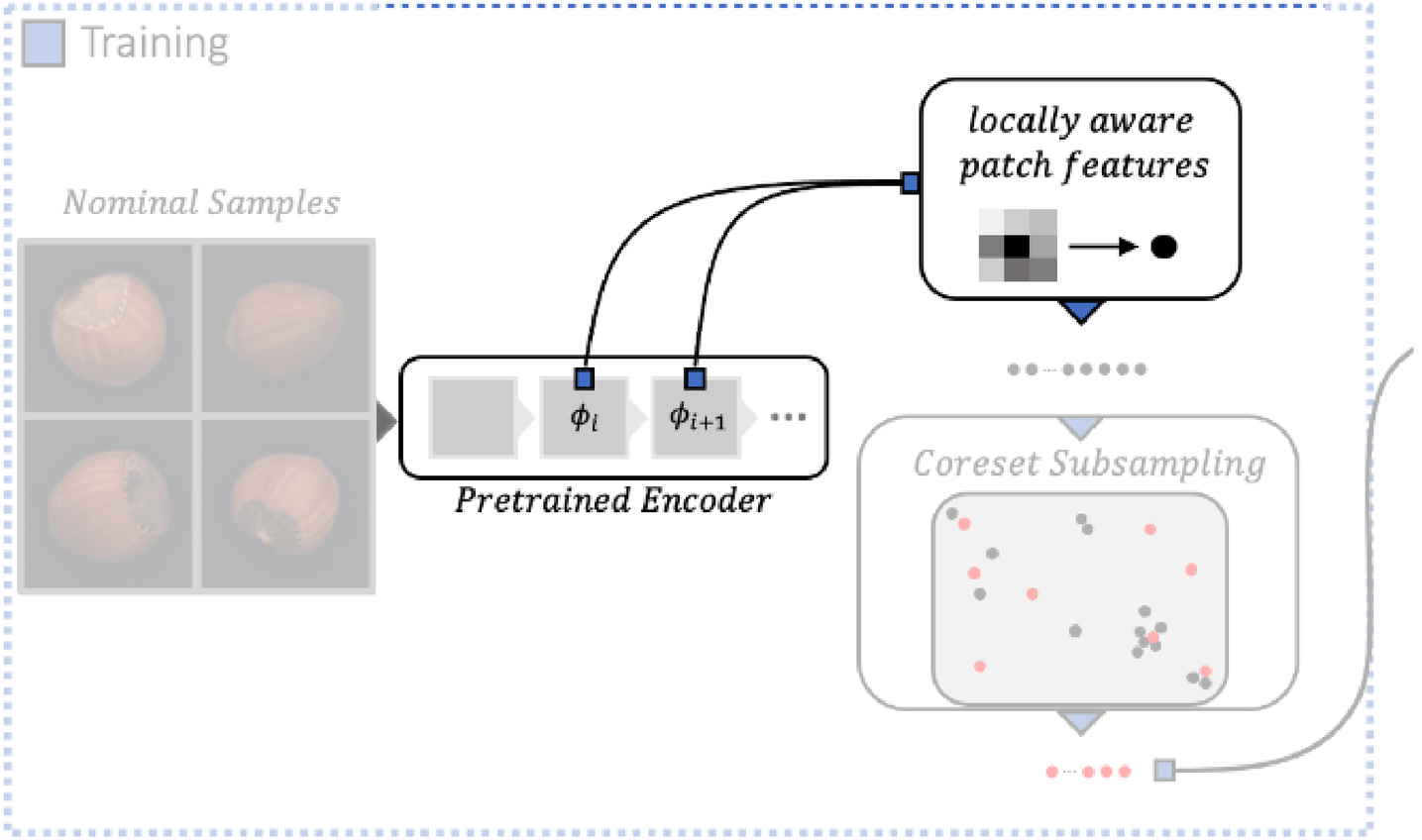
Local patch features & Coreset Subsampling

Density-based Approach: Patchcore

- Density-based method: detecting abnormal data based on the distribution of normal data.

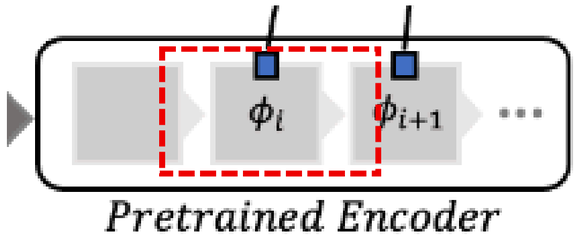


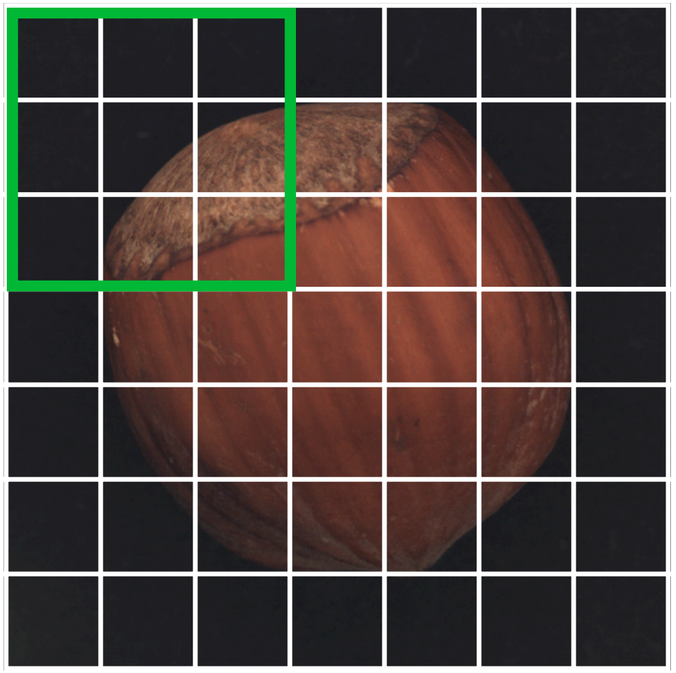
Patchcore: Local patch features

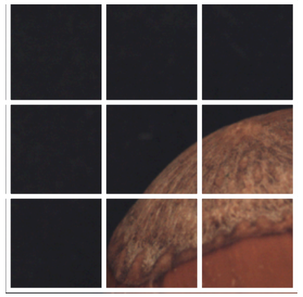


Patchcore: Local patch features

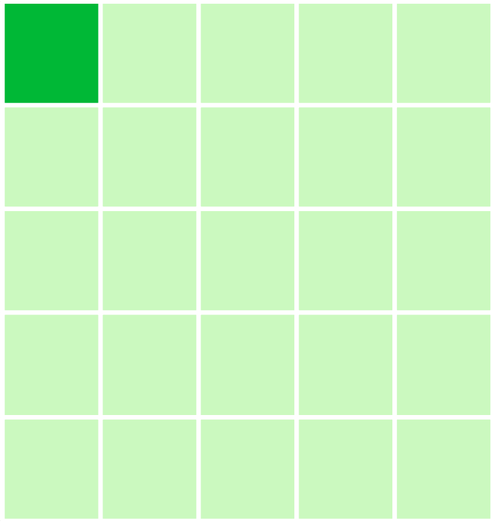
- **Feature map**: C x 6 x 6
- **Patch size (p)**: 3 / **Stride**: 1 / **Padding**: 1







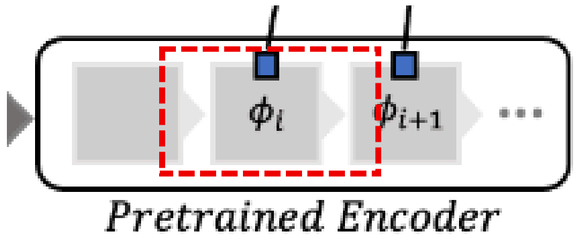
local patch feature **i**

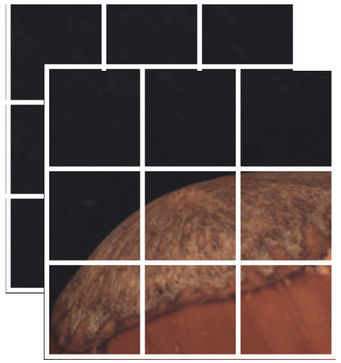


$(5 \times 5) \times (C \times 3 \times 3)$

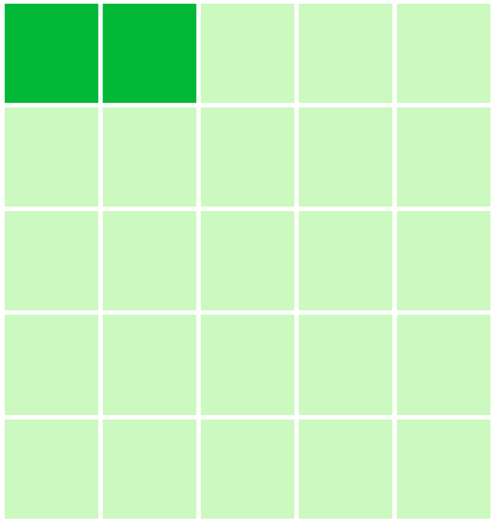
Patchcore: Local patch features

- **Feature map**: C x 6 x 6
- **Patch size (p)**: 3 / **Stride**: 1 / **Padding**: 1





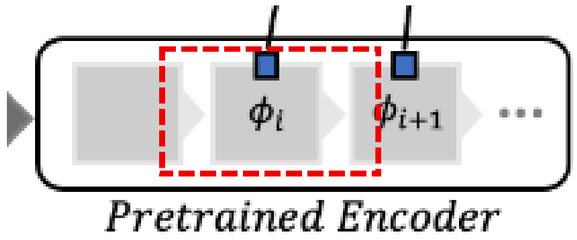
local patch feature **i**

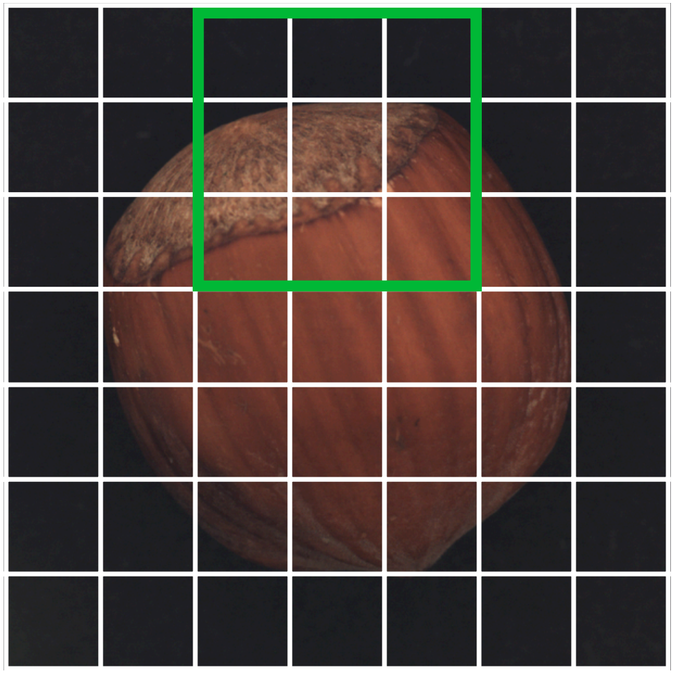


$(5 \times 5) \times (C \times 3 \times 3)$

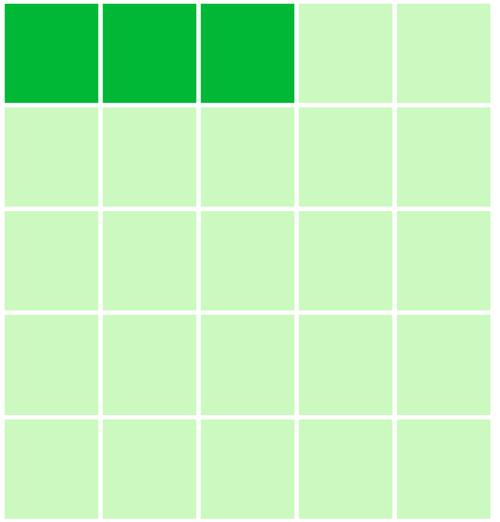
Patchcore: Local patch features

- **Feature map**: C x 6 x 6
- **Patch size (p)**: 3 / **Stride**: 1 / **Padding**: 1





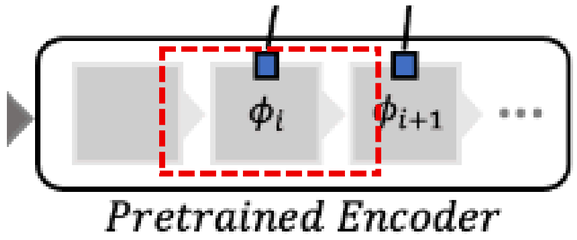
local patch feature **i**



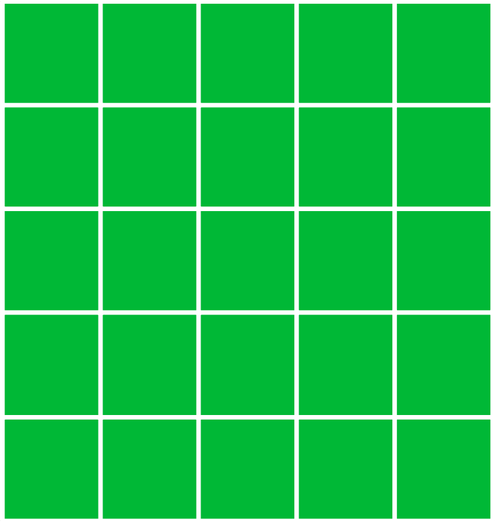
$(5 \times 5) \times (C \times 3 \times 3)$

Patchcore: Local patch features

- **Feature map**: C x 6 x 6
- **Patch size (p)**: 3 / **Stride**: 1 / **Padding**: 1



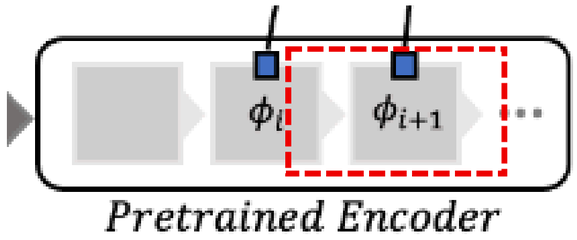
local patch feature **i**



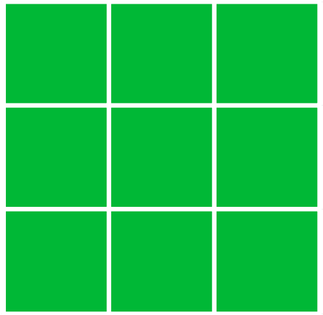
$(5 \times 5) \times (C \times 3 \times 3)$

Patchcore: Local patch features

- **Feature map**: C x 5 x 5
- **Patch size (p)**: 3 / **Stride**: 1



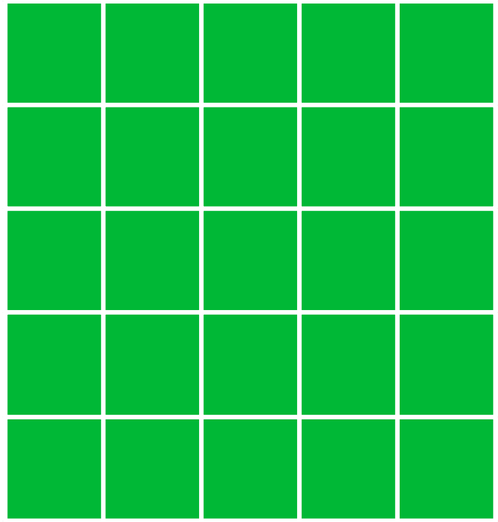
local patch feature **i+1**



$(3 \times 3) \times (C \times 3 \times 3)$

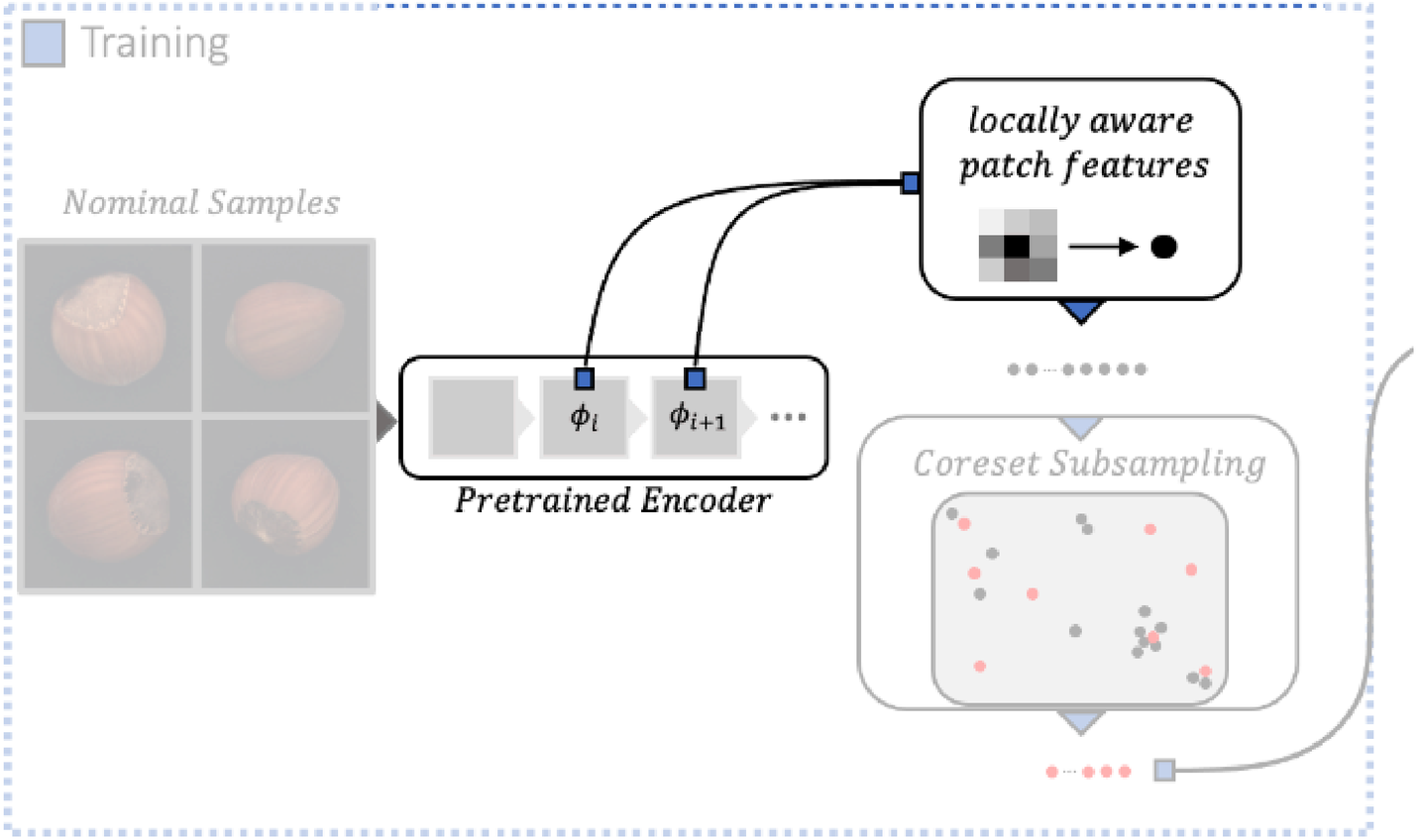
Interpolation

local patch feature **i**

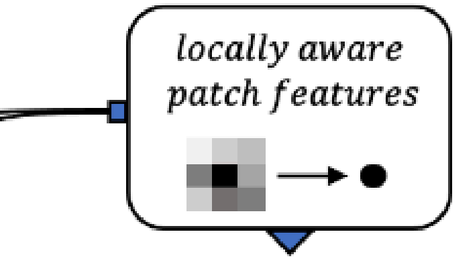


$(5 \times 5) \times (C \times 3 \times 3)$

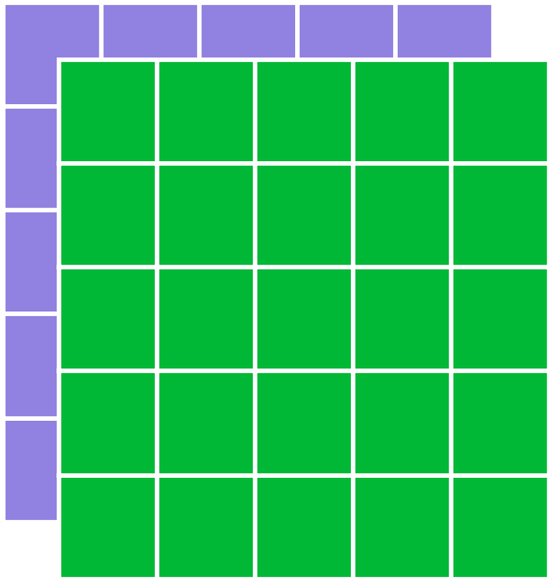
Patchcore: Local patch features



Patchcore: Local patch features



local patch features

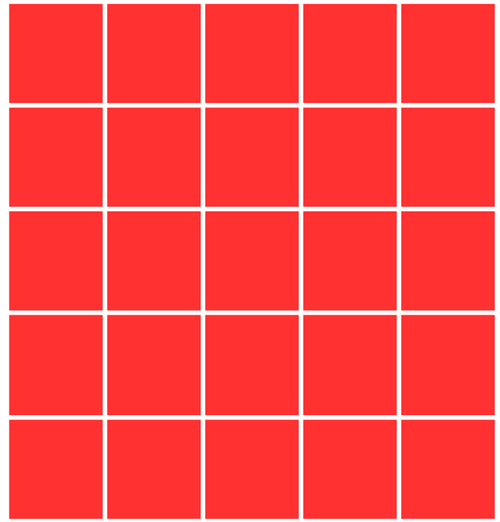


$(5 \times 5) * 2 * (C \times 3 \times 3)$

$(5 \times 5) * 2 * (C_{target})$

Global
Average
Pooling

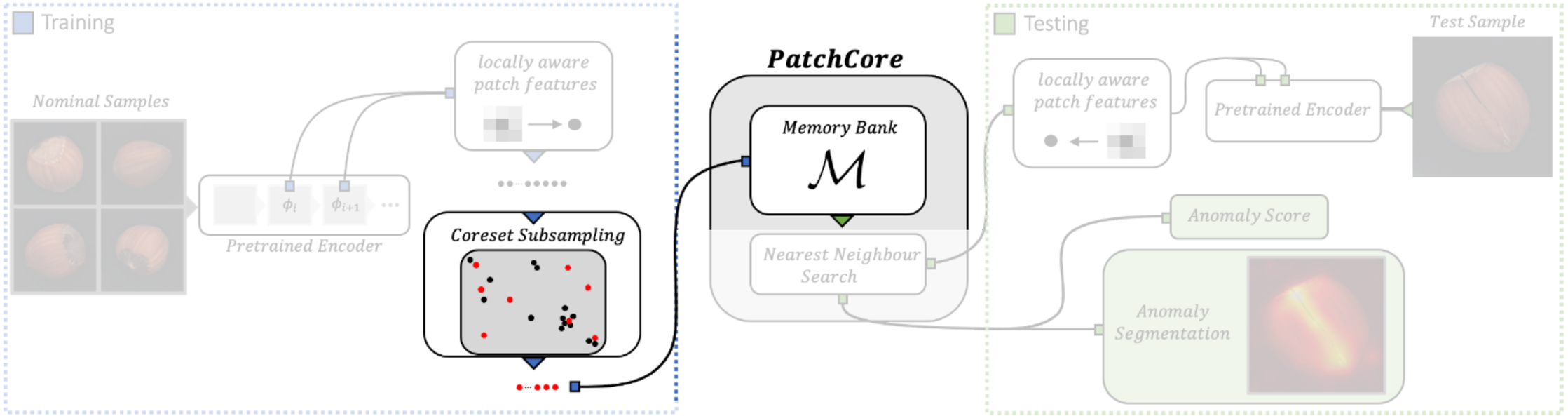
corset



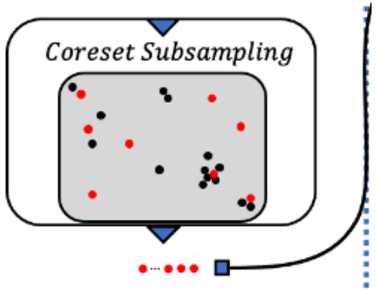
$(5 \times 5) * (2 * C_{target})$

$(5 \times 5) * C_{target}$

Patchcore: Coreset Subsampling



Patchcore: Coreset Subsampling



Algorithm 1: *PatchCore* memory bank.

Input: Pretrained ϕ, hierarchies j, nominal data $\mathcal{X}_N$, stride s, patchsize p, coreset target l, random linear projection ψ.

Output: Patch-level Memory bank $\mathcal{M}$.

Algorithm:

$\mathcal{M} \leftarrow \{\}$

for $x_i \in \mathcal{X}_N$ **do**

$\quad \mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{P}_{s,p}(\phi_j(x_i))$

end

/* Apply greedy coreset selection. */

$\mathcal{M}_C \leftarrow \{\}$

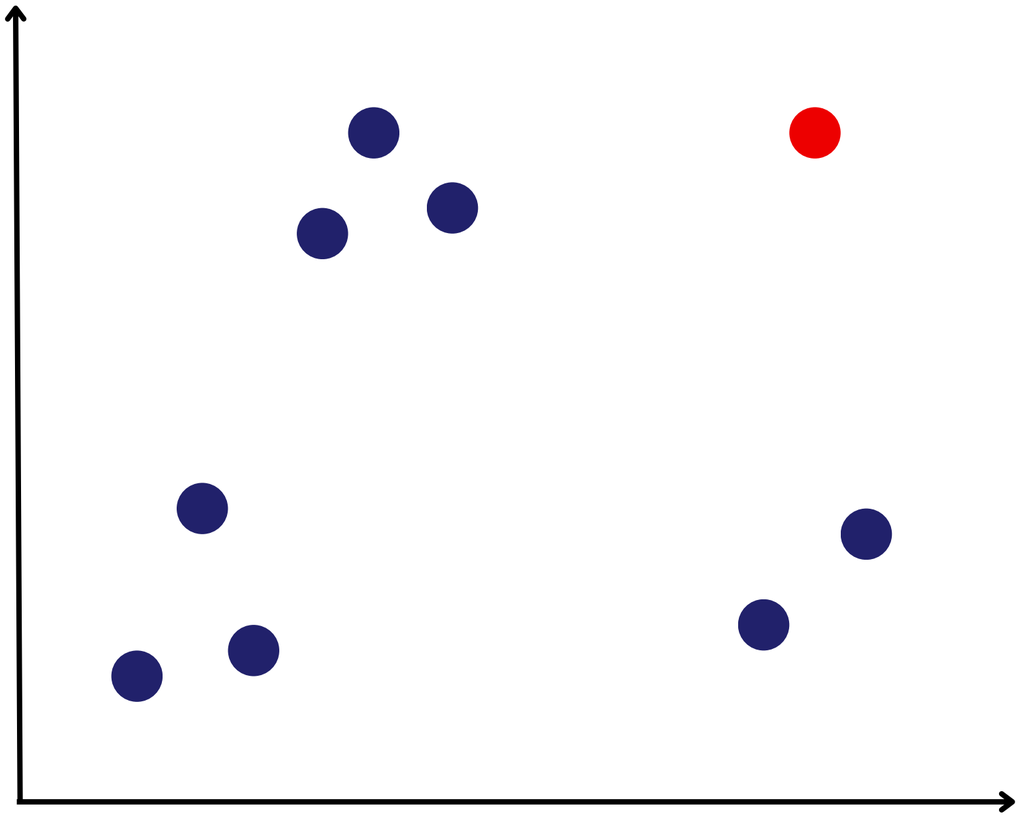
for $i \in [0, ..., l-1]$ **do**

$\quad m_i \leftarrow \underset{m \in \mathcal{M} - \mathcal{M}_C}{\arg\max} \min_{n \in \mathcal{M}_C} \|\psi(m) - \psi(n)\|_2$

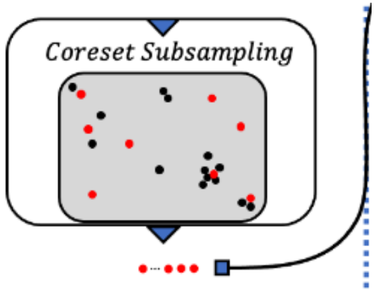
$\quad \mathcal{M}_C \leftarrow \mathcal{M}_C \cup \{m_i\}$

end

$\mathcal{M} \leftarrow \mathcal{M}_C$



Patchcore: Coreset Subsampling



Algorithm 1: *PatchCore* memory bank.

Input: Pretrained ϕ, hierarchies j, nominal data $\mathcal{X}_N$, stride s, patchsize p, coreset target l, random linear projection ψ.

Output: Patch-level Memory bank $\mathcal{M}$.

Algorithm:

$\mathcal{M} \leftarrow \{\}$

for $x_i \in \mathcal{X}_N$ **do**

$\quad \mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{P}_{s,p}(\phi_j(x_i))$

end

/* Apply greedy coreset selection. */

$\mathcal{M}_C \leftarrow \{\}$

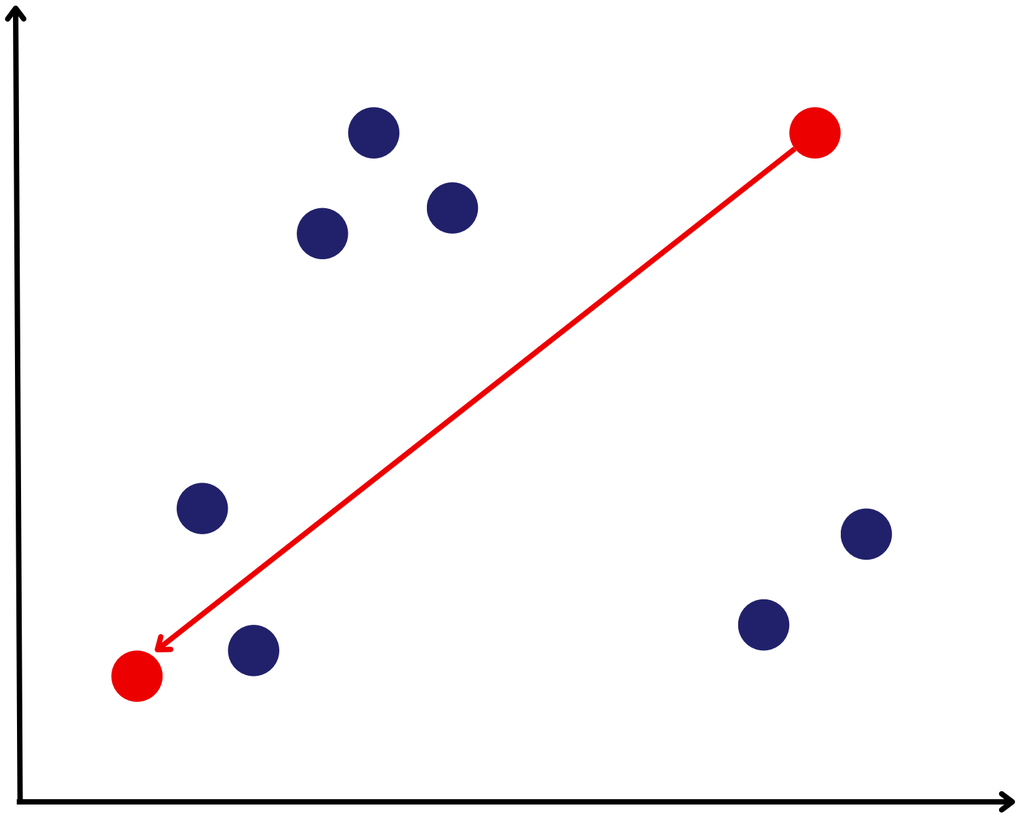
for $i \in [0, ..., l-1]$ **do**

$\quad m_i \leftarrow \underset{m \in \mathcal{M} - \mathcal{M}_C}{\arg\max} \; \underset{n \in \mathcal{M}_C}{\min} \; \|\psi(m) - \psi(n)\|_2$

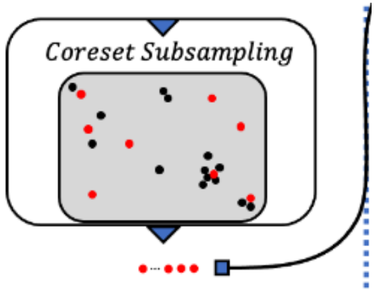
$\quad \mathcal{M}_C \leftarrow \mathcal{M}_C \cup \{m_i\}$

end

$\mathcal{M} \leftarrow \mathcal{M}_C$



Patchcore: Coreset Subsampling



Algorithm 1: *PatchCore* memory bank.

Input: Pretrained ϕ, hierarchies j, nominal data $\mathcal{X}_N$, stride s, patchsize p, coreset target l, random linear projection ψ.

Output: Patch-level Memory bank $\mathcal{M}$.

Algorithm:

$\mathcal{M} \leftarrow \{\}$

for $x_i \in \mathcal{X}_N$ **do**

$\quad \mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{P}_{s,p}(\phi_j(x_i))$

end

/* Apply greedy coreset selection. */

$\mathcal{M}_C \leftarrow \{\}$

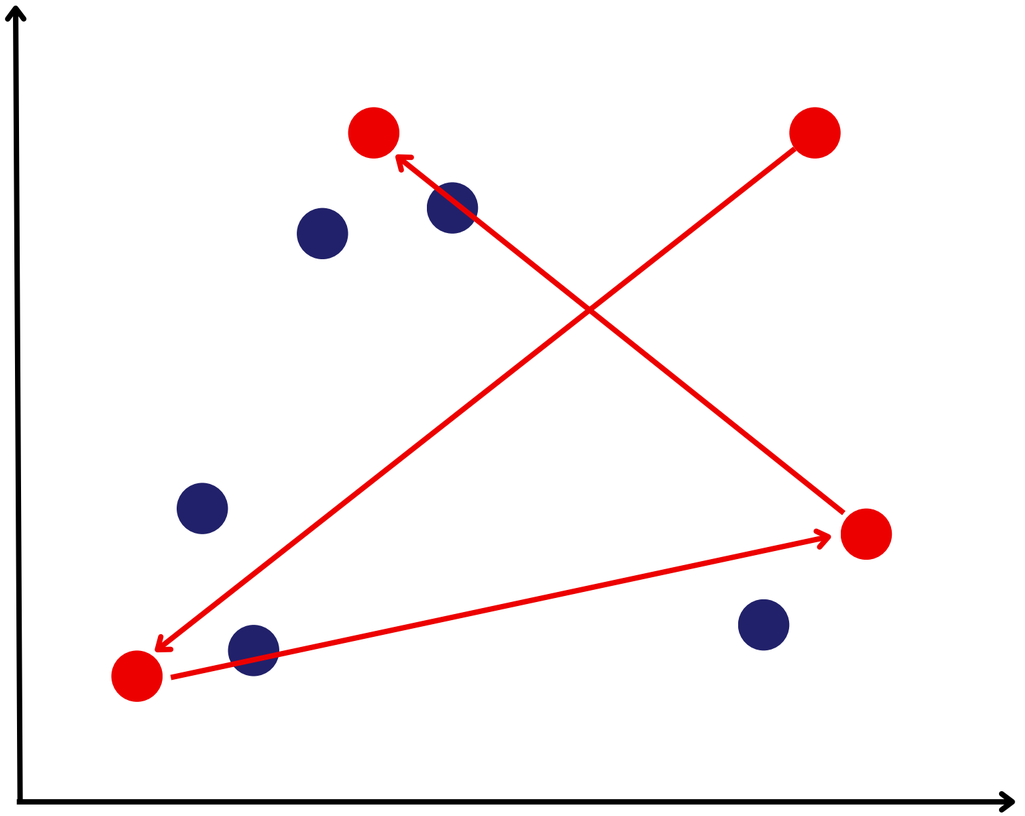
for $i \in [0, ..., l-1]$ **do**

$\quad m_i \leftarrow \underset{m \in \mathcal{M} - \mathcal{M}_C}{\arg\max} \min_{n \in \mathcal{M}_C} \|\psi(m) - \psi(n)\|_2$

$\quad \mathcal{M}_C \leftarrow \mathcal{M}_C \cup \{m_i\}$

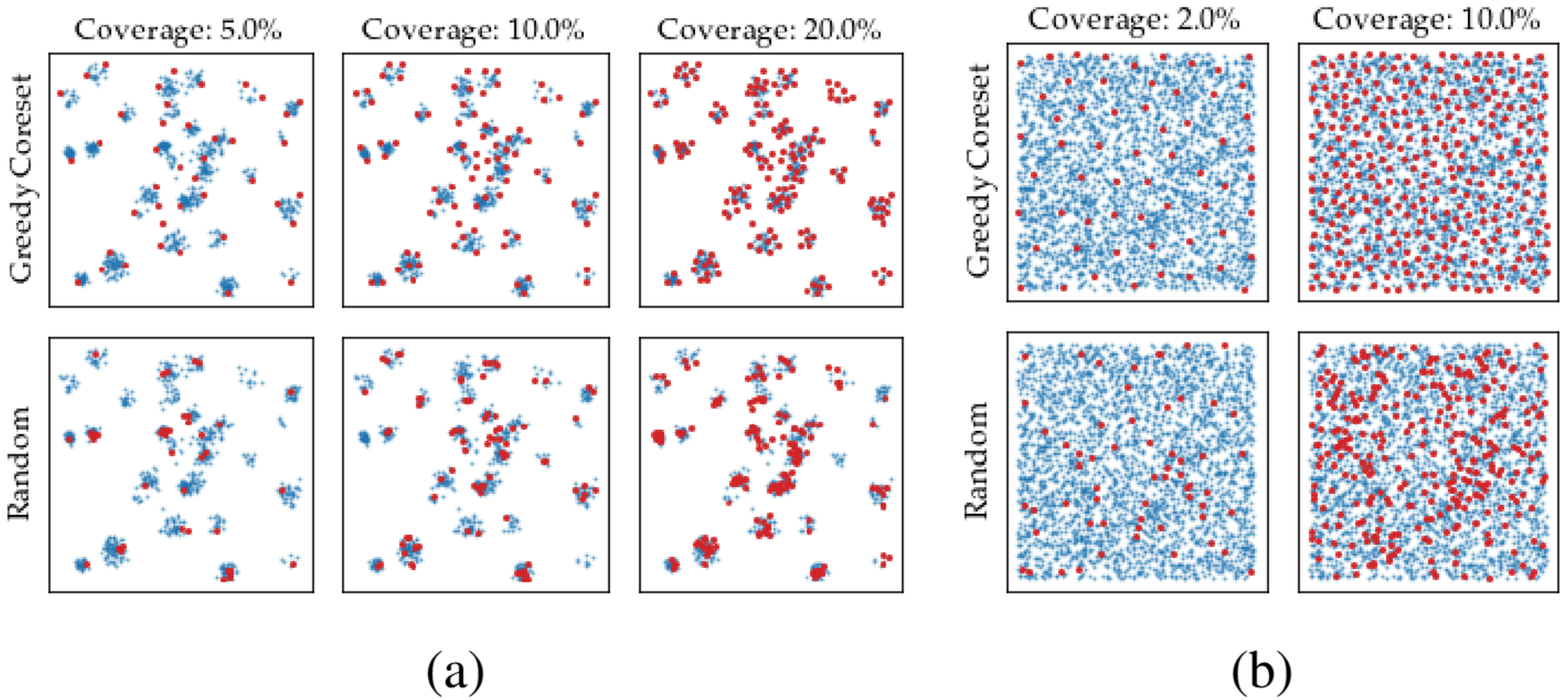
end

$\mathcal{M} \leftarrow \mathcal{M}_C$

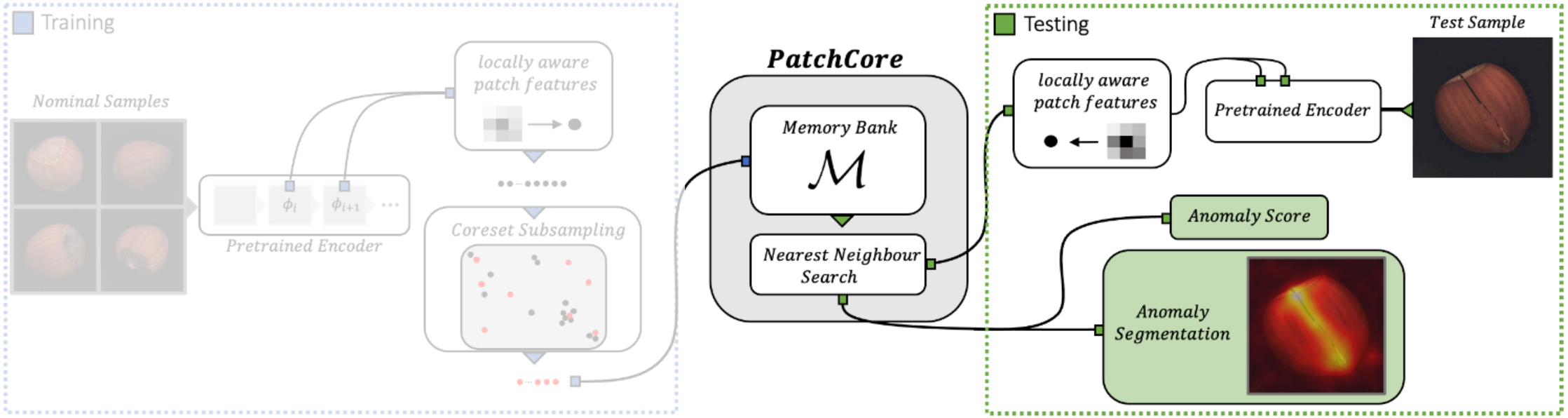


Patchcore: Coreset Subsampling

- Compared to Random Subsampling, there are fewer duplicate weights

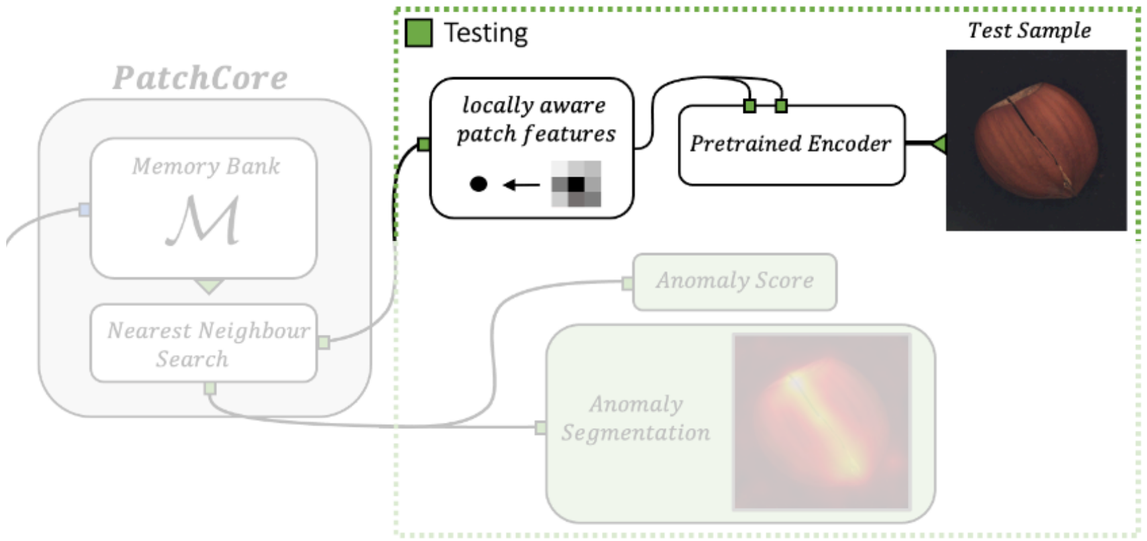


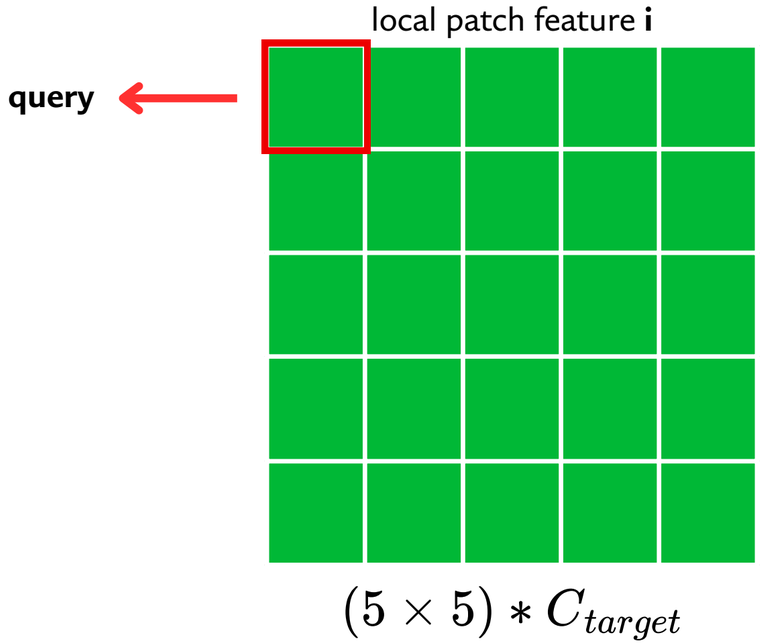
Patchcore: Detection & Localization



Patchcore: Detection & Localization

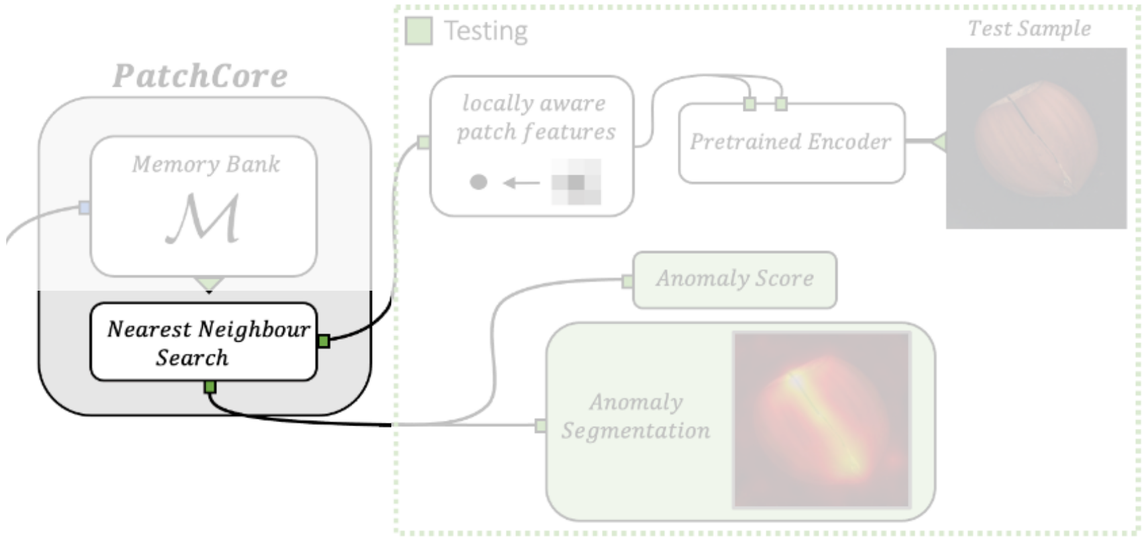
- Anomaly score: is calculated by **distance score** between **the query** and **coreset** using Nearest Neighbor.
- Nearest Neighbor: can be computed by default method in vector database like *faiss*, or *milvus*.

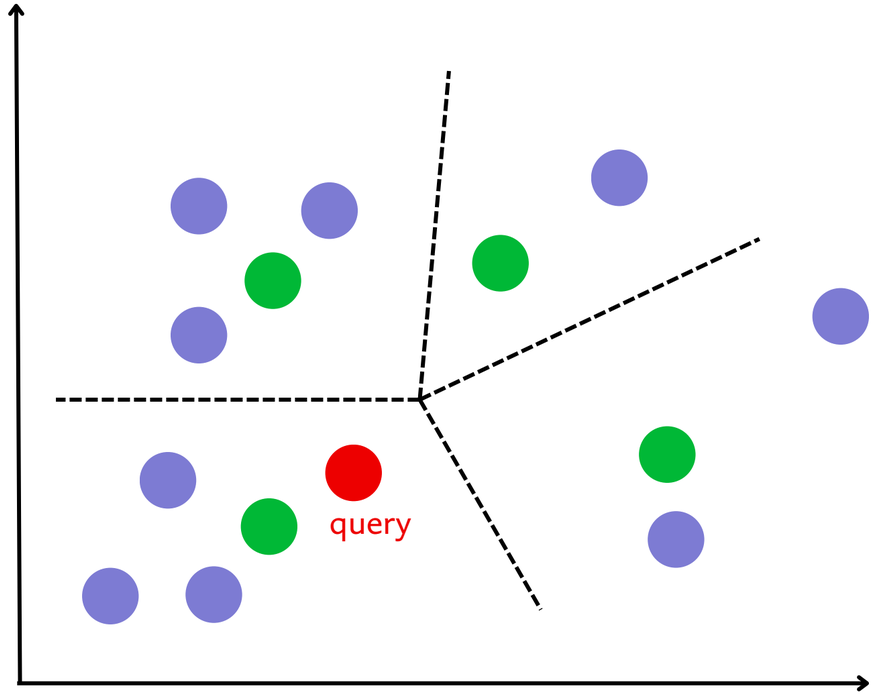




Patchcore: Detection & Localization

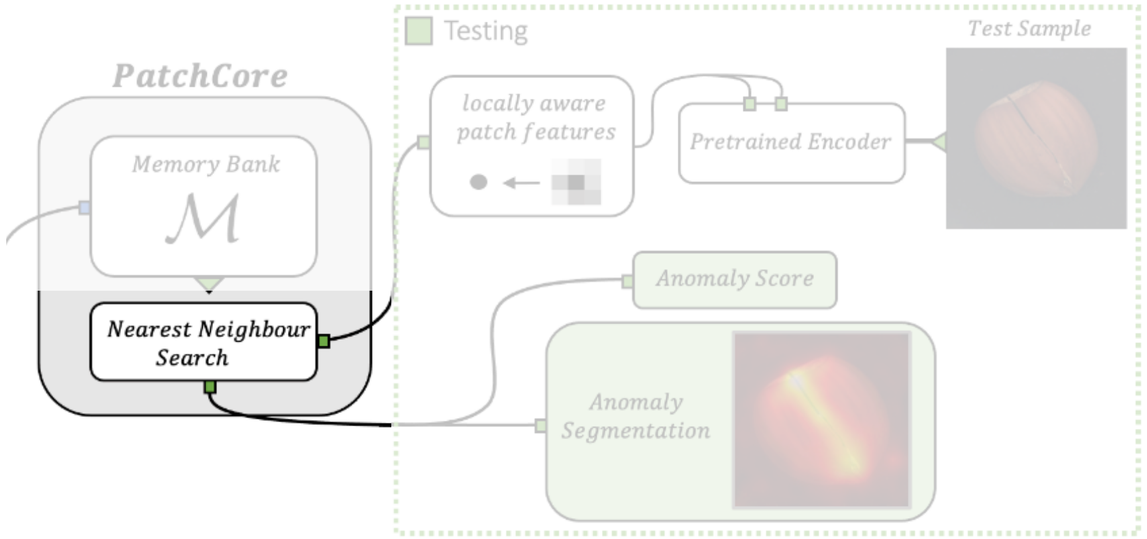
- Approximate Nearest Neighbor (ANN) Search

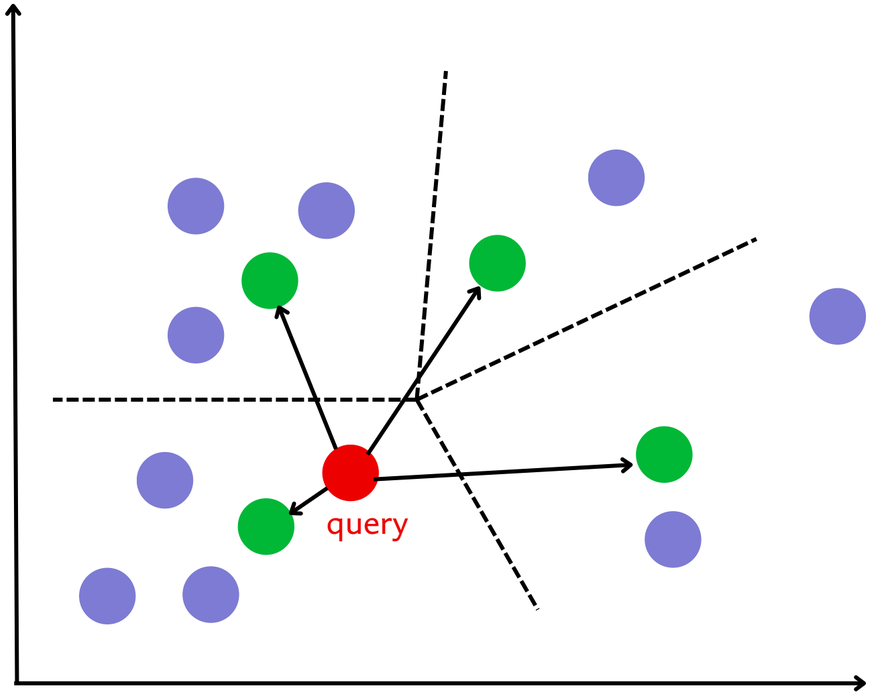




Patchcore: Detection & Localization

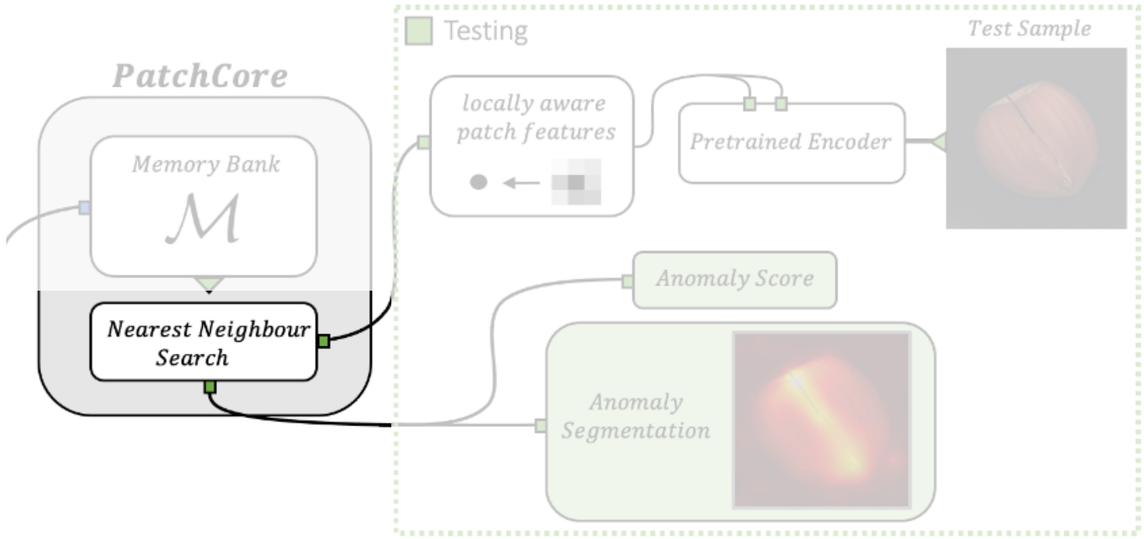
- Approximate Nearest Neighbor (ANN) Search

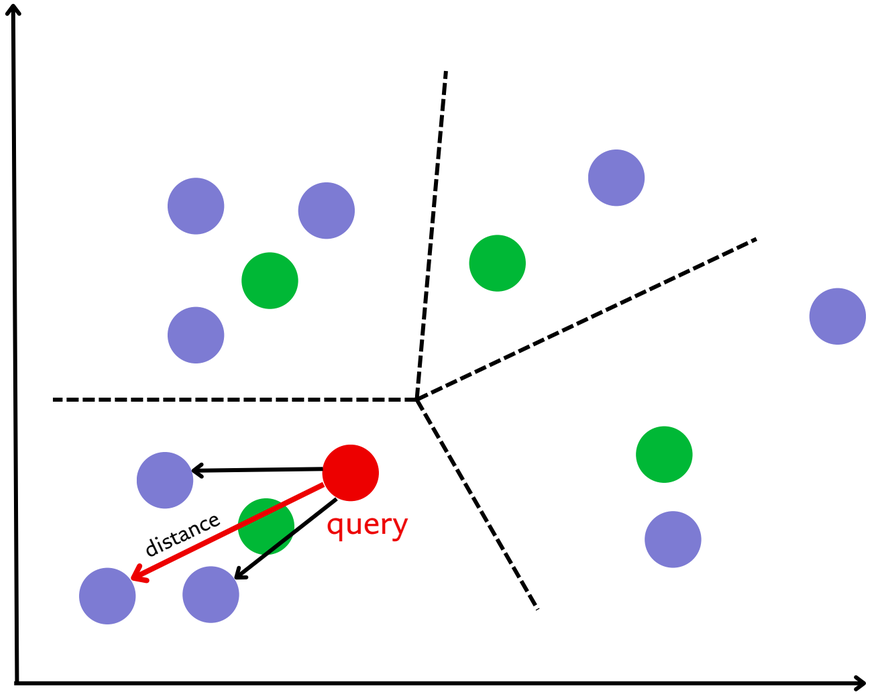




Patchcore: Detection & Localization

- Approximate Nearest Neighbor (ANN) Search

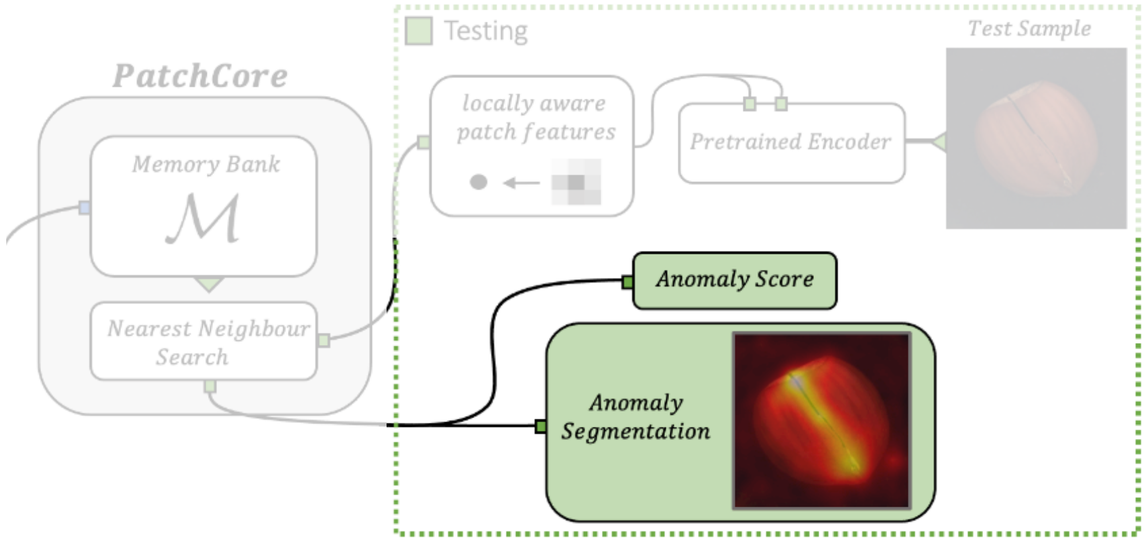


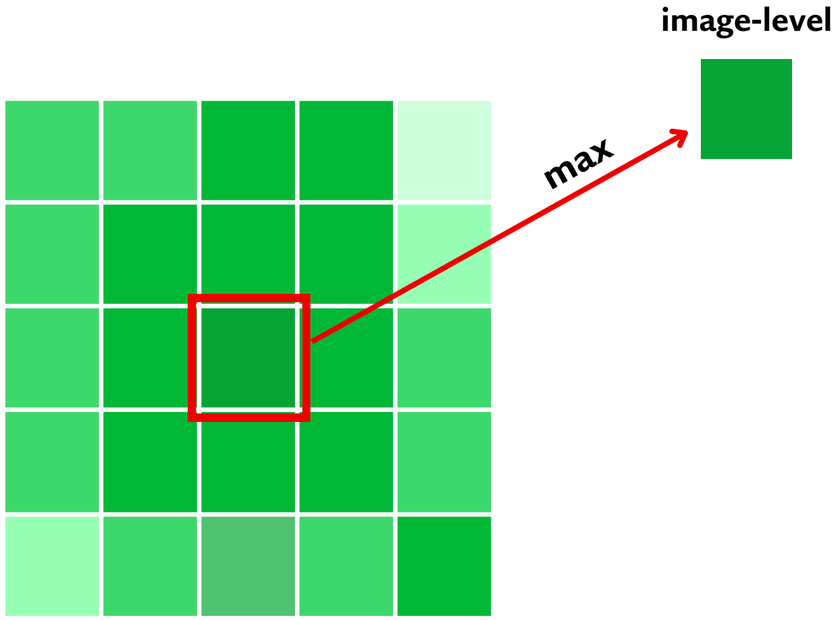


Patchcore: Detection & Localization

- Image-level: if a patch is detected as anomaly

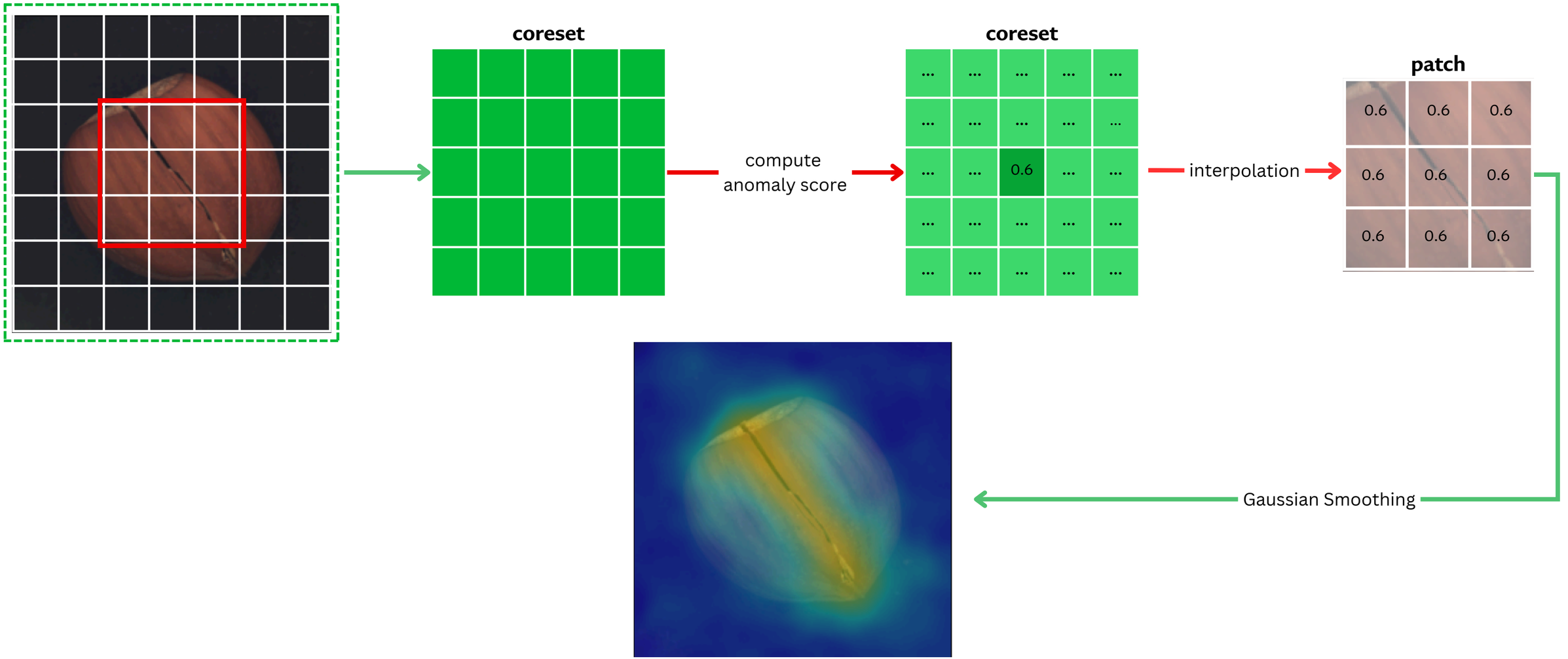
  ➡ A whole image will be detected as anomaly.



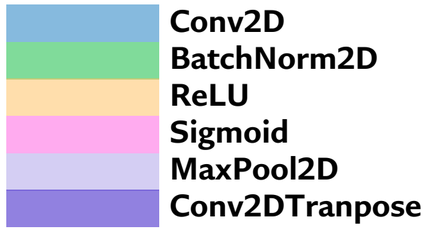


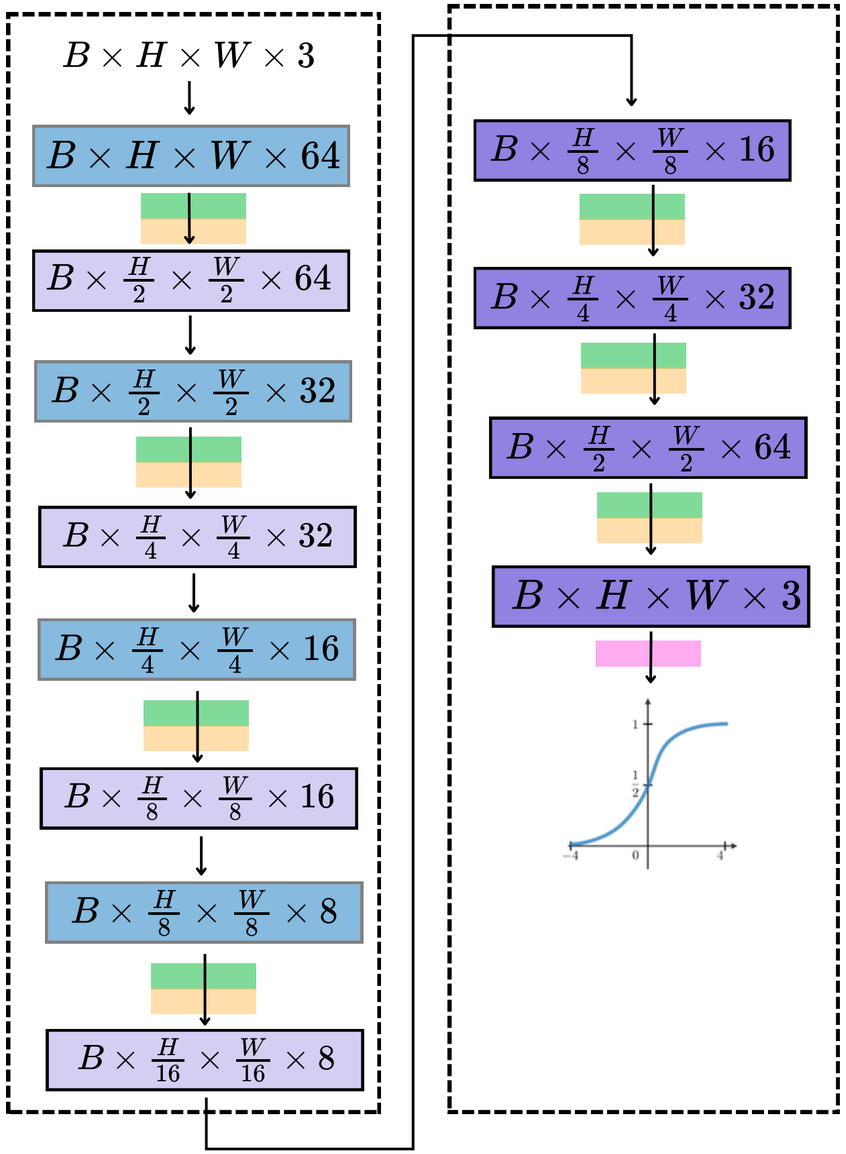
Patchcore: Detection & Localization

- Pixel-level:



Experiment: AutoEncoder Architecture

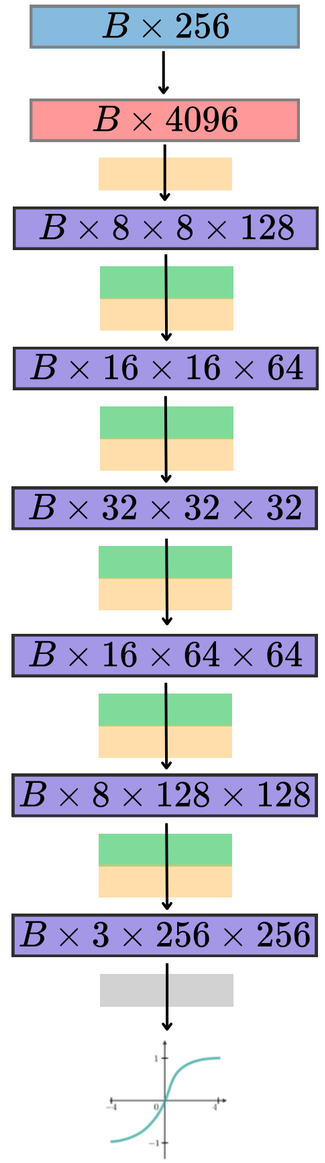




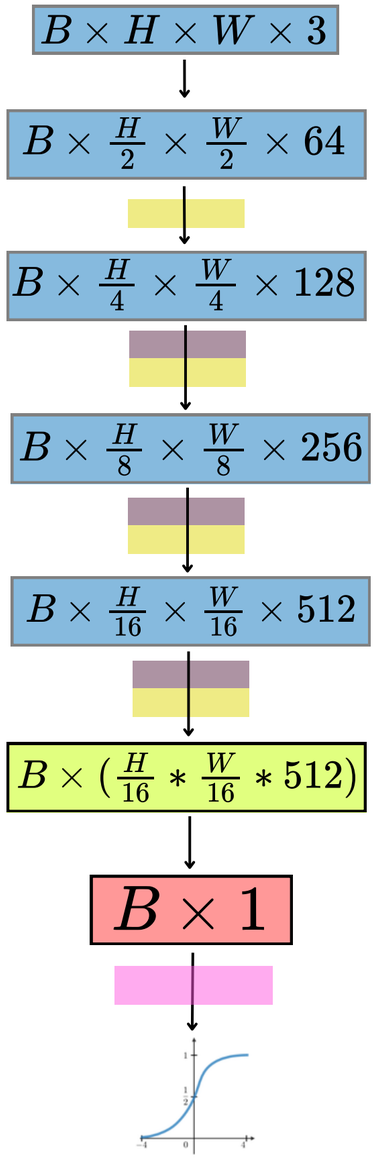
Experiment: GAN Architecture

- Conv2D
- BatchNorm2D
- ReLU
- Linear
- Sigmoid
- MaxPool2D
- Flatten
- Conv2DTranpose
- Reshape
- Tanh
- LeakyReLU
- InstaneNorm2d

Generator

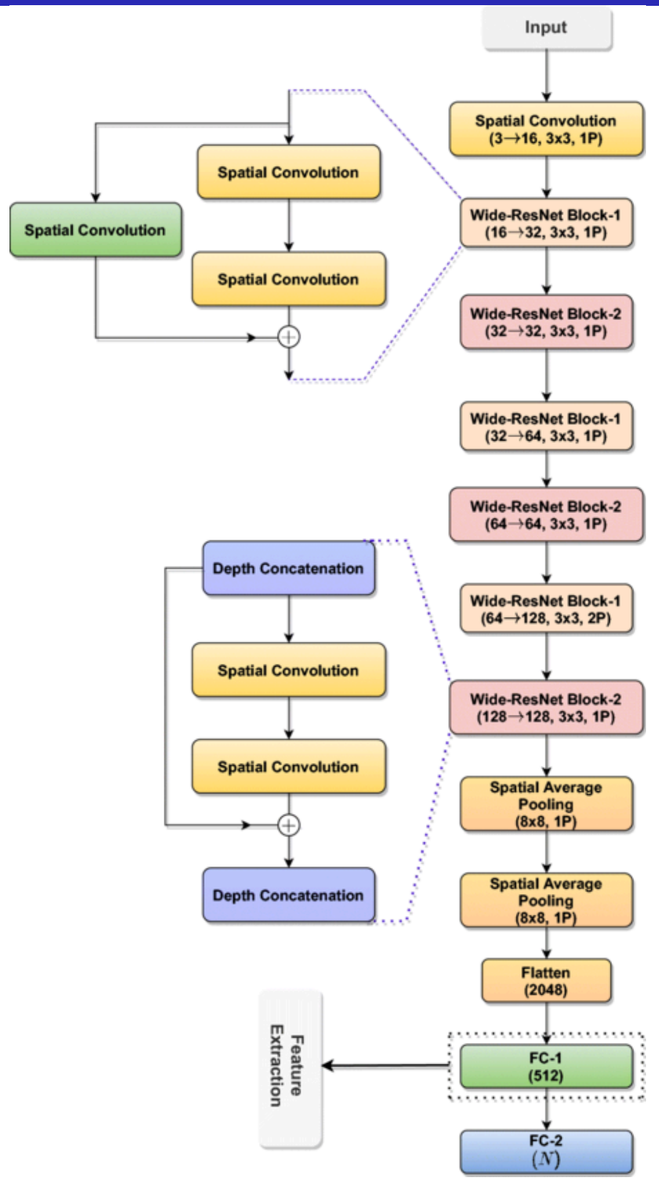


Discriminator



Experiment: PatchCore Architecture

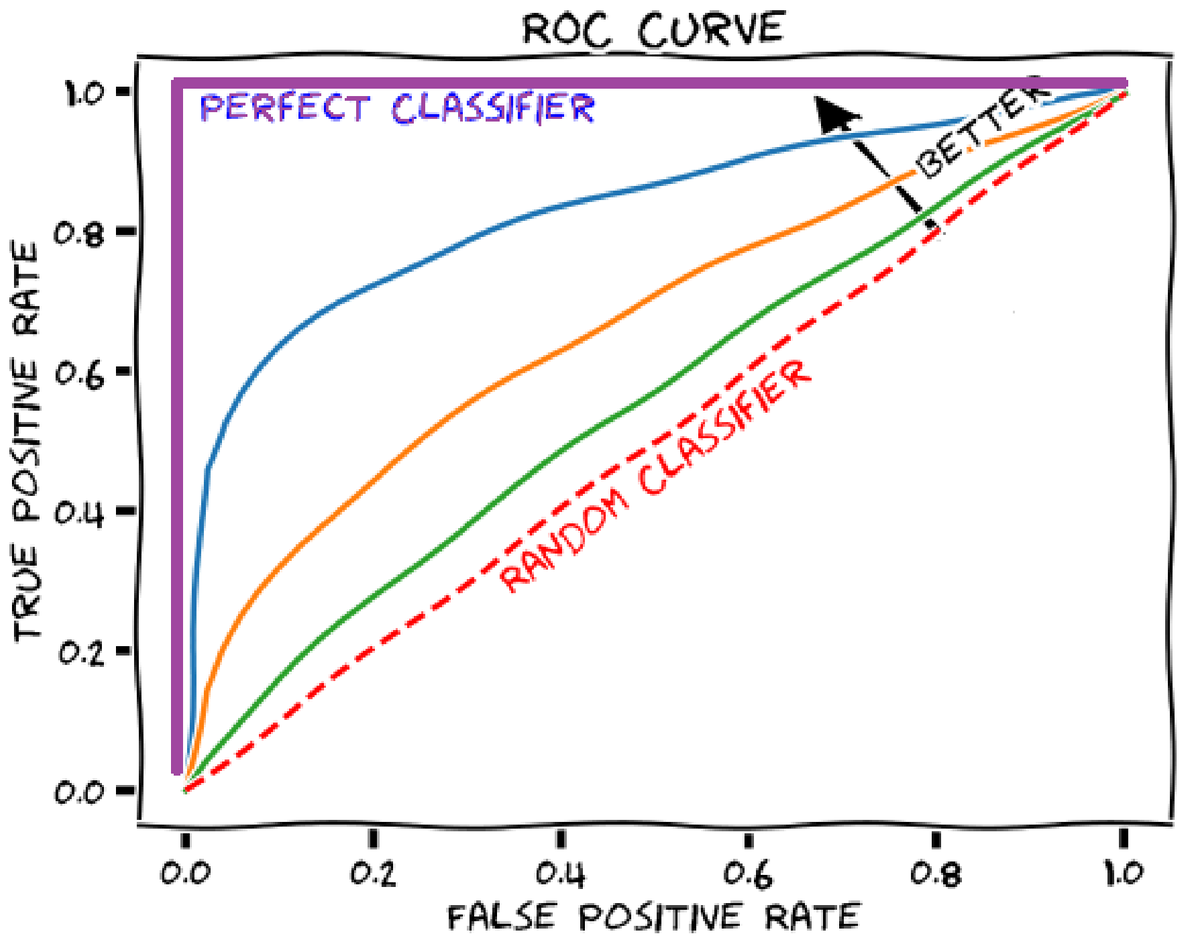
- Backbone: Wide-ResNet50-2
  Vector database: memory bank (**anomalib**)



Evaluation metrics

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$



Results

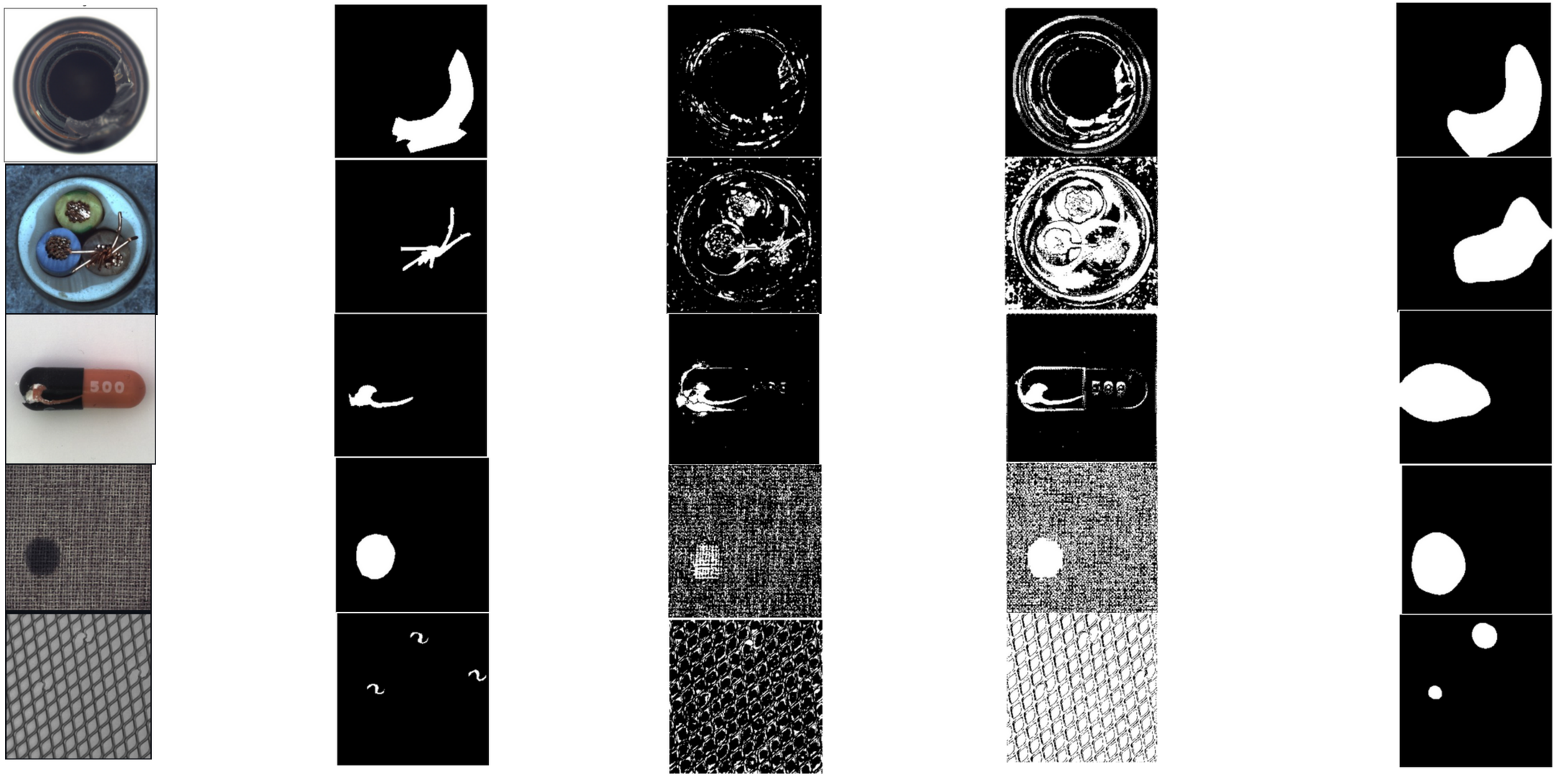
Image-ROC

Categories/Methods	Reconstruction		Density-based
	AE	GAN	Patchcore
Carpet	0.3419	0.5983	0.9903
Grid	0.7778	0.4428	0.9766
Leather	0.5557	0.7351	1
Tile	0.4975	0.6542	0.9877
Wood	0.9088	0.9425	0.9885
Bottle	0.7294	0.7103	1
Cable	0.5652	0.7192	0.9833
Capsule	0.5768	0.4587	0.9752
Hazelnut	0.91	0.8807	1
Metal nut	0.4179	0.7185	0.998
Pill	0.7867	0.6465	0.9372
Screw	0.7965	0.4384	0.9819
Toothbrush	0.6056	0.65	0.9944
Transistor	0.5762	0.824	0.9991
Zipper	0.4073	0.4799	0.9923
Mean	0.6508	0.6643	0.9867

Pixel-ROC

Categories/Methods	Reconstruction		Density-based
	AE	GAN	Patchcore
Carpet	0.5382	0.5542	0.9886
Grid	0.6564	0.5445	0.9795
Leather	0.8477	0.6892	0.9899
Tile	0.4817	0.5736	0.9476
Wood	0.6596	0.6632	0.93
Bottle	0.6938	0.7793	0.9815
Cable	0.6578	0.8345	0.9806
Capsule	0.8167	0.8161	0.9874
Hazelnut	0.9375	0.8971	0.9846
Metal nut	0.7759	0.7179	0.9821
Pill	0.8407	0.8115	0.9767
Screw	0.9055	0.7807	0.9881
Toothbrush	0.8035	0.9041	0.9862
Transistor	0.5817	0.7254	0.9701
Zipper	0.7265	0.713	0.9793
Mean	0.7417	0.7464	0.9759

Inference



Conclusion & Limitations

	Reconstruction (Naive) Approaches	PatchCore
Pros	• Easy to implement	• No need for re-training • Effective for anomaly detection
Cons	• Instability in training • Overfitting to normal data • Need more complex structures to achieve higher performance	• Dependence on the quality and diversity of the feature bank • Memory size and inference time

THANKS FOR WATCHING!