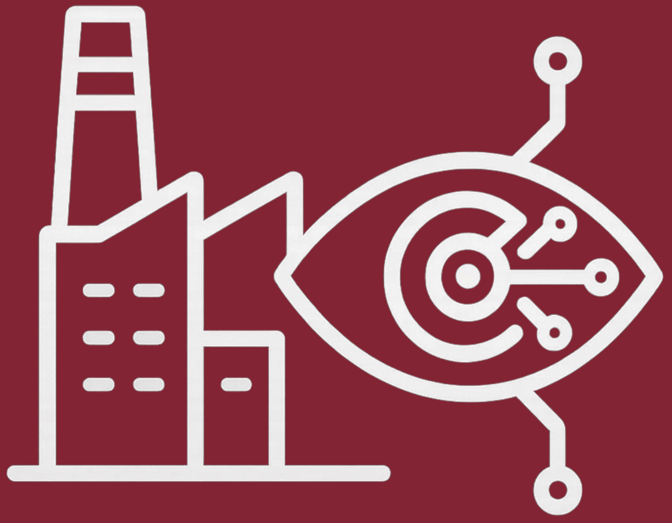
Efficient Anomaly Detection in Industrial Images using Transformers with Dynamic Tanh

Alessandro Massari
Matteo Pelliccione

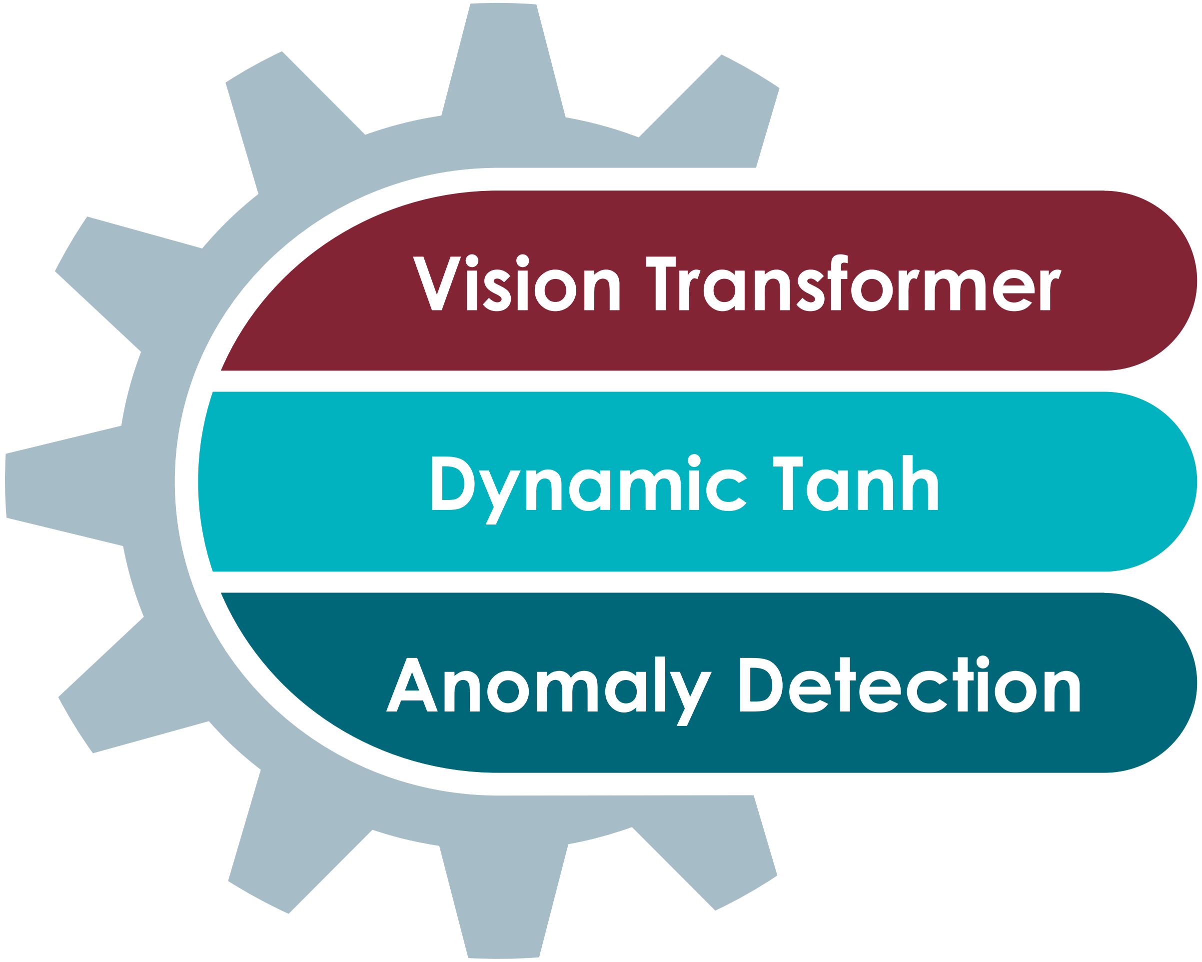
Computer Vision 2024/2025



Outline

- What’s the problem?
- S.O.T.A
- Our approach
- Datasets
- The setup
- Evaluations
- Conclusions
- References

What's the problem?



The task

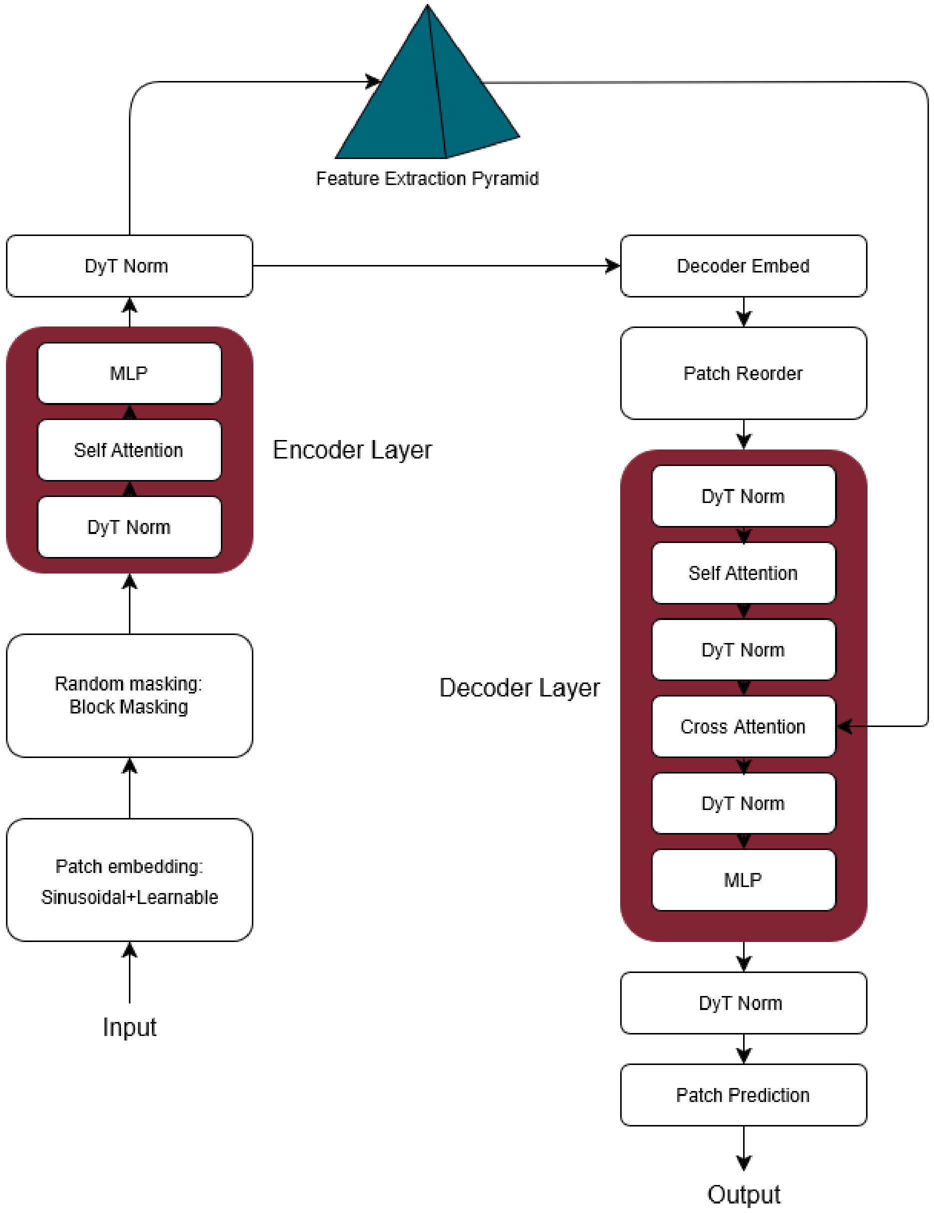
Vision Transformers (ViTs) and Dynamic Tanh (DyT) together enhance industrial anomaly detection by combining powerful feature extraction with efficient processing, improving quality control, safety, and scalability in complex visual data environments, we try to create a pipeline to demonstrate this.

S.O.T.A

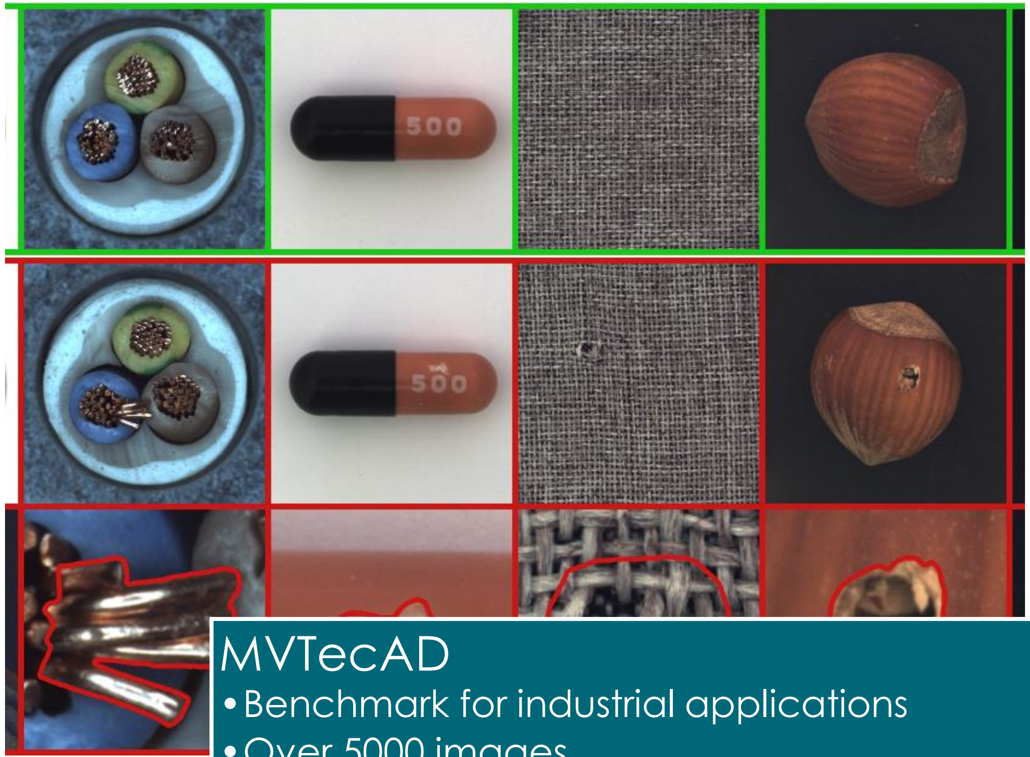
Model / Method	**Year**	**Datasets**	**Strengths**	**Limitations**
VT-ADL (Vision Transformer ADL)	2021	MvTec AD, BTAD	Early ViT-based AD method; combines transformer with GMM for anomaly localization	Moderate localization accuracy; lower BTAD performance
ViT-AE + Memory (Sensors)	2024	MvTec AD, BTAD	Autoencoder + memory + coordinate attention; good detection + localization	Struggles with very fine-grained defects
MSTAD (Masked Subspace Transformer)	2023–24	MvTec AD, BTAD	Masking + subspace embedding improves both detection and localization	Higher model complexity; sensitive to hyperparameters

Our approach

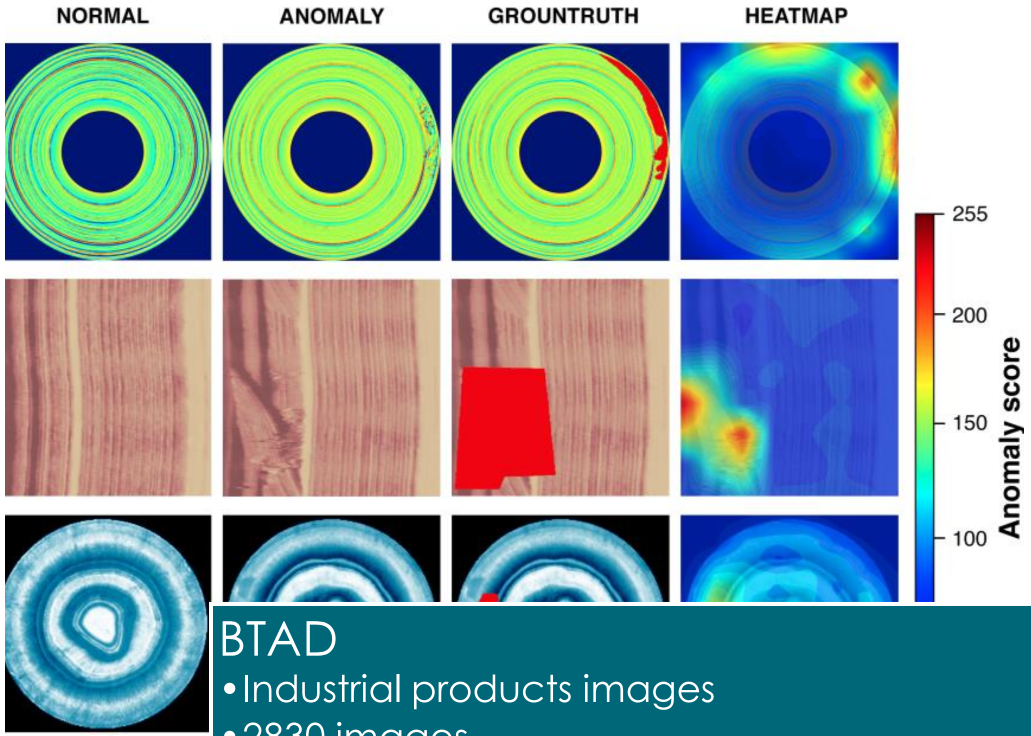
- Masked Auto Encoder
- Feature Pyramid
- Cross Attention
- Dual Training
- Creating Bad Reconstructions



Datasets



MVTecAD
- Benchmark for industrial applications
- Over 5000 images
- 15 classes, multiple kinds of defects per class



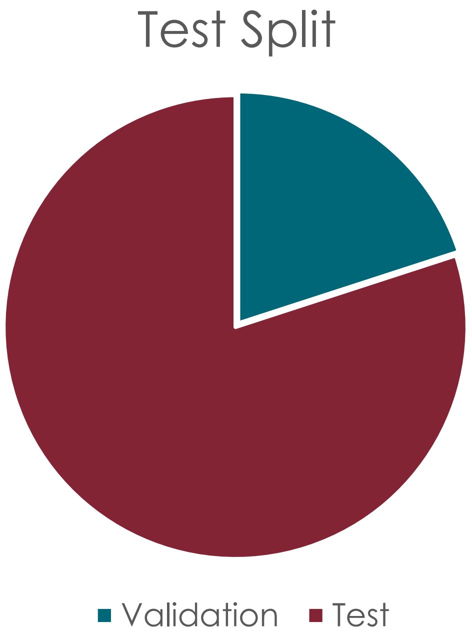
BTAD
- Industrial products images
- 2830 images
- 3 classes, every class has different size

The setup

MVTech AD

BTAD

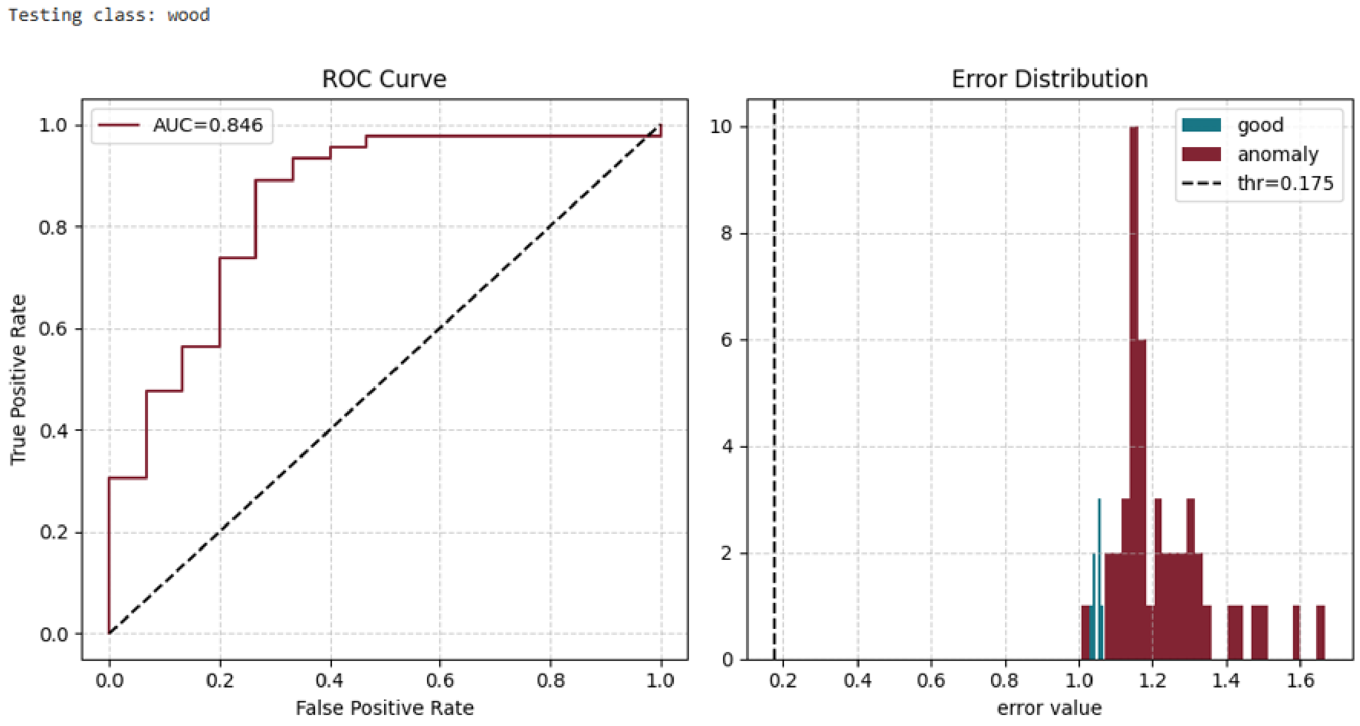
- Images resized to 256x256
- Encoder depth is 16
- Decoder depth is 2
- Drop rate=0.1
- Different embedding dimensions
- 75% of the image masked
- Trained only on good samples



Our seed obviously is:

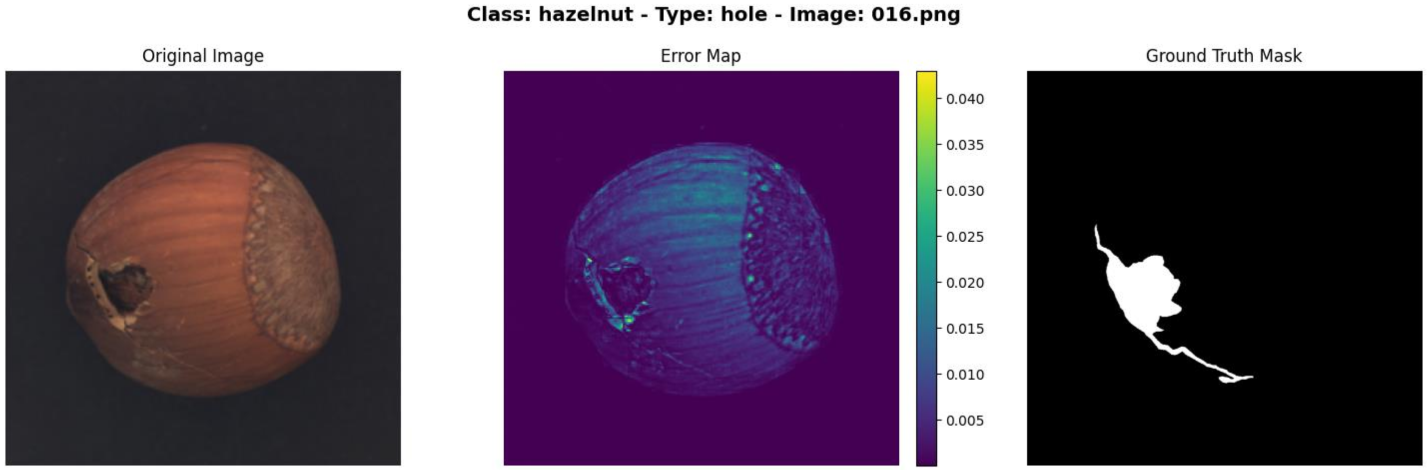
Evaluation methods

- AUROC
- PRO: Per Region Overlap
- $F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$

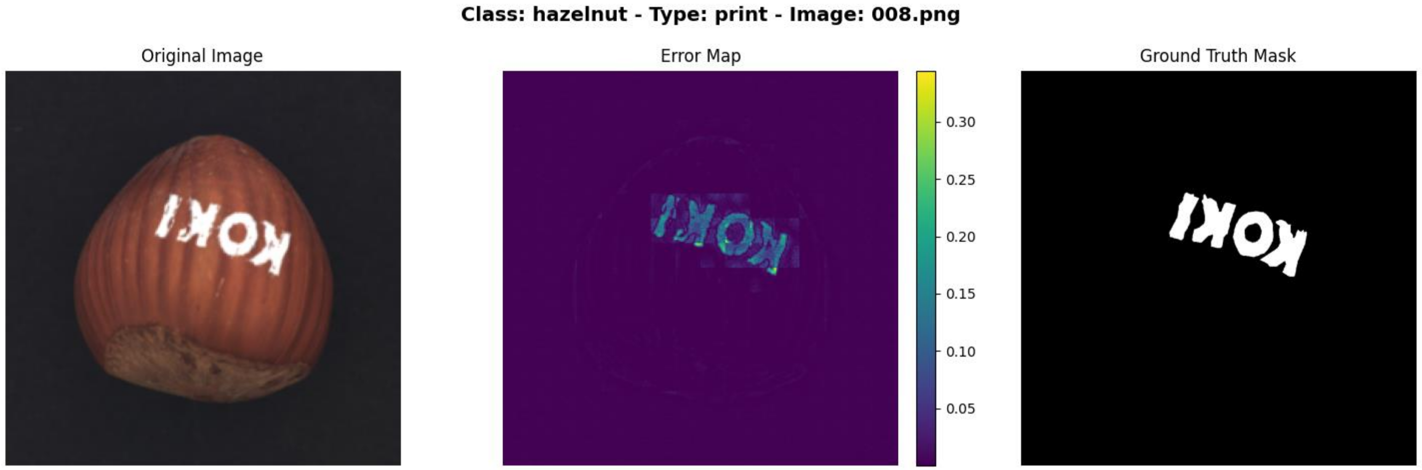


What did we accomplish?

- Just Pretraining



- With Fine Tuning



What did we accomplish?

```
classes = ['screw']  #classes 'toothbrush','transistor','wood','zipper'
dataloaders = create_class_dataloader(classes=classes,batch_size=32)
model = MAE(**hyperparameters)
val_loader_to_check = dataloaders['screw']['val']
mae_sanity_check(model, val_loader_to_check, device, hyperparameters)
```

```
Creating dataloader for class: screw
----- Model Sanity check started ----
Input check: OK
Output prediction shape check: OK
Model size: 113.62 MB (29.78 M params)
Model forward sanity check: PASSED 🎯

[INFO] Register count_convNd() for <class 'torch.nn.modules.conv.Conv2d'>.
[INFO] Register count_linear() for <class 'torch.nn.modules.linear.Linear'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.dropout.Dropout'>.

Model Flops with DyT using Dummy input
Model FLOPs: 2936012800.0 FLOPs
```

DyT FLOPs

```
classes = ['screw']  #classes 'toothbrush','transistor','wood','zipper'
dataloaders = create_class_dataloader(classes=classes,batch_size=32)
model = MAE(**hyperparameters)
val_loader_to_check = dataloaders['screw']['val']
mae_sanity_check(model, val_loader_to_check, device, hyperparameters)
```

```
Creating dataloader for class: screw
----- Model Sanity check started ----
Input check: OK
Output prediction shape check: OK
Model size: 113.62 MB (29.78 M params)
Model forward sanity check: PASSED 🎯

[INFO] Register count_convNd() for <class 'torch.nn.modules.conv.Conv2d'>.
[INFO] Register count_normalization() for <class 'torch.nn.modules.normalization.LayerNorm'>.
[INFO] Register count_linear() for <class 'torch.nn.modules.linear.Linear'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.dropout.Dropout'>.

Model Flops with LayerNorm using Dummy input
Model FLOPs: 2955149312.0 FLOPs
```

LayerNorm FLOPs

What did we accomplish?

```python
model = model.to(device)
model.eval()

# Define a dummy input tensor with the correct dimensions based on hyperparameters
dummy_input = torch.randn(1, hyperparameters['in_channels'], hyperparameters['image_size'], hy
input_image = dummy_input.to(device)

# warm-up (optional, for accurate timing on GPU)
for _ in range(5):
    _ = model(input_image)

# start timing
torch.cuda.synchronize() if device.type == 'cuda' else None
start_time = time.time()

# forward pass
with torch.no_grad():
    output = model(input_image)

torch.cuda.synchronize() if device.type == 'cuda' else None
end_time = time.time()

# inference time calculation
inference_time = end_time - start_time
print(f"GPU Inference time: {inference_time*10**3:.6f} milliseconds")
```

```
GPU Inference time: 13.448954 milliseconds
```

GPU inference

```python
# move model and input to CPU
device = torch.device('cpu')
model = model.to(device)
model.eval()

input_image = dummy_input.to(device)

for _ in range(5):
    _ = model(input_image)

start_time = time.time()

with torch.no_grad():
    output = model(input_image)

end_time = time.time()

inference_time = end_time - start_time
print(f"CPU inference time: {inference_time*10**3:.6f} milliseconds")
```

```
CPU inference time: 138.644695 milliseconds
```

CPU Inference

What did we accomplish?

Classe	AUC	F1-Score	AUPRO
Bottle	0.48	0.86	0.33
Cable	0.51	0.75	0.3
Capsule	0.58	0.9	0.21
Carpet	0.38	0.86	0.47
Grid	0.82	0.84	0.41
Hazelzut	0.81	0.77	0.72
Leather	0.68	0.85	0.46
Metal Nut	0.34	0.9	0.7
Pill	0.67	0.92	0.76
Tile	0.72	0.83	0.46
ToothBrush	0.37	0.84	0.74
Transistor	0.38	0.57	0.39
Wood	0.85	0.86	0.46
Zipper	0.46	0.88	0.39
01	0.25	0.83	0.65
02	0.71	0.93	0.43
03	0.44	0.17	0.58

Category	1-NN	OC SVM	VT-ADL (Ours)
Carpet	0.512	0.355	**0.773**
Grid	0.228	0.125	**0.871**
Leather	0.446	0.306	0.728
Tile	0.822	0.722	0.796
Wood	0.502	0.336	**0.781**
Bottle	0.898	0.85	**0.949**
Cable	0.806	0.431	0.776
Capsule	0.631	0.554	0.672
Hazelnut	0.861	0.616	0.897
Metal Nut	0.705	0.319	0.726
Pill	0.725	0.544	0.705
Screw	0.604	0.644	**0.928**
Toothbrush	0.675	0.538	**0.901**
Transistor	0.68	0.496	**0.796**
Zipper	0.512	0.355	0.808
Means	0.64	0.479	0.807

Some PRO Scores

Prdt	PRO Score ours	PR AUC ours	AE MSE	AE MSE+SSIM
0	*0.92*	0.99	0.49	0.53
1	*0.89*	0.94	0.92	0.96
2	*0.86*	0.77	0.95	0.89
Mean	*0.89*	**0.90**	0.78	0.79

TABLE IV

References

- Mishra, P., Verk, R., Fornasier, D., Piciarelli, C., & Foresti, G. L. (2021, June). VT-ADL: A Vision Transformer Network for Image Anomaly Detection and Localization. 2021 IEEE 30th International Symposiumon Industrial Electronics (ISIE), 01–06. doi:10.1109/isie45552.2021.9576231
- Zhu, J., Chen, X., He, K., LeCun, Y., & Liu, Z. (2025). Transformers without Normalization. arXiv [Cs.LG]. Retrieved from http://arxiv.org/abs/2503.10622
- Wenping Jin, Fei Guo, & Li Zhu. (2023). ISSTAD: Incremental Self-Supervised Learning Based on Transformer for Anomaly Detection and Localization. https://arxiv.org/abs/2303.17354
- Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, & Ross Girshick. (2021). Masked Autoencoders Are Scalable Vision Learners. https://arxiv.org/abs/2111.06377v2