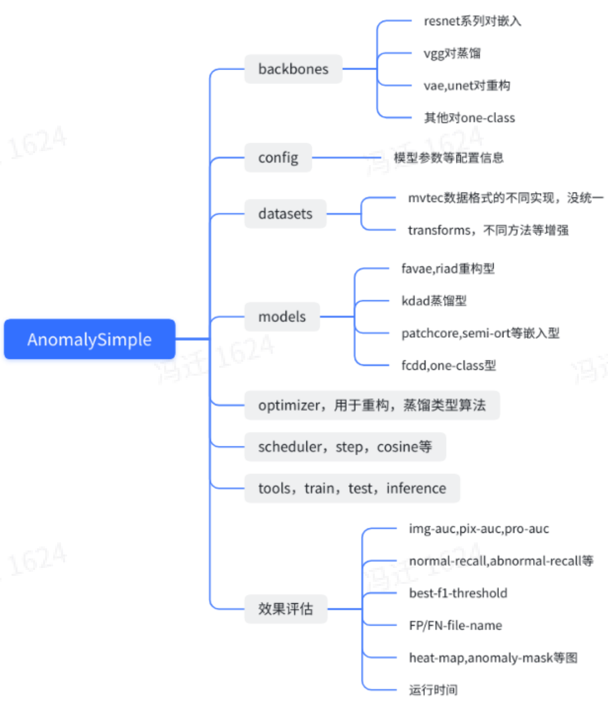
G100算法，代码简要说明

- 文章
  - https://www.sciencedirect.com/science/article/abs/pii/S0031320320305094 riad
  - https://arxiv.org/pdf/2008.05369.pdf favae
  - https://arxiv.org/pdf/2011.11108.pdf kdad
  - https://arxiv.org/pdf/2106.08265.pdf patchcore
  - https://arxiv.org/pdf/2105.14737.pdf semi-ort
- 代码结构



- 算法核心思想
  - Riad
    - 原始自编码重构泛化性太强，将问题转化为Reconstruction-by-inpainting anomaly detection，给原图打不相交的码，然后分别重构打码的部分，然后合并，网络结构Unet。
    - 损失函数： $L=\lambda_G L_G+\lambda_S L_S+L_2$,G多尺度梯度相似度，S：SSIM图像结构相似度函数，L2：二范数。Structural_similarity(SSIM) ,MSGMS:

      $g(\mathbf{I})=\sqrt{(\mathbf{I}*\mathbf{h}_x)^2+(\mathbf{I}*\mathbf{h}_y)^2}\quad \mathrm{GMS}(\mathbf{I},\mathbf{I}_r)=\dfrac{2g(\mathbf{I})g(\mathbf{I}_r)+c}{g(\mathbf{I})^2+g(\mathbf{I}_r)^2+c}$,L2。
    - 推理：异常分数GMS，异常区域分割：GMS后的分数做gaussian_filter，以及归一化计算阈值，分割区域做开运算。

- 实验参数：训练数据增加了水平垂直翻转，k:[2,4,8,16], n=3，k原图分成的网格大小(随机)，n，每批次生成的不相交的掩码个数。每个掩码大小k*k......

Python

```python
img_size = data.size(-1)
k_value = random.sample(self.k_value, 1).  # self.k_value = [2, 4, 8, 16]
Ms_generator = gen_mask(k_value, 3, img_size)
Ms = next(Ms_generator)  # 不相交的mask==0,需要被重构

inputs = [data * (torch.tensor(mask, requires_grad=False).to(self.device)) for mask in Ms]
outputs = [self.model(x) for x in inputs]
output = sum(map(lambda x, y: x * (torch.tensor(1 - y, requires_grad=False).to(self.device)), outputs, Ms))  # 将重构的部分相加

l2_loss = mse(data, output)
gms_loss = msgms(data, output)
ssim_loss = ssim(data, output)

loss = gamma * l2_loss + alpha * gms_loss + belta * ssim_loss

def predict(self, test_data, **kwargs):
    self.model.eval()
    msgms_score = MSGMS_Score()
    score = 0
    with torch.no_grad():
        data = test_data.to(self.device)
        for k in self.k_value:
            img_size = data.size(-1)
            N = img_size // k
            Ms_generator = gen_mask([k], 3, img_size)
            Ms = next(Ms_generator)
            inputs = [data * (torch.tensor(mask, requires_grad=False).to(self.device)) for mask in Ms]
            outputs = [self.model(x) for x in inputs]
            output = sum(map(lambda x, y: x * (torch.tensor(1 - y, requires_grad=False).to(self.device)), outputs, Ms))
            score += msgms_score(data, output) / (N**2)

    score = score.squeeze().cpu().numpy()
    if score.ndim < 3:
        score = np.expand_dims(score, axis=0)
    for i in range(score.shape[0]):
        score[i] = gaussian_filter(score[i], sigma=7)

    if (self.val_max_as is not None) and (self.val_min_as is not None):
        # print('Normalizing!')
        score = (score - self.val_min_as) / (self.val_max_as - self.val_min_as)

    img_score = score.reshape(score.shape[0], -1).max(axis=1)

    return img_score, score
```

- Patchcore
  - 特征嵌入算法，主要贡献点：
    - Locally aware patch features：多尺度局部特征融合
    - Coreset-reduced patch-feature memory bank：保持结构信息不变的情况下减少正常样本池的数量Coreset-reduced 问题 $\mathcal{M}_C^* = \underset{\mathcal{M}_C \subset \mathcal{M}}{\arg\min} \max_{m \in \mathcal{M}} \min_{n \in \mathcal{M}_C} \|m-n\|_2$ ，同时做了维度降维映射。Johnson-Lindenstrauss theorem。
    - 异常分数： $s = \left(1 - \frac{\exp \|m^{\text{test},*} - m^*\|_2}{\sum_{m \in \mathcal{N}_b(m^*)} \exp \|m^{\text{test},*} - m\|_2}\right) \cdot s^*$ ，其中

$$m^{\text{test},*}, m^* = \underset{m^{\text{test}} \in \mathcal{P}(x^{\text{test}})}{\arg\max} \ \underset{m \in \mathcal{M}}{\arg\min} \left\|m^{\text{test}} - m\right\|_2$$
$$s^* = \left\|m^{\text{test},*} - m^*\right\|_2$$
。

- 整体结构：

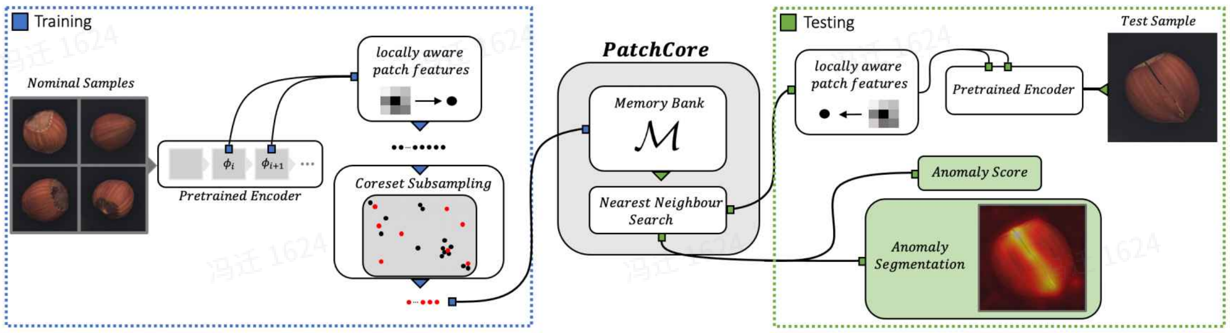


Figure 2: Overview of *PatchCore*. During training, nominal samples are broken down into a memory bank of neighbourhood-aware patch-level features. For reduced redundancy and inference time, this memory bank is downsampled via greedy coreset subsampling. At test time, images are classified as anomalies if at least on patch is anomalous, and pixel-level anomaly segmentation is generated by scoring each patch-feature.

Python

```python
# 多尺度局部特征融合
def embedding_concat(x, y):
    # from https://github.com/xiahaifeng1995/PaDiM-Anomaly-Detection-Localization-master
    B, C1, H1, W1 = x.size()
    _, C2, H2, W2 = y.size()
    s = int(H1 / H2)
    x = F.unfold(x, kernel_size=s, dilation=1, stride=s)
    x = x.view(B, C1, -1, H2, W2)
    z = torch.zeros(B, C1 + C2, x.size(2), H2, W2)
    for i in range(x.size(2)):
        z[:, :, i, :, :] = torch.cat((x[:, :, i, :, :], y), 1)
    z = z.view(B, -1, H2 * W2)
    z = F.fold(z, kernel_size=s, output_size=(H1, W1), stride=s)

    return z

# 特征池缩小和降维
from anomaly.utils import kCenterGreedy # 欧式距离
from sklearn.random_projection import SparseRandomProjection

# 推理
# from sklearn.neighbors import NearestNeighbors
# nbrs = NearestNeighbors(n_neighbors=self.args.n_neighbors, algorithm='ball_tree', metric='minkowski', p=2).fit(self.embedding_coreset) # p=2,欧式
# score_patches, _ = nbrs.kneighbors(embedding_test)

#Cuda版本
knn = KNN(torch.from_numpy(self.embedding_coreset).cuda(), k=9)
score_patches = knn(torch.from_numpy(embedding_test).cuda())[0].cpu().detach().numpy()

# 异常分数
anomaly_map = score_patches[:,0].reshape((28,28))   # 28x28
N_b = score_patches[np.argmax(score_patches[:,0])]  # ^{*}=d(m^{test} - m^{*})
w = (1 - (np.max(np.exp(N_b))/np.sum(np.exp(N_b))))
score = w*max(score_patches[:,0]) # Image-level score :w*s^{*}, 不明原因

gt_np = gt.cpu().numpy()[0,0].astype(int)
anomaly_map_resized = cv2.resize(anomaly_map, (self.args.input_size, self.args.input_size))
anomaly_map_resized_blur = gaussian_filter(anomaly_map_resized, sigma=4)
```

- Semi-orthogonal
  - 特征融合比patchcore多一个尺度

- 将特征空间做一个半正定矩阵映射
  - 特征降秩，降低协方差矩阵的求逆时间
- 计算正异样本的特征向量的马氏距离。
- 参数：k=100

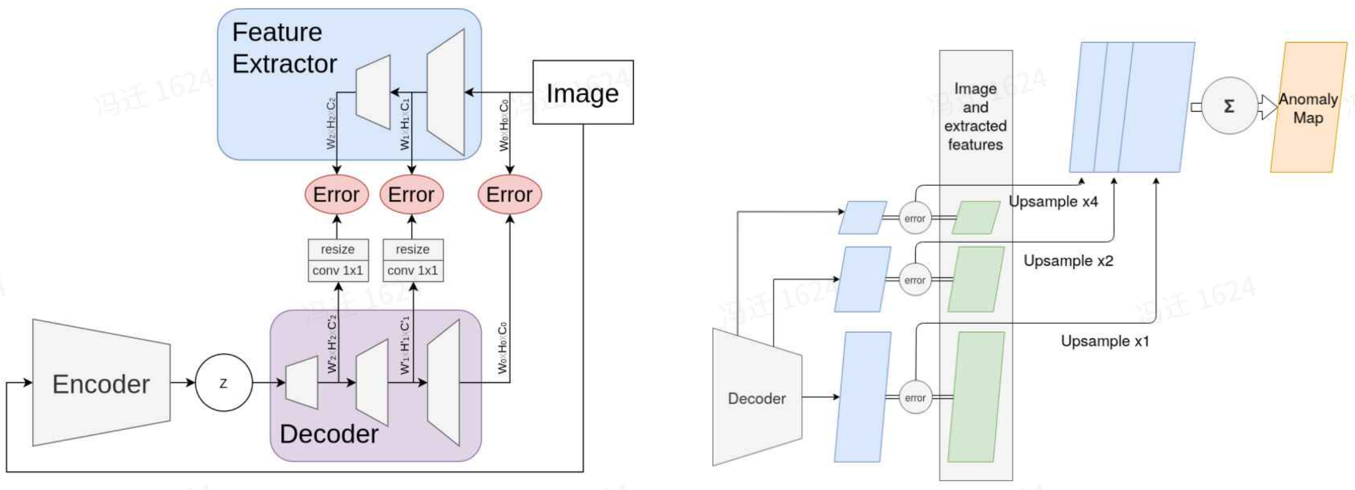
Theorem 1. (Low-rank embedding of precision matrix) Let $\mathbf{W} \in \mathbb{R}^{F\times k}$, where $k \le \min(F, N)$ be a low-rank embedding matrix. Then, we have that:

$\mathbf{U}_k \in \arg\min_{\mathbf{W}} \left\| \mathbf{C}^{-1} - \mathbf{W}\left(\mathbf{W}^\top \mathbf{C}\mathbf{W}\right)^{-1}\mathbf{W}^\top \right\|^2$ ，其中 $\mathbf{C} \in \mathbb{R}^{F\times F} = X * X^T, X \in \mathbb{R}^{F\times N}$ 。

Python
```python
# 半正定矩阵
def _generate_W(F: int, k: int, device: str):
    omega = torch.randn((F, k), device=device)
    q, r = torch.linalg.qr(omega)
    W = q @ torch.sign(torch.diag(torch.diag(r)))
    return W

# 嵌入
embeddings = torch.matmul(self.W.T, embeddings)  # tensor of size W, H, k, N

# 特征向量的均值和方差
b = embeddings.size(3)
embeddings = embeddings.reshape(
    (self.num_patches, self.k, -1))  # (w * h) * k * b
self.covs += torch.einsum("wib,wjb->wij", embeddings, embeddings)
self.means += torch.einsum("wib->wi", embeddings)
self.N += b  # number of images

# 马氏距离
distances = mahalanobis_sq(embeddings, means, inv_cvars) # 此处实现为torch版本

# 响应度高斯平滑，分数归一化
gaussian_smoothing = transforms.GaussianBlur(9)

amaps = torch.cat(amaps)
amaps = F.interpolate(amaps, size, mode="bilinear", align_corners=True)
amaps = gaussian_smoothing(amaps)
amaps -= amaps.min()
amaps /= amaps.max()
amaps = amaps.squeeze().cpu().numpy()
```

- FAVAE

- 结构



(a) Architecture of the model (b) Anomaly map computations

Figure 2: Illustrations of the model (a) and the anomaly map computations (b) for $L = 2$

- 利用预训练模型的感知更接近人对异常区域的定义，增加了imagenet上的vgg预训练模型，做为解码特征层的的限制，同时保持原始编解码和输入的损失。
- 假设正常样本的编码特征隐变量符合多元0-1高斯分布
- 异常检测：有解码重构损失和预训练的特征层损失，同时增加了patch处理。

Python

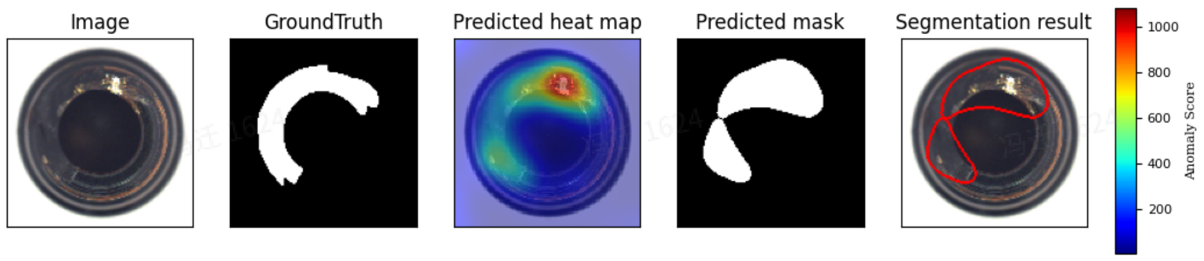
```python
def predict(self, test_data, **kwargs):
    self.model.eval()
    self.teacher.eval()
    MSE_loss = nn.MSELoss(reduction='none')
    score = 0
    with torch.no_grad():
        data = test_data.to(self.device)
        img_size = data.size(-1)
        assert (img_size >= self.crop_size), 'Input size should not be smaller than the crop size.'
        if img_size > self.crop_size:
            data = get_patch(data, self.crop_size)    # patch处理，原图分成块，并在新维度上合并
            # print(data.shape)
            z, output, _, _ = self.model(data)
            # print(output.shape)
            # get model's intermediate outputs
            s_activations, _ = self._feature_extractor(z, self.model.decode, target_layers=['11', '17', '23'])
            t_activations, _ = self._feature_extractor(data, self.teacher.features, target_layers=['7', '14', '21'])
            score = MSE_loss(output, data).sum(1, keepdim=True)
            for i in range(len(s_activations)):
                s_act = self.model.adapter[i](s_activations[-(i + 1)])
                mse_loss = MSE_loss(s_act, t_activations[i]).sum(1, keepdim=True)
                score += F.upsample(mse_loss, size=data.size(2), mode='bilinear', align_corners=False)
                # 将对应特征与预训练模型的特征的差异叠加到重构差异上，类似与蒸馏做法，但蒸馏但异常检测可用梯度反向传播但幅值来刻画
            # print(score.shape)
            output = patch2img(output.cpu(), img_size, self.crop_size)   # 同时做了分块处理，增加了小异常和异常但检出率
            # print(output.shape)
            score = patch2img(score.cpu(), img_size, self.crop_size)

        else:
            z, output, _, _ = self.model(data)
            # get model's intermediate outputs
            s_activations, _ = self._feature_extractor(z, self.model.decode, target_layers=['11', '17', '23'])
            t_activations, _ = self._feature_extractor(data, self.teacher.features, target_layers=['7', '14', '21'])

            score = MSE_loss(output, data).sum(1, keepdim=True)
            for i in range(len(s_activations)):
                s_act = self.model.adapter[i](s_activations[-(i + 1)])
                mse_loss = MSE_loss(s_act, t_activations[i]).sum(1, keepdim=True)
                score += F.upsample(mse_loss, size=data.size(2), mode='bilinear', align_corners=False)

    score = score.squeeze().cpu().numpy()
    if score.ndim < 3:
        score = np.expand_dims(score, axis=0)
    for i in range(score.shape[0]):
        score[i] = gaussian_filter(score[i], sigma=7)   # 高斯处理

    if (self.val_max_as is not None) and (self.val_min_as is not None):
        # print('Normalizing!')
        score = (score - self.val_min_as) / (self.val_max_as - self.val_min_as)   # 归一化

    img_score = score.reshape(score.shape[0], -1).max(axis=1)

    return img_score, score
```

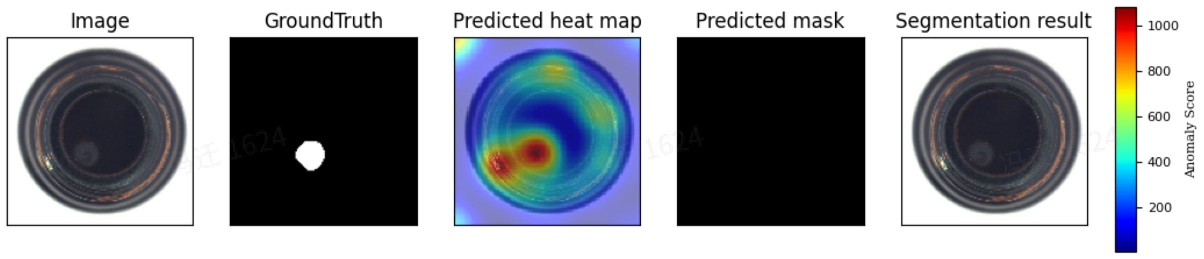
- mvtec数据性能局部示意：

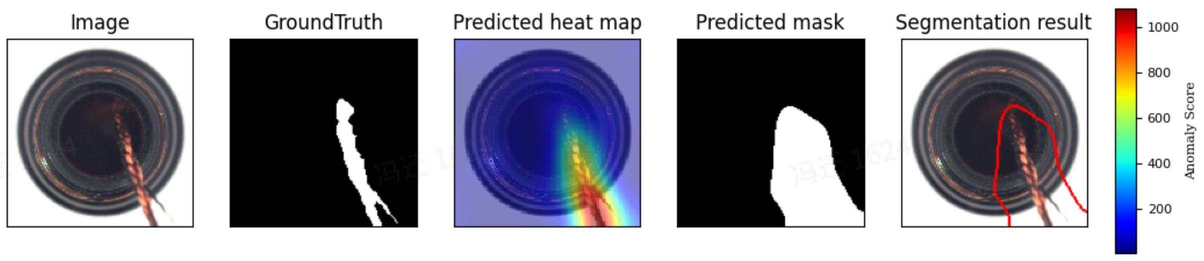
算法	threshol	noraml_r	abnormal_r	precision_	image	pixel	class	time/im

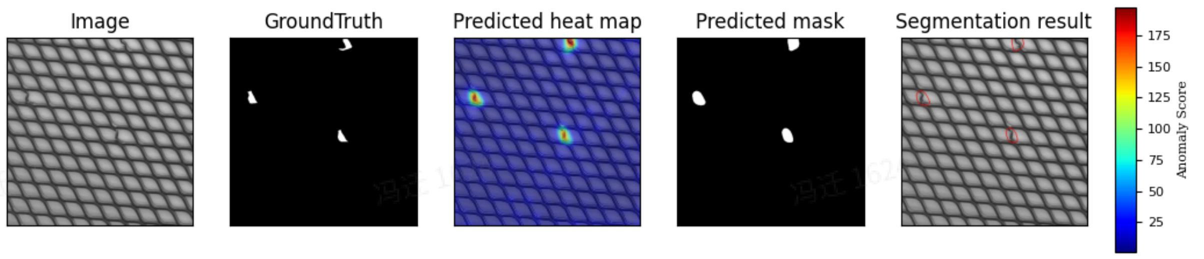
	d	ecall	ecall	img	ROCAUC	ROCAUC		g
favae	0.698/0.323	1	0.968	0.976	0.960	0.995	bottle	0.0476s batch=24
riad	0.47/0.53	0.85	0.936	0.916	0.959	0.929	bottle	0.34s/0.19(batch 4,24)
patchcore	2.044139	0.7143	0.9474	0.8846	0.9457	0.9578	grid	0.425s+0.2
semi-ort	12.16	0.95	0.968	0.964	0.989	0.9864/pro	bottle	0.146s-0.01
kdad								
fcdd								

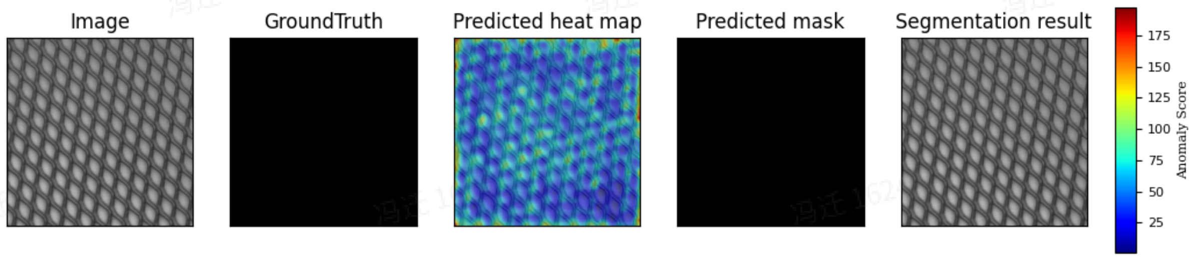
- favae算法结果示例：
-











- 特点：
    - 增加异常定位：多尺度局部融合；patch处理；同构L2损失+蒸馏损失；同构L2损失+自定义相似梯度损失；蒸馏+梯度反传；多元高斯分布+马氏距离；最后高斯平滑+分数归一化。
- 算法优化方向：
    - 增加实验，favae,semi...