Wood_LE_2022

一、Paper Reading

前置：标题、摘要、导言、方法、实验、结论。

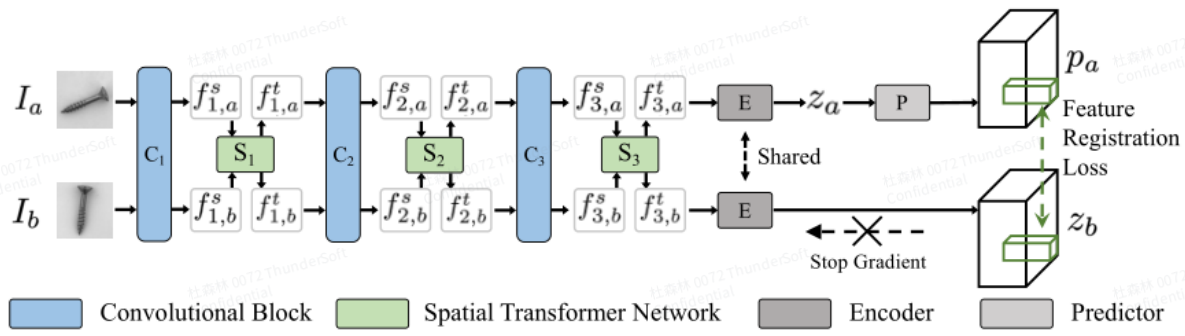
First-screening papers： 标题、摘要、结论、论文中的图片和表格。

Second-Read full non-depth: 厘清文章脉络，知道在讲什么、怎么做的、做的怎么样，同时也决定你是否要读第三遍。

Third-真正读懂。

1、Registration based Few-Shot Anomaly Detection: 无需微调即可推广，上交大、上海人工智能实验室等提出基于配准的少样本异常检测框架

RegAD的模型架构：



The model architecture of the proposed RegAD. Given paired images from the same category, features are extracted by three convolutional residual blocks each followed by a spatial transformer network. A Siamese network acts as the feature encoder, supervised by a registration loss for feature similarity maximization （给定来自同一类别的图像对，通过三个卷积残差块提取特征，每个残差块后跟一个空间变换网络。孪生网络充当特征编码器， 由配准损失来监督特征相似度最大化）.

复现：

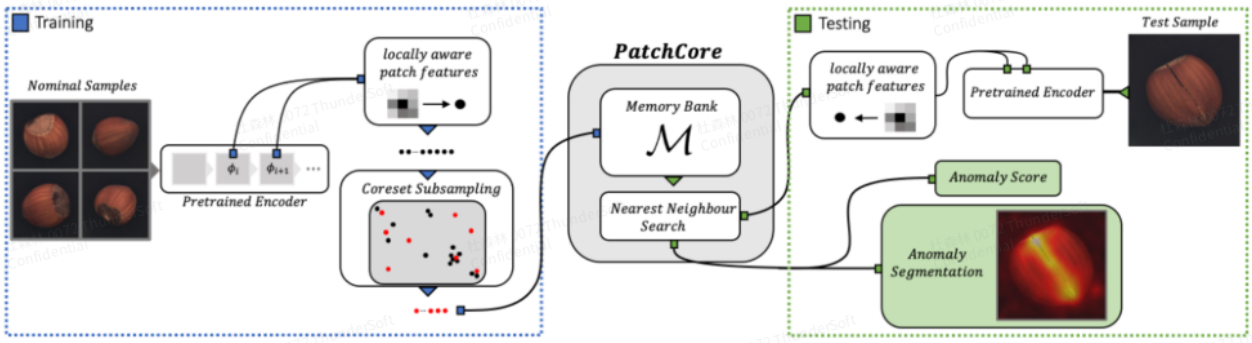
2、MsAeDefectDetector 检测异常

😶 参考tscv和tsnn实现代码 理解和复现

可参考： http://192.168.87.232:10040/tree/project/r/wood/Work_Project/demoapp/Wood/research/Pytorch

3、PatchCore: Towards Total Recall in Industrial Anomaly Detection 工业异常检测中的全面召回

PatchCore 模型架构：



Python

```
Patch：所谓补丁，指的是像素；
Embedding：所谓嵌入，指的是将网络提取的不同特征组合到一块；

训练流程：无神经网络参数更新，不需要训练，不需要更新参数 requires_grad==false，只需要使用预训练模型提取特征，构建 Memory Bank 相当于使用正常数据进行模型训练、构建。

推理流程：提取Patch特征，再Faiss搜索。
```

Pretrained Encoder: 使用预训练模型（wide_resnet50_2） backbone 提取图像特征，采用[2, 3]层特征作为图像特征，具有较强的底层特征（轮廓、边缘、颜色、纹理和形状特征），更能够反映图像内容。不采用最后几层原因：深层特征偏向于分类任务，具有更强的语义信息。

Locally aware patch features：提取图像的 Patch特征，这个特征带有周围数据的信息。特征值的集合构建 PatchCore memory bank。

Python

```
for feature in features:
    m = torch.nn. AvgPool2d(3, 1, 1)
    embeddings.append(m(feature)) # 特征值：以（h,w)为中心的邻居点集得到，论文中说使用adaptive average pooling. 实验使用AvgPool2d，指标更高。
# embedding_concat：特征连接，参考 https://github.com/xiahaifeng1995/PaDiM-Anomaly-Detection-Localization-master
embedding = embedding_concat(embeddings[0], embeddings[1])
# reshape：维度变换
self.embedding_list.extend(reshape_embedding(np.array(embedding)))
```

1、PatchCore与PaDiM对比: patchcore 使用高效的patch-feature memory bank, 在测试时所有patch都可以访问该内存库。 PaDiM 是针对每个 patch 的马氏距离度量。 相比之下， PatchCore对图像对齐的依赖性降低。

2、特征提取：特征表示为何不选择网络特征层次的最后一级：（1）会丢失更多的局部正常样本信息；（2）深层特征具有更强的语义信息，偏向于分类任务。

3、patch特征：可以理解为 训练图片上所有的点，以该点为中心的邻居点集得到的特征值，特征值的集合就是 PatchCore memory bank。

以(h , w) 为中心，以p 为直径的正方形包围住的点；

$$\mathcal{N}_p^{(h,w)} = \{(a,b)|a \in [h - \lfloor p/2 \rfloor, ..., h + \lfloor p/2 \rfloor], \\ b \in [w - \lfloor p/2 \rfloor, ..., w + \lfloor p/2 \rfloor]\},$$

围绕这些点计算的特征图上的点为：

$$\phi_{i,j}\left(\mathcal{N}_p^{(h,w)}\right) = f_{\text{agg}}\left(\{\phi_{i,j}(a,b)|(a,b) \in \mathcal{N}_p^{(h,w)}\}\right), \tag{2}$$

一张图像的Patch特征集合：

$$\begin{aligned}\mathcal{P}_{s,p}(\phi_{i,j}) = \{&\phi_{i,j}(\mathcal{N}_p^{(h,w)})| \\ &h, w \bmod s = 0, h < h^*, w < w^*, h, w \in \mathbb{N}\},\end{aligned} \tag{3}$$

正常训练集图像的Patch特征集合：

$$\mathcal{M} = \bigcup_{x_i \in \mathcal{X}_N} \mathcal{P}_{s,p}(\phi_j(x_i)). \tag{4}$$

Coreset-reduced patch-feature memory bank：Coreset Subsampling 核心集二次抽样:稀疏采样目的是Reduce memory bank，加快算法运行速度。

PHP

```
# Memory Bank:将收集到的正常图像 Patch 特征放入 MemoryBank
total_embeddings = np.array(self.embedding_list)
# Random projection  随机投影
self.randomprojector = SparseRandomProjection(n_components='auto', eps=0.9) #
'auto' => Johnson-Lindenstrauss lemma
self.randomprojector.fit(total_embeddings)
# Coreset Subsampling 核心集二次抽样:稀疏采样 Reduce memory bank
# 参考
 https://github.com/google/active-learning/blob/master/sampling_methods/kcenter_

selector = kCenterGreedy(total_embeddings,0,0)
selected_idx = selector.select_batch(model=self.randomprojector,
already_selected=[],
N=int(total_embeddings.shape[0]*args.coreset_sampling_ratio))
self.embedding_coreset = total_embeddings[selected_idx]
```

NP-Hard问题，建议使用迭代贪婪近似。

$$\mathcal{M}_C^* = \arg\min_{\mathcal{M}_C \subset \mathcal{M}} \max_{m \in \mathcal{M}} \min_{n \in \mathcal{M}_C} \|m - n\|_2 . \tag{5}$$

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**

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

end

/* Apply greedy coreset selection. */

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

$\mathcal{M}_C \leftarrow \{\}$
for $i \in [0, ..., l-1]$ **do**
  $m_i \leftarrow \underset{m \in \mathcal{M} - \mathcal{M}_C}{\arg\max} \min_{n \in \mathcal{M}_C} \|\psi(m) - \psi(n)\|_2$
  $\mathcal{M}_C \leftarrow \mathcal{M}_C \cup \{m_i\}$
end
$\mathcal{M} \leftarrow \mathcal{M}_C$

Anomaly Detection with PatchCore：

Python

```python
features = self(x)
embeddings = []
for feature in features:
    m = torch.nn.AvgPool2d(3, 1, 1)
    embeddings.append(m(feature))
embedding_ = embedding_concat(embeddings[0], embeddings[1])
embedding_test = np.array(reshape_embedding(np.array(embedding_)))
score_patches, _ = self.index.search(embedding_test , k=args.n_neighbors)
anomaly_map = score_patches[:,0].reshape((28,28))
N_b = score_patches[np.argmax(score_patches[:,0])]
# 论文解释:
'''
To obtain s, we use a scaling w on s  to account for the behaviour of neighbouring patches:
    If the memory bank features closest to the anomaly candidate mtest, , m ,
    is itself relatively far from neighbouring samples and thereby an already rare nominal occurence,
    we increase the anomaly score
    如果内存库特征最接近异常候选 m^test,* , m^*, 本身距离近邻样本相对较远，因此是少见的正常发生，使用权重增加异常分数。
    相当于计算了一个softmax
'''
w = (1 - (np.max(np.exp(N_b))/np.sum(np.exp(N_b))))
score = w*max(score_patches[:,0]) # Image-level score
gt_np = gt.cpu().numpy()[0,0].astype(int)
# 将结果放大：匹配原始输入分辨率
anomaly_map_resized = cv2.resize(anomaly_map, (args.input_size, args.input_size))
# 高斯平滑
anomaly_map_resized_blur = gaussian_filter(anomaly_map_resized, sigma=4)
```

提取Patch特征：

$$\mathcal{P}(x^{\text{test}}) = \mathcal{P}_{s,p}(\phi_j(x^{\text{test}}))$$

计算异常值分数：集合P(Xtest) 到M的距离。公式arg min ||m^test - m|| 表示点m^test到集合M的距离，然后找到最远的点m^test,*。

$$m^{\text{test},*}, m^* = \underset{m^{\text{test}} \in \mathcal{P}(x^{\text{test}})}{\arg\max} \; \underset{m \in \mathcal{M}}{\arg\min} \left\| m^{\text{test}} - m \right\|_2 \qquad (6)$$
$$s^* = \left\| m^{\text{test},*} - m^* \right\|_2 .$$

$$s = \left(1 - \frac{\exp \left\| m^{\text{test},*} - m^* \right\|_2}{\sum_{m \in \mathcal{N}_b(m^*)} \exp \left\| m^{\text{test},*} - m \right\|_2} \right) \cdot s^*, \qquad (7)$$

复现：

环境问题：

```python
# 原有环境:存在问题：roc_auc_score 不高，且在coreset_sampling_ratio=0.1，文件较大（100M）时候报错：Faiss assertion 'err == CUBLAS_STATUS_SUCCESS' failed in void faiss::gpu::runMatrixMult(faiss::gpu::Tensor<float, 2, true>&, bool, faiss::gpu::Tensor<T, 2, true>&, bool, faiss::gpu::Tensor<IndexType, 2, true>&, bool, float, float, cublasHandle_t, cudaStream_t) [with AT = float; BT = float; cublasHandle_t = cublasContext*; cudaStream_t = CUstream_st*] at /project/faiss/faiss/gpu/utils/MatrixMult-inl.cuh:265; details: cublas failed (13): (512, 1536) x (16385, 1536)' = (512, 16385) gemm params m 16385 n 512 k 1536 trA T trB N lda 1536 ldb 1536 ldc 16385
pytorch-lightning==1.2.0
torch==1.9.0+cu111
faiss-gpu==1.7.1.post3

# 尝试解决：重新安装faiss-gpu；结果：roc_auc_score在0.001，0.01的指标变得正常，在0.1（coreset_sampling_ratio越大）时候，稀疏采样（selector.select_batch）越慢 ，
conda install -c pytorch faiss-gpu cudatoolkit=11.0
```

常用的自定义数据格式：

```haskell
data:
├── test
│   ├── good
│   └── ng
└── train
    └── good
```

Resource:

Paper: https://arxiv.org/abs/2106.08265

Code: https://github.com/hcw-00/PatchCore_anomaly_detection

论文解读：https://blog.csdn.net/fuyouzhiyi/article/details/124725323

https://www.zywvvd.com/notes/study/deep-learning/anomaly-detection/patchcore/patchcore/

4、FastFlow: Unsupervised Anomaly Detection and Localization via 2D Normalizing Flows 基于2D归一化流的无监督异常检测与定位

FastFlow 是一个基于二维归一化流的概率分布估计器。它可以用做任何的深度特征提取器的插件模块，比如ResNet 和 vision transformer, 用于无监督异常检测和定位。在训练阶段，FastFlow 学习将输入视觉特征转换位易于处理的分布；在推理阶段，它评估失败异常的可能性。【商汤等】

阅读笔记：

Introduction:

在无监督异常检测中，一个很有promising method 是的使用 deep neural networks 获取正常图像的特征，并用一些统计方法对分布进行建模，然后检测具有不同分布的异常样本。按照这种方法，有两个主要组成部分：特征提取模块模块和分布估计模块。 分布估计模块：1、使用非参数的方法来建模正常图像的特征分布。例如，通过计算特征的均值和方差来估计多维高斯分布，或者使用聚类算

法通过正态聚类来估计这些正态特征。 2、（本文）使用归一化流来估计分布, 并将原始的归一化流扩展到二维空间。通过使正常图像特征的对数似然最大化的可训练过程，将正常图像特征嵌入到标准正态分布中，并使用概率来识别和定位异常。 特征提取模块：vision transformers and CNN.

Related Work:

异常检测方法：基于重构的方法。利用生成模型，如自动编码器 or 生成对抗网络来编码和重构正常数据 (这些方法认为异常是不能被重建的，因为它们不存在于训练样本)。**基于表示的方法**。使用深度卷积神经网络提取正常图像或者正常image patches 的判别特征，并建立这些正常特征的分布。这些方法通过计算测试图像的特征与正常特征分布之间的距离来获得异常得分。 本文：基于表示的方法，从vision transformer 或者 resnet 中提取视觉特征，并通过FastFlow模型建立分布。

用于异常检测的特征提取器：CNN or ViT

Normalizing Flow(标准化流): https://blog.csdn.net/weixin_46782905/article/details/119958725

Methodology:

主流方法之一是基于表示的方法，该方法从正常图像或者正常图像块中提取鉴别特征向量来构造分布，并通过测试图像embedding与分布之间的距离来计算异常得分。该分布通常由正常图像的n球中心、正常图像的高斯分布或从KNN获得的 the normal embedding cluster stored in the memory bank （存储的正常特征簇存储库）来表征。

Feature Extractor: 1、ViT。使用某一层的特征，因为ViT具有更新的能力捕捉局部块和全局特征之间的关系。 2、ResNet。使用前3个块中最后一层的特征，并将这些特征放入3个对应的FastFlow模型中。

2D Flow Model：双射函数；对数似然；将原始的normalizing flow转换成2D方式，采用二维卷积层保留流模型中的空间信息，并相应的调整损失函数。采用3x3卷积和1x1卷积交替出现的全卷积网络，再flow model中更能保留空间信息。

Experments:

suppresses all other methods in anomaly detection task.

Conclusion:

SOTA; robustness;

模型结构：

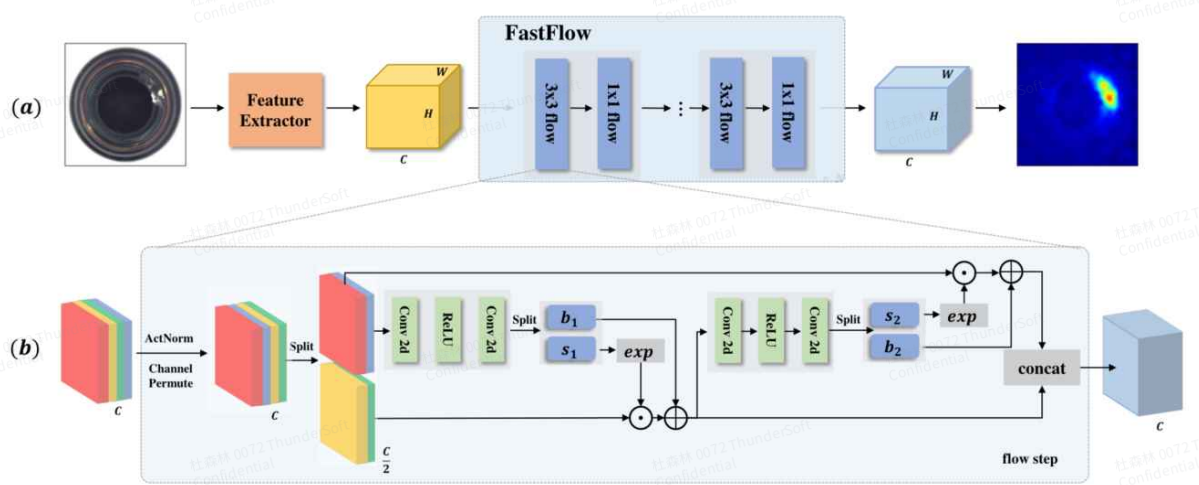


Figure 2: (a) the whole pipeline for unsupervised anomaly detection and localization in our method, which consists of a feature extractor and our FastFlow model. We can use an arbitrary network as the feature extractor such as CNN or vision transformer. FastFlow is alternatly stacked by the "3 × 3" and "1 × 1" flow. (b) one flow step for our FastFlow, the "Conv 2d" can be 3 × 3 or 1 × 1 convolution layer for 3 × 3 or 1 × 1 flow, respectively.

1、特征提取模块

backbone 充当特征提取器，不需要训练，不需要更新参数 requires_grad==false。

2、norms 模块

对于 resnet，norms 是可训练的 LayerNorm。

3、flows模块

可逆神经网络。(参考FrEIA)

复现：

环境：

```
Apache
# 先前环境： sudo pip install torch==1.9.0+cu111 torchvision==0.10.0+cu111 torchaudio==0.9.0 -f https://download.pytorch.org/whl/torch_stable.html
pip install torch==1.9.1+cu111 torchvision==0.10.1+cu111 -f https://download.pytorch.org/whl/torch_stable.html
```

Resource:

论文解读：https://blog.csdn.net/qq_45700830/article/details/122690958

https://blog.csdn.net/weixin_46782905/article/details/119958725

https://blog.csdn.net/qq_35030874/article/details/122426454

5、目标检测-Yolo系列

5.1 yolov1/2/3

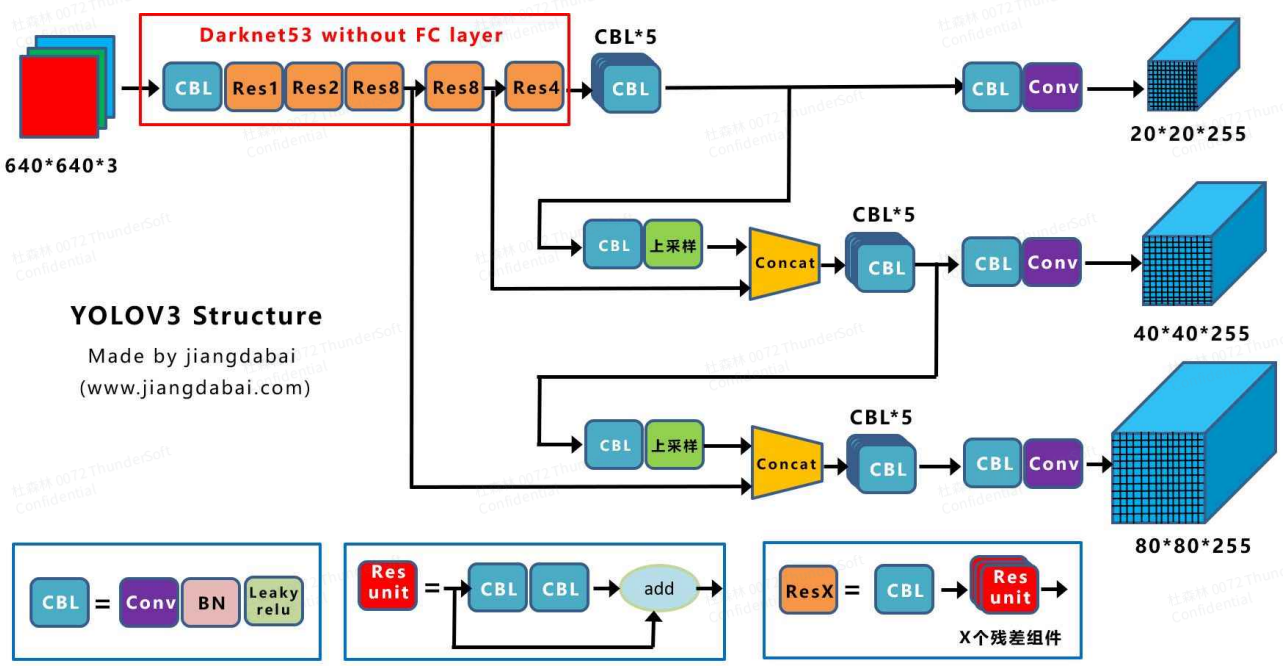
作者：Joseph Redmon

代码（darket版本）：https://github.com/pjreddie/darknet

pytorch版本：https://github.com/ultralytics/yolov3

基础知识：Deep Learning

yolov3网络结构图：



具体模块理解：

- CBL: Yolov3网络结构中的最小组件，由**Conv+Bn+Leaky_relu**激活函数三者组成

- Res unit: 借鉴Resnet网络中的残差结构，让网络可以构建的更深。
- ResX: 由一个CBL和X个残差组件构成，是Yolov3中的大组件。每个Res模块前面的CBL都起到下采样的作用，因此经过5次Res模块后，得到的特征图是608->304->152->76->38->19大小。
- Concat: 张量拼接，会扩充两个张量的维度，例如26*26*256和26*26*512两个张量拼接，结果是26*26*768。Concat和cfg文件中的route功能一样;
- add: 张量相加，张量直接相加，不会扩充维度，例如104*104*128和104*104*128相加，结果还是104*104*128。add和cfg文件中的shortcut功能一样;
- 卷积层数量计算：主干网络Backbone中一共包含
  1+（1+2*1）+（1+2*2）+（1+2*8）+（1+2）+（1+2*4）=52，再加上一个FC全连接层，即可以组成一个Darknet53分类网络。不过在目标检测Yolov3中，去掉FC层，不过为了方便称呼，仍然把Yolov3的主干网络叫做Darknet53结构。

理解：（1) DarkNet-53: 残差思想, 缓解梯度消失的问题，使得模型更容易收敛； 深、浅层特征融合+多层特征图（multi-scales）；无池化层（使用卷积步长为2）；

（2）多尺度预测：使用聚类算法得到9种不同大小宽高的先验框，3个特征图，每个特征图预测3个先验框；COCO数据集：一个先验框：80个类别预测值 + 4个位置预测值 + 1个置信度预测值；3个预测框3x(80+5) =255=每一个特征图的预测通道数；

（3）使用Logistic函数（sigmoid）代替Softmax函数处理类别的预测得分，Softmax只能预测一个类别，Logistic分类器相互独立，可以实现多类别的预测, Softmax可以被多个独立的Logistic分类器取代，并且准 确率不会下降；

5.2 yolov4

作者：Alexey Bochkovskiy

代码（作者官方版本）：https://github.com/AlexeyAB/darknet

pytorch版本（随便找的一个）：https://github.com/Tianxiaomo/pytorch-YOLOv4

算法理解：

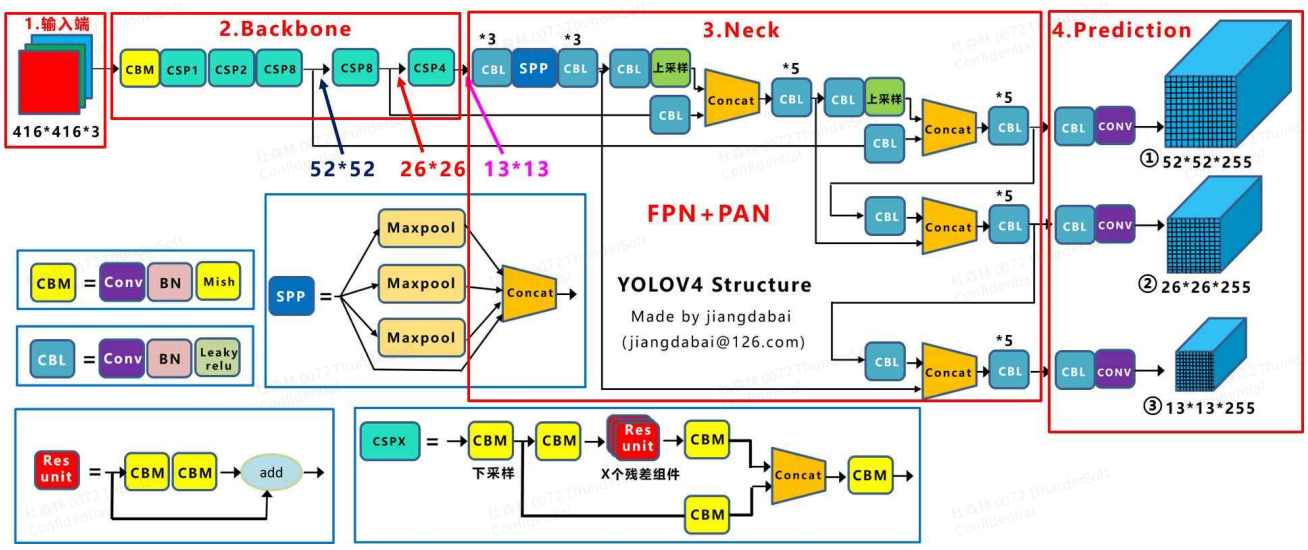
输入端：Mosaic数据增强（丰富训练数据集；减少GPU数量）；cmBN; SAT自对抗训练；

Backbone: CSPDarkNet(参考cspnet, 增强CNN学习能力；降低计算瓶颈；降低内存成本)；Mish激活函数；Dropblock（Dropout 随机失活，大量用在全连接层中；dropblock, 用在卷积神经网络中，受到cutout数据增强思想启发，多个局部区域，整体删减丢弃）；

Neck: SPP（四种最大池化方式，1x1; 5x5; 9x9; 13x13）; FPN(上采样，分辨率变小，感受野增大，自顶向下，传达强语义特征) + PAN（下采样，自底向上，传达强定义特征），理解可参考 https://blog.csdn.net/junqing_wu/article/details/105598849

输出层：CIoU_loss（用于坐标的回归损失函数）; DIoU_nms(缓解遮挡问题);

yolov4网络结构图：



模块理解：

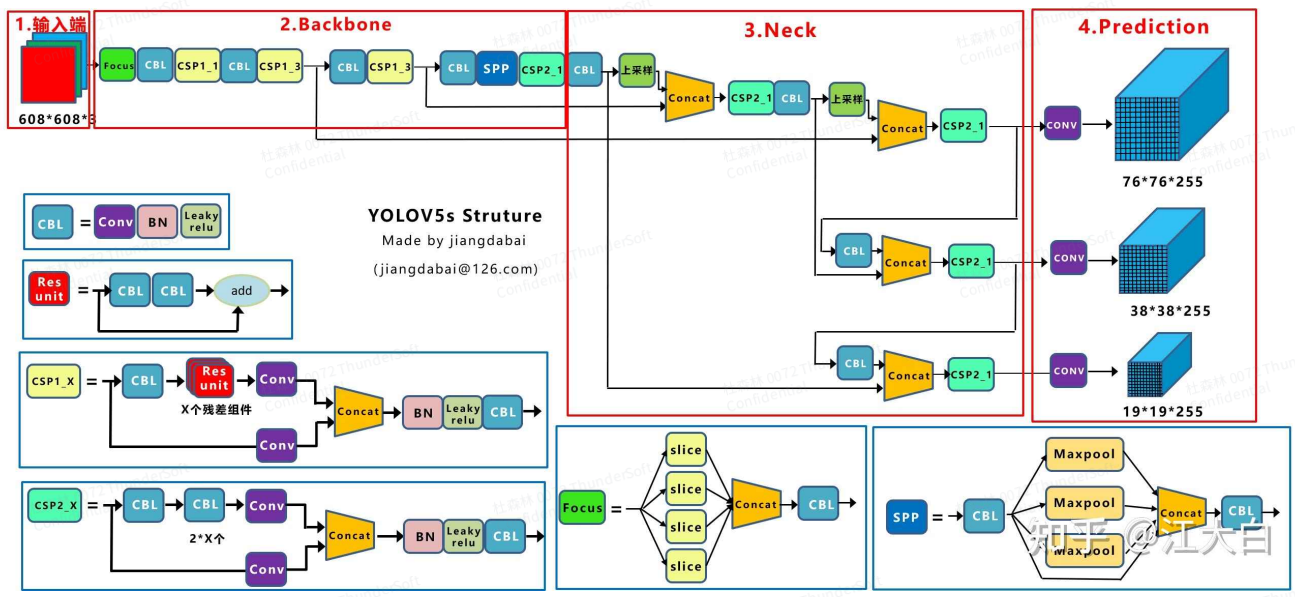
- CBM: Yolov4网络结构中的最小组件，由Conv+Bn+Mish激活函数三者组成;
- CBL: 由Conv+Bn+Leaky_relu激活函数三者组成; 同yolov3
- Res unit: 借鉴Resnet网络中的残差结构，让网络可以构建的更深; 同yolov3
- CSPX: 借鉴CSPNet网络结构，由卷积层和X个Res unint模块Concate组成;
- SPP: 采用1×1，5×5，9×9，13×13的最大池化的方式，进行多尺度融合。

5.3 yolov5

作者：Ultralytics团队 无论文。相关讨论：https://cloud.tencent.com/developer/article/1661447

代码：https://github.com/ultralytics/yolov5

网络结构图：



理解：

- 输入端：Mosaic数据增强；自适应锚框计算；自适应图片缩放；
- Backbone: Focus结构；CSP结构；
- Neck: FPN + PAN结构，同yolov4;
- 输出端：Bounding box的损失函数是CIoU_loss, 同yolov4；非极大值抑制：DIoU_nms，同yolov4;

5.4 yolox

作者: 旷视BaseDetection团队

代码: https://github.com/Megvii-BaseDetection/YOLOX

blog: https://zhuanlan.zhihu.com/p/397993315

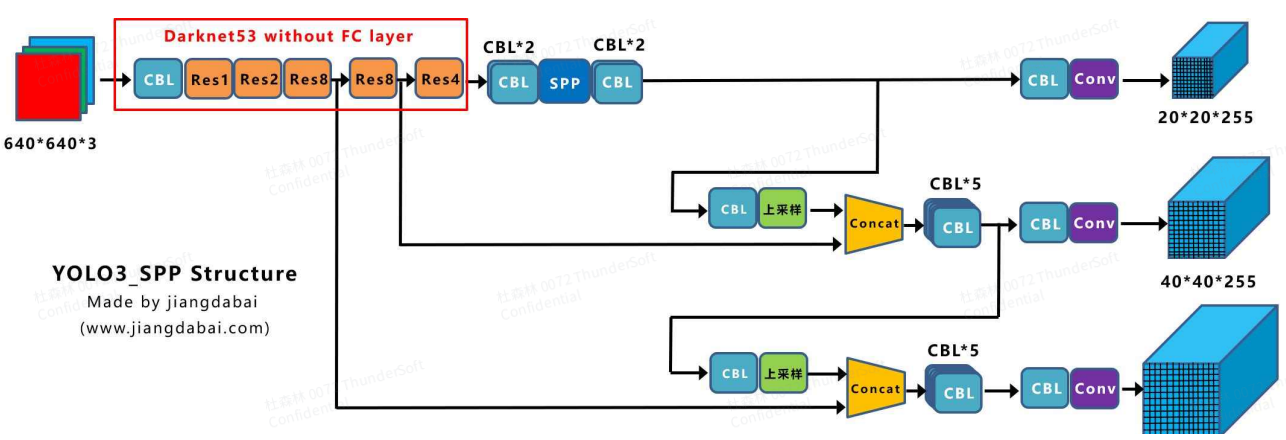
标准网络结构：Yolox-s、Yolox-m、Yolox-l、Yolox-x、Yolox-Darknet53；

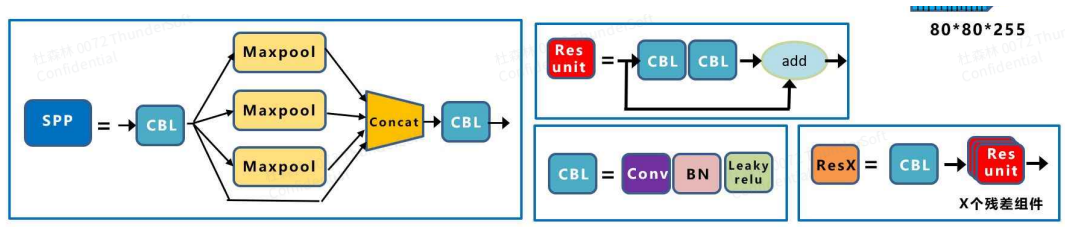
轻量级网络结构：Yolox-Nano、Yolox-Tiny；

改进思路：

（1）基准模型：Yolov3_spp

选择Yolov3_spp结构，并添加一些常用的改进方式，作为Yolov3 baseline基准模型；

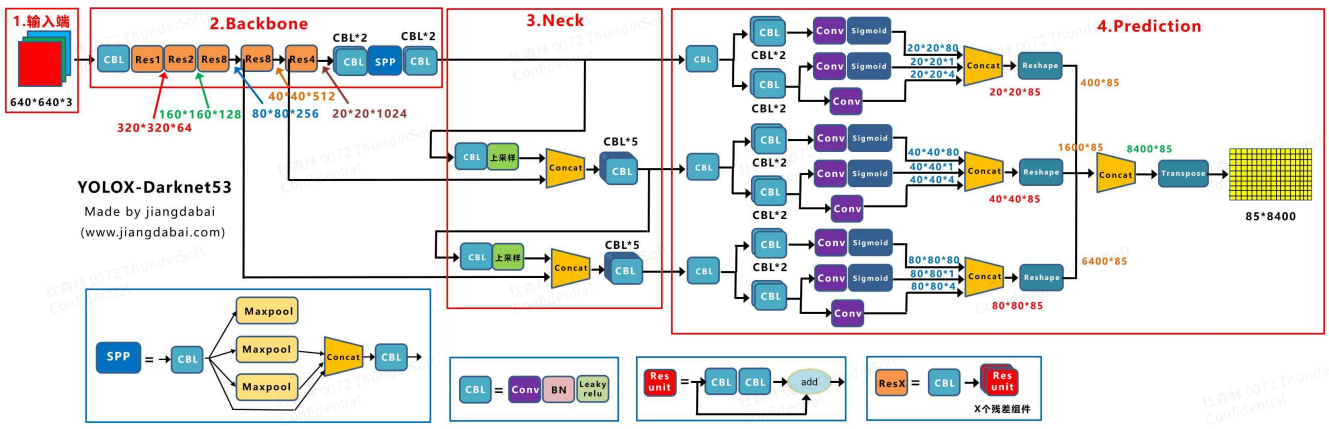




训练过程改进：a、添加了EMA权值更新、Cosine学习率机制等训练技巧；b、使用IOU损失函数训练reg分支，BCE损失函数训练cls与obj分支；c、添加了RandomHorizontalFlip、ColorJitter以及多尺度数据增广，移除了RandomResizedCrop；

(2)Yolox-Darknet53:

对Yolov3 baseline基准模型，添加各种trick，比如**Decoupled Head、SimOTA**等，得到Yolox-Darknet53版本；



- 输入端：Mosaic数据增强（**随机缩放、随机裁剪、随机排布**的方式进行拼接）；Mixup数据增强（两张图片融合）；训练的**最后15个epoch**，这两个数据增强会被关闭掉；由于采取了更强的数据增强方式，作者在研究中发现，**ImageNet预训练将毫无意义**，因此，所有的模型，均是**从头开始训练的；**
- Backbone: Yolov3 baseline
- Neck: FPN结构融合（**FPN自顶向下**，将高层的特征信息，通过**上采样的方式进行传递融合**，得到进行预测的特征图）
- Prediction层：
  a. Decoupled Head: 三个Decoupled Head, 称“解耦头”；细节：三个分支，类别分支-N个类别的二分类判断，Sigmoid激活函数；前景背景判断分支；目标框坐标信息分支；三个分支concat融合，再进行总体的concat, 得到8400*85的预测信息，8400指预测框的数量。**将检测头解耦，会增加运算的复杂度——可使用 1个1x1 的卷积先进行降维。**
  b. Anchor-free: Yolov3、Yolov4、Yolov5中，通常都是采用**Anchor Based的方式**，来提取目标框，进而和标注的groundtruth进行比对，判断两者的差距。Anchor Free方式：参数量少了2/3；Anchor框信息，巧妙的将Backbone下采样的大小信息引入进来。400个框，对应锚框大小2^5=32, 1600个预测框对应锚框大小16*16，6400个预测框对应锚框大小8*8； 锚框相当于桥梁，将8400给锚框和图片上目标框进行关联，挑选出正样本锚框，**正样本锚框**所对应的位置，就可以将**正样本预测框**。关联方式--标签分配。
  c. 标签分配：如何挑选正样本锚框？两个关键点：初步筛选，SimOTA;

  初步筛选：根据中心点/根据目标框（如从8400提出来1000个）；

SimOTA(精细化筛选): **初筛正样本信息提取**：拿到第一步初步筛选提出的框的位置信息+前景背景信息+类别信息； **Loss函数计算**：对初步筛选的框与gt计算loss函数；**cost成本计算**：根据上一步计算的位置损失和类别损失计算cost成本函数；**SimOTA**: 例如假设三个目标框，为每个目标框挑选10个iou最大的候选框，根据iou和计算每个目标框分配几个候选框（如分别3，4，3），再去1000个候选框中选择对应数量的cost最小的框；如果候选框和多个目标检测框关联，选择较小cost的；

- Loss计算：经过第三部分的标签匹配，目标框和正样本预测框对应起来了。然后计算两者的Loss, 位置损失函数：iou_loss/giou_loss; 目标损失函数：BCE_loss (二分类交叉熵损失函数); 分类损失：BCE_loss. 注意：（1） 前面精细化筛选中，使用了reg_loss和cls_loss，筛选出和目标框所对应的预测框。因此这里的iou_loss和cls_loss，只针对目标框和筛选出的正样本预测框进行计算。而obj_loss，则还是针对8400个预测框； （2）在Decoupled Head中，**cls_output**和**obj_output**使用了sigmoid函数进行归一化，但是在训练时，并没有使用sigmoid函数，原因是训练时用的**nn.BCEWithLogitsLoss函数**，已经包含了sigmoid操作。

(3) Yolox-s、Yolox-m、Yolox-l、Yolox-x系列

对Yolov5的四个版本，采用这些有效的trick，逐一进行改进，得到Yolox-s、Yolox-m、Yolox-l、Yolox-x四个版本；

(4)轻量级网络：Yolox-Nano, Yolox-Tiny;

5.5 yolov6

作者：美团

代码：GitHub - meituan/YOLOv6: YOLOv6: a single-stage object detection framework dedicated to industrial a

blog：https://blog.csdn.net/Pariya_Official/article/details/125794741

blog: https://tech.meituan.com/

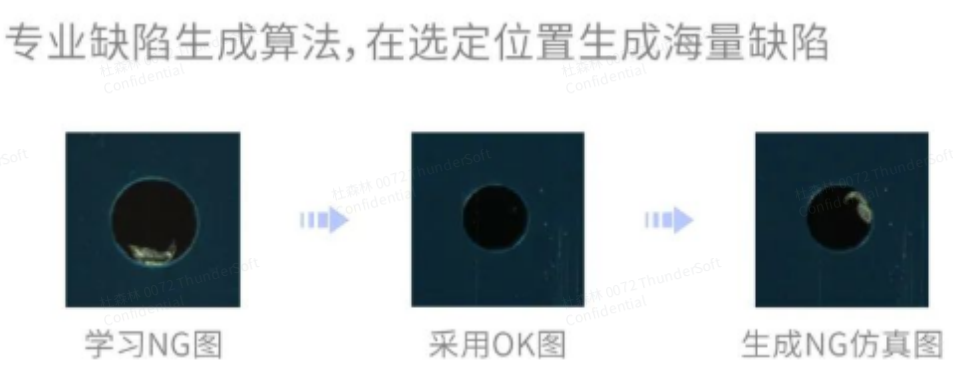
5.6 yolov7

作者：有Alexey Bochkovskiy

代码：GitHub - WongKinYiu/yolov7: Implementation of paper - YOLOv7: Trainable bag-of-freebies sets new sta

6、缺陷生成算法

对标样例：



https://www.degruyter.com/document/doi/10.1515/epoly-2022-0071/html

7、目标检测-Faster RCNN系列

二、useful info

1、海思平台

yuv图片输入目标检测：海思AI芯片(Hi3519A/3559A)方案学习(二十四)如何直接对yuv图片进行目标检测 - 极术社区 - 连接开发者与智能计算生态

pytorch2caffe: https://github.com/inisis/brocolli

2、book: https://github.com/Tvirus/ebook https://github.com/csc-training/hpc-python

三、ProEx Summary

1、武汉施耐德模块

1-1、异常检测模型

1、源码和论文

😶 源码：https://github.com/gathierry/FastFlow

论文：https://paperswithcode.com/paper/fastflow-unsupervised-anomaly-detection-and

2、复现

1-2、YoloX onnx 内存泄漏

2、保隆气门嘴

2-1、一期分类模型

2-2、二期目标检测模型

3、保隆胶片

3-1、棋盘格标定、畸变校正算法

3-2、基于轮廓的模板匹配算法

3-3、卡尺测量方法

4、保隆储气罐

4-1、基于轮廓的模板匹配算法

4-2、卡尺测量圆

5、高通8450平台道通算法支撑

5-1、高通CPU-GPU/DSP数据迁移

5-2、SNPE批量推理Demo

5-3、Make 编译C/C++

6、AIForest 模型量化和压缩

6-1、Pytorch 模型量化

6-2、AIMET 模型量化和压缩（yolov3/yolox）

7、人体关键点检测APP demo

8、施耐德上海SSIC

四、数据集整理

1、异常检测

五、Math

1、线性代数

2、概率论