Supplementary Material for PyramidFlow: High-Resolution Defect Contrastive Localization using Pyramid Normalizing Flow

1. Implementation Details

1.1. Experimental Settings

Hardware. We implemented our models in Python3.8 and Pytorch1.10 [7]. Experiments are run on NVIDIA GTX3060 GPUs.

Baseline method. We train our baseline model on 256×256 image. During all experiments, the training batch size is fixed to 2. Model parameters are updated using Adam optimizer [6] with a constant learning rate of 2×10^{-4}, epsilon of 1×10^{-4}, weight decay of 1×10^{-5}, and beta parameters of $(0.5, 0.9)$. In addition, we apply gradient clipping with a maximum gradient of 1.0 for training stability.

Pre-trained method. For the pre-trained version of PyramidFlow, we used ImageNet-pretrained ResNet18 from torchvision. The pre-trained encoder is the first two layers of ResNet18 for extracting the features from 1024×1024 image to 256×256 features with 64 channels.

1.2. Model Architecture

In this subsection, we provide the detailed architecture of the proposed PyramidFlow, including invertible pyramids, pyramid coupling blocks, and volume normalization.

Invertible Pyramid. The invertible pyramid is inspired by the Laplacian pyramid [2], which is commonly used in image processing. In invertible pyramids, the pyramid decomposition and composition are performed on the per-channel features. The linear downsampling operator $D(\cdot)$ first applies a Gaussian filter with kernel size 5×5, then downsamples using nearest-neighbor interpolation. In contrast, upsampling $U(\cdot)$ performs nearest-neighbor interpolation before applying Gaussian filtering.

Pyramid Coupling Block. For the example of dual coupling blocks, denoting the feature notations as shown in Fig. 3(d), the corresponding pseudocode is described in Algorithm 1. It is mainly composed of three custom functions - *AffineParamBlock, VolumeNorm2d,* and *InvConv*.

Volume Normalization. The proposed volume normalization is similar to some normalization techniques such as Batch Normalization [5], but without normalizing the standard deviation. Taking Channel Volume Normalization (CVN) as an example, it can be described by the Algorithm 2.

2. More Experiment Results

2.1. Detailed Ablation Results

We present the detailed ablation results of Sec 4.3, as shown in Tables S1 and S2.

Textural Image. As shown in Table S1. For most textural categories, occurring performance degradation when the proposed methods are ablated. However, the results on the carpet show abnormal performance improvement. This means that inductive bias brings positive or negative effects on various categories.

Object Image. As shown in Table S2. Due to the image patch in object categories with larger variances, the influence of volume normalization and the latent template is also larger. The performance of the object categories is less influenced by pyramid difference, indicating that multi-scale is not a critical factor for object defect detection.

Algorithm 1 Dual Coupling Block. (Python-like Pseudocode)

```
Input: x_0, x_1, x_2
Output: z_0, z_1, z_2
  x_cat0 = Interpolate(x_0, x_1.shape)
  x_cat2 = Interpolate(x_2, x_1.shape)
  x_cat = Concat(x_cat0, x_cat2)
  s_1, t_1 = AffineParamBlock(x_cat)
  y_1 = exp(s_1) ⊙ x_1 + t_1
  z_0, z_1, z_2 = x_0, InvConv(y_1), x_2

  def AffineParamBlock(x, clamp=2):
    params = CNN2d(x) % only two convolutional layers and one
  activation layer
    s_0, t = Chunk2d(params)
    s = VolumeNorm2d(clamp*0.636*atan(s_0/clamp)) % as shown in
  Algorithm 2.  Where 0.636 is an approximation of 2/π.
    return s,t

  def InvConv(y):
    s̃_i = s_i - mean(s_i)
    kernel = PL(U + diag(exp(s̃_i))
    z = Conv2d(y, kernel)
    return z
```

Algorithm 2 Volume Normalization. (Pytorch-like Pseudocode)

```
Input: input x, momentum β
Output: output y
  def VolumeNorm2d(x, β = 0.1):
    if training:
      x̄ = mean(x, dim=1) % CVN: zero-mean normalization along
  channel dimensions
      y = x - x̄
      x̄_running = (1−β) ×x̄_running + β × x̄ % update running mean
    else:
      y = x - x̄_running
    return  y
```

Table S1. The ablation study on textural images in MVTecAD. For each cell in the table, the first row is Pixel-AUROC% and the second is AUPRO% .

Method	Texture					Mean
	carpet	grid	leather	tile	wood	
Ours(baseline)	90.8	94.2	99.6	97.9	93.8	95.2
	91.0	92.7	99.7	95.8	96.2	95.1
I. w/o Volume Normalization	93.5	88.5	99.5	74.4	91.3	89.4
	93.7	88.1	95.5	65.7	94.2	87.5
II. w/o Latent Template	91.8	86.8	99.4	94.8	93.0	93.1
	91.3	88.0	97.7	89.9	92.7	91.9
III. w/o Pyramid Difference	75.9	78.0	99.3	96.0	89.7	87.8
	76.1	76.1	99.4	94.4	93.0	87.8
IV. w/o Fourier Loss	90.5	84.3	99.4	96.2	89.7	92.0
	91.4	86.2	99.6	92.6	94.0	92.8

Table S2. The ablation study on object images in MVTecAD. For each cell in the table, the first row is Pixel-AUROC% and the second is AUPRO% .

Method	Object										Mean
	bottle	cable	capsule	hazelnut	metalnut	pill	screw	toothbrush	transistor	zipper	
Ours(baseline)	95.9	92.1	96.1	98.0	92.8	96.2	94.0	98.9	97.4	95.4	95.7
	94.0	86.4	93.1	97.3	89.5	96.3	94.1	97.9	91.4	95.1	93.5
I. w/o Volume Normalization	76.5	84.7	82.9	97.9	87.9	94.8	94.1	56.4	82.2	95.0	85.2
	77.8	75.1	81.3	95.4	81.5	81.5	94.0	74.2	82.7	92.6	83.6
II. w/o Latent Template	83.2	87.8	90.0	97.9	87.6	94.6	93.0	84.7	94.8	93.7	90.7
	82.4	76.6	87.3	83.9	74.2	89.3	92.7	90.7	77.4	92.8	84.7
III. w/o Pyramid Difference	92.8	91.4	96.0	97.5	86.4	95.3	92.7	98.0	95.4	85.2	93.1
	83.4	84.1	94.0	97.6	81.2	95.4	93.1	97.1	90.7	77.2	89.4
IV. w/o Fourier Loss	88.0	88.6	95.1	97.3	88.9	96.2	94.2	98.3	95.1	90.9	93.3
	88.0	81.2	94.0	98.3	89.0	96.9	94.4	97.9	88.0	90.8	91.9

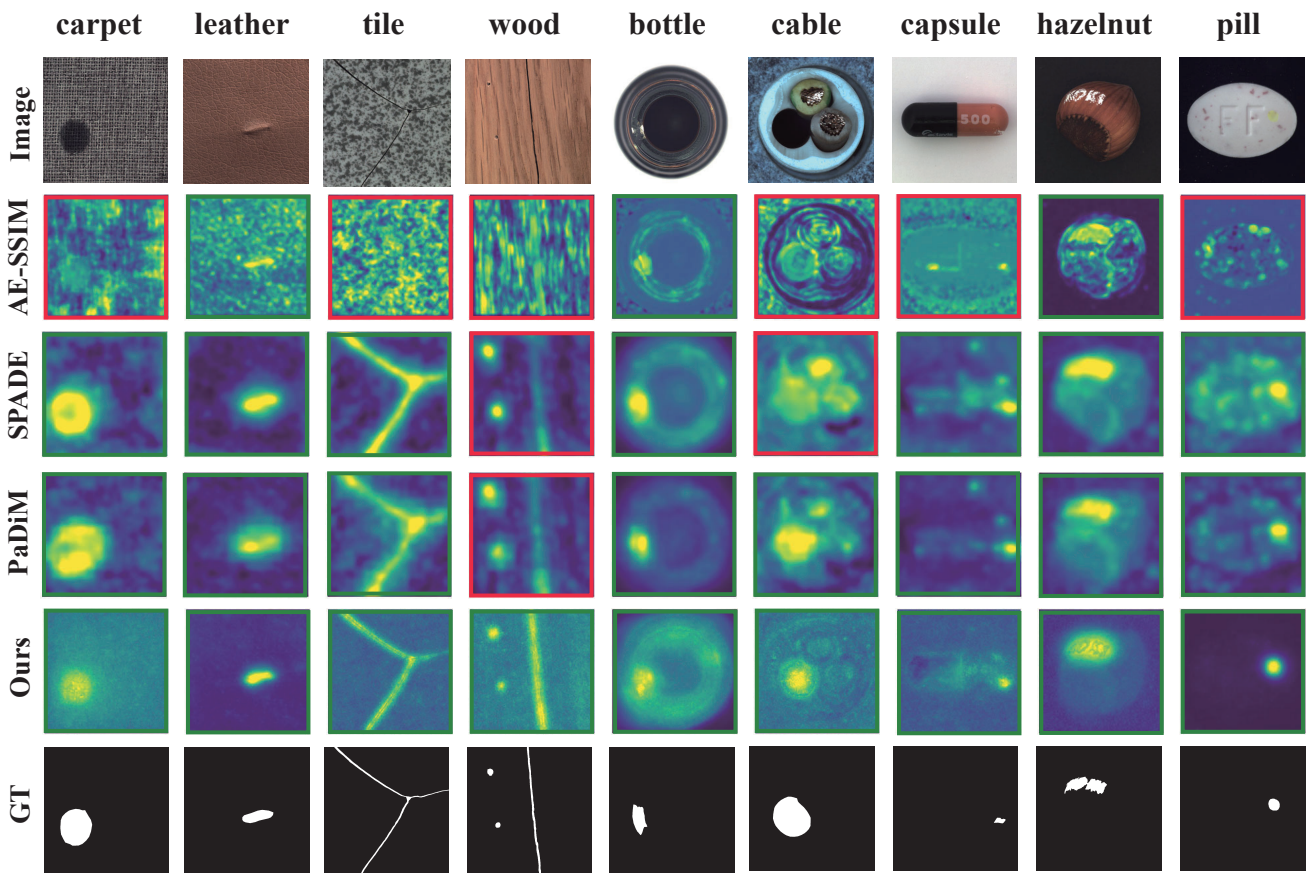

Figure S1. Visualization of competitive results on MVTecAD. From top to bottom are original images, AE-SSIM [1] results, SPADE [3] results, PaDiM [4] results, our results, and ground truths. The red box indicates the localization is ambiguous and non-unique, while the green indicates successful results.

2.2. More Visualization Results

In this subsection, we present more visualization results of Sec 4.4. Since many categories, we separated results into two charts for visualization, as shown in Figs. S1 and S2.

MVTecAD. As Figs. S1 and S2 shows, AE-SSIM performs better for simple categories, such as the bottle and zipper. However, it does not work in complex scenarios, *e.g.*, it cannot localize carpet defects with fixed patterns or pill defects with high-frequency noises. It is worth noticing that AE-SSIM is a template-based method, which maintains the resolution during processing, enabling preserve the details in defect localization.

SPADE and PaDiM are pre-trained-based methods. They achieve better results in almost all categories but still maintain

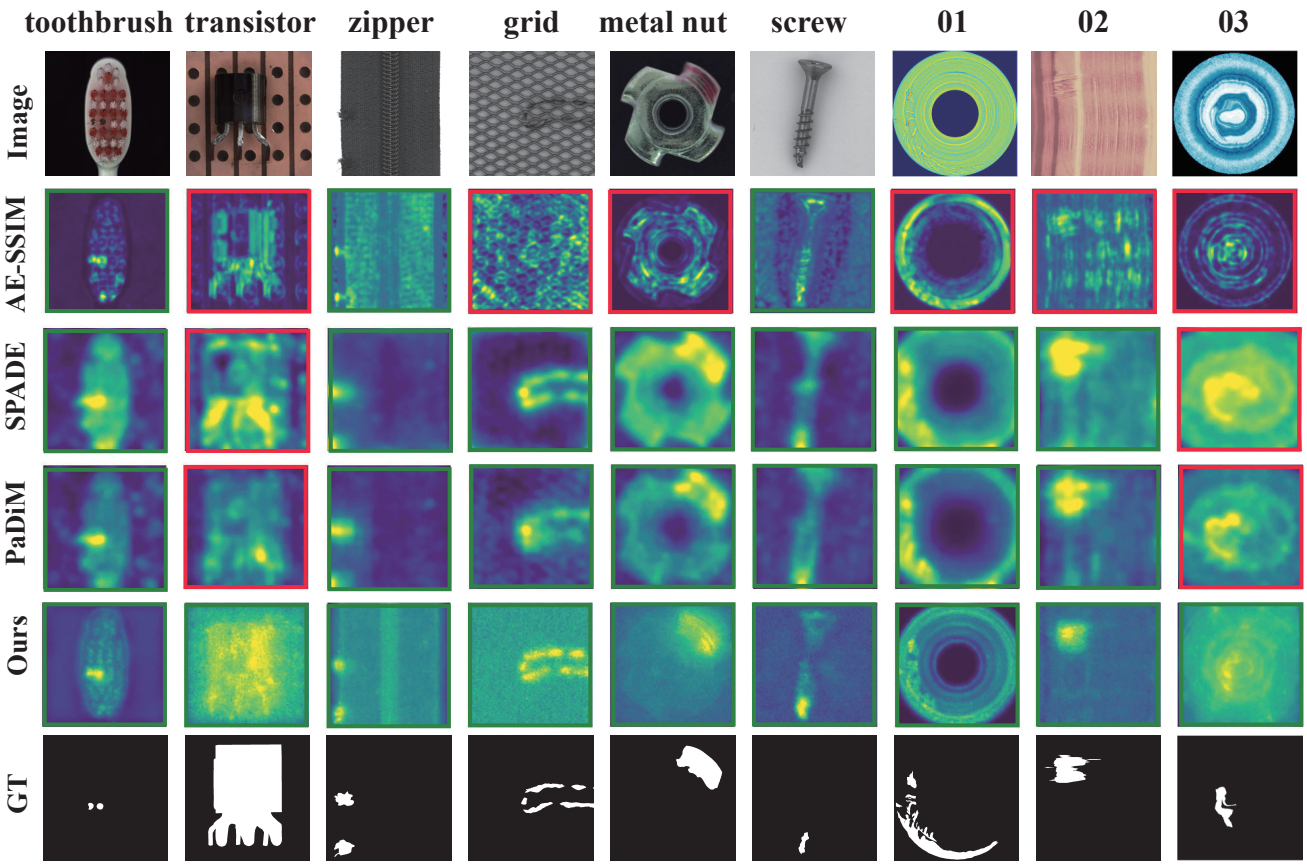

Figure S2. Visualization of competitive results on MVTecAD and BTAD. From top to bottom are original images, AE-SSIM [1] results, SPADE [3] results, PaDiM [4] results, our results, and ground truths. The last three columns are the results of BTAD. The red box indicates the localization is ambiguous and non-unique, while the green indicates successful results.

some shortcomings. On the one hand, their localization results are blurry and larger than ground truths. On the other hand, they cannot localize tiny defects, such as cracks in the wood.

Our proposed PyramidFlow is based on latent templates, which allows for preserving details effectively, with the ability to detect tiny defects and show their scale. In all categories in MVTecAD, our method achieves the best visual performance.

BTAD. BTAD is more challenging than MVTecAD, as shown in the last three columns of Fig. S2. The AE-SSIM method almost failed in BTAD without beneficial results. For categories 01 and 02, the localization areas of SPADE and PaDiM are obviously larger than ground truths. For the most challenging category 03, their results are incredibly varied from GT.

Our method provides more accurate results for BTAD defect localization. For the 01 categories, the localization results preserve the original details. Categories 02 and 03 also mostly reflect the essential shape of the defect.

References

[1] Paul Bergmann, Sindy Löwe, Michael Fauser, David Sattlegger, and Carsten Steger. Improving unsupervised defect segmentation by applying structural similarity to autoencoders. In *VISIGRAPP (5: VISAPP)*, 2019.

[2] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, 1983.

[3] Niv Cohen and Yedid Hoshen. Sub-image anomaly detection with deep pyramid correspondences. *CoRR*, abs/2005.02357, 2020.

[4] Thomas Defard, Aleksandr Setkov, Angelique Loesch, and Romaric Audigier. Padim: a patch distribution modeling framework for anomaly detection and localization. In *International Conference on Pattern Recognition*, pages 475–489. Springer.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[7] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.