Image Manipulation Detection by Multi-View Multi-Scale Supervision

Xinru Chen, Chengbo Dong, Jiaqi Ji, Juan Cao, Xirong Li

摘要

图像篡改检测的关键挑战是如何学习对新数据篡改敏感的泛化特征同时防止真实图像上的假警报，当前的研究强调 sensitivity，忽视了 specificity，该论文通过多视角特征学习 (multi-view feature learning) 和多尺度监督 (multi-scale supervision) 两个支路来学习对操作敏感的语义无关特征，同时具体防止错误警报。多视角特征学习支路利用篡改区域周围的噪声分布和边界伪影，学习语义不可知的特征，从而获得更普遍的特征；多尺度监督允许模型从真实的图像中学习。基于此论文提出一个新的网络结构叫 MVSS-Net，在五个基准数据集上像素级别和图像级别检测表现不错。

关键词：图像篡改检测；多视图特征学习；多尺度监督；模型的敏感性和特异性

1 引言

图像处理常见的方式有三种，复制粘贴 (copy-move) 将元素从给定图像中的一个区域复制并移动到另一个区域，拼接组合 (splicing) 将元素从一个图像复制并粘贴到另一个图像上，修饰 (inpainting) 移除不需要的元素。论文的目的就是对这些类型的篡改图像实现自动检测，不仅检测出篡改与否，还要在像素级别精确定位出图像篡改位置。

最先进的图像篡改定位技术是基于深度学习方法的，比如 SPAN[1]，MFCN[2]，ManTra-Net[3]，constrained R-CNN[4]等。这些模型大多只考虑两个类（操纵与真实），完成的任务被考虑为图像语义分割的简化案例，但是由于这些模型被设计用于捕获语义信息，使得模型过于依赖数据集，导致训练出来的模型泛化性极差。为了学习语义不可知特征 (semantic-agnostic features)，有必要抑制图像内容，使得模型可以学习到语义无关的特征。比如在做篡改和非篡改的二分类任务时，分类模型除了会学到篡改特征之外，还会学习到其他与篡改特征无关的特征因素，这些特征对于篡改检测是不需要的，所以要抑制这些图像内容的干扰。根据抑制发生的阶段将现有的方法分为两组，噪声视图方法 (noise-view methods) 和边缘监督方法 (edge-supervised methods)，噪声视图方法旨在利用由 splicing 和 inpainting 操作而引入的新元素在噪声分布方面与真实部分不同的差异，通过预定义高通滤波器或其可训练的对应物生成的输入图像的噪声图被单独或与输入图像一起送入深度网络，但对没有新元素引入的 copy-move 图像是无效的；边缘监督方法集中于发现边界伪影作为篡改区域周围的篡改轨迹，通过添加辅助分支来重建篡改区域的边缘实现。但是现有技术统一地将来自主干的不同层的特征连接为辅助分支的输入。因此，负责操作检测的更深层特性仍然是语义感知的，泛化能力还是不够好。

模型篡改性能的评估大多是在公开数据集（如 CASIA2.0）上训练，然后再其他数据集（如 CASIA1.0，Columbia）上测试，但是大多数是在篡改图像上评估，忽视了真实图像的学习，这些模型在现实场景上使用对真实图像的虚警率高得离谱。所以通过对真实图像的学习来提高模型的特异性 (specificity) 是非常重要的。

受启发于模型 Border Network[5]逐步聚合特征来预测物体边界，以及 LesionNet[6]模型集成了图像分类损失，用于视网膜病变分割，论文提出一种新的网络结构 MVSS-Net，结合多视图和多尺度监督来学习伪影特征。

2 相关工作

2.1 传统图像篡改技术

早期针对整张图像的篡改定位方法大多通过滑动窗口的方式来实现，为了提高算法的准确率和篡改定位的性能，大多篡改定位方法结合小尺寸的滑动窗口以及不同的特征提取算法。例如: Bianchi 和 Piva[7]提出在 8×8 的图像块中分析 JPEG 重压缩痕迹来定位篡改区域; Ferrara[7]等人尝试在最小为 2 ×2 的图像块中提取去马赛克痕迹来判断是否发生了篡改; Chierchia[8]等人利用贝叶斯框架来估计传感器模式噪声，并结合马尔可夫随机场 (M R F) 对像素间关系建模，得到关于篡改区域的预测; Fan[9] 等人提出利用高斯混合模型 (GMM) 对图像小块的统计特性进行建模; Korus 和 Huang[Runge]通过融合多种尺寸滑动窗口的输出结果来提高篡改定位的性能。虽然这些方法在某些篡改定位任务上取得了不错的效果，但仍有一些因素影响了它们进一步的发展。首先，它们所采用的基于手工设计的特征受到一定先验条件的限制，往往并不能较好地应对现实情况中纷繁复杂的图像来源和篡改手段。其次，基于滑动窗口的定位策略在使用时不易确定较优的窗口尺寸参数，而且计算效率不高，尤其是当图像尺寸较大时处理时间显著增加。

2.2 基于深度学习的图像篡改技术

近年来，深度学习在计算机视觉、语音识别、自然语言处理等领域引领了技术进步的潮流。在此过程中，深度学习技术也被逐渐引入到图像篡改定位领域并取得不错的检检测定位效果。例如使用自编码器 (Autoencoder，AE) 作为基础框架，而卷积神经网络 (Convolutional Neural Network，CNN) 则被普遍地使用。在图像语义分割和目标检测中常用的全卷积网络 (Fully Convolutional Network，FCN) 和 R-CNN (R egion proposals with CNN features) 系列网络也被大量应用在图像篡改定位领域，同时还有孪生网络 (Siamese Network)，长短期记忆网络 (Long short-term memory，LSTM) 和生成对抗网络 (Generative Adversarial Network，GAN) 也有相关的应用。

Methods	Views			Backbone	Scales of Supervision		
	RGB	*Noise*	*Fusion*		*pixel*	*edge*	*image*
Bappy *et al.* 2017, J-LSTM [1]	+	-	-	Patch-LSTM	+	-	-
Salloum *et al.* 2017, MFCN [21]	+	-	-	FCN	+	+	-
Zhou *et al.* 2020, GSR-Net [29]	+	-	-	Deeplabv2	+	+	-
Li & Huang 2019, HP-FCN [16]	-	High-pass filters	-	FCN	+	-	-
Yang *et al.* 2020, CR-CNN [27]	-	BayarConv2D	-	Mask R-CNN	+	-	-
Zhou *et al.* 2018, RGB-N [30]	+	SRM filter	late fusion (bilinear pooling)	Faster R-CNN	*	-	-
Wu *et al.* 2019, ManTra-Net [26]	+	SRM filter, BayarConv2D	early fusion (feature concatenation)	Wider VGG	+	-	-
Hu *et al.* 2020, SPAN [14]	+	SRM filter BayarConv2D	early fusion (feature concatenation)	Wider VGG	+	-	-
MVSS-Net(This paper)	+	BayarConv2D	late fusion (dual attention)	FCN	+	+	+

图 1: 图像篡改定位的先进工作

上图是作者调研的最近的一些工作，论文主要关注 copymove/ splicing/ inpainting 三种类型，对于高斯模糊和 JPEG 压缩这种引用 Constrained CNN。可以从表中看出使用 BayerConv 这种受约束的卷积层有助于提取噪声信息，但单独使用他们会带来丢失原始 RGB 输入中其他有用信息的风险。双流

FasterrCNN 用的是 SRM 滤波器，MVSS-Net 使用了噪声图，并且在后期融合了 RGB 和噪声图，并且融合使用的也不是不可训练的双线性池化，而是 Dual attention。

3 本文方法

3.1 MVSSNet 模型结构

下图是用于图像操作检测的多视图多尺度监督网络的概念图。使用边缘监督分支（ESB）和噪声敏感分支（NSB）来学习语义不可知的特征来进行操作检测，并使用多尺度监督来在模型的灵敏度和特异性之间取得适当的平衡。σ 层为 sigmoid，Gclf 模块负责将像素级分割图 S (x) 转换为图像级预测 C (x)。

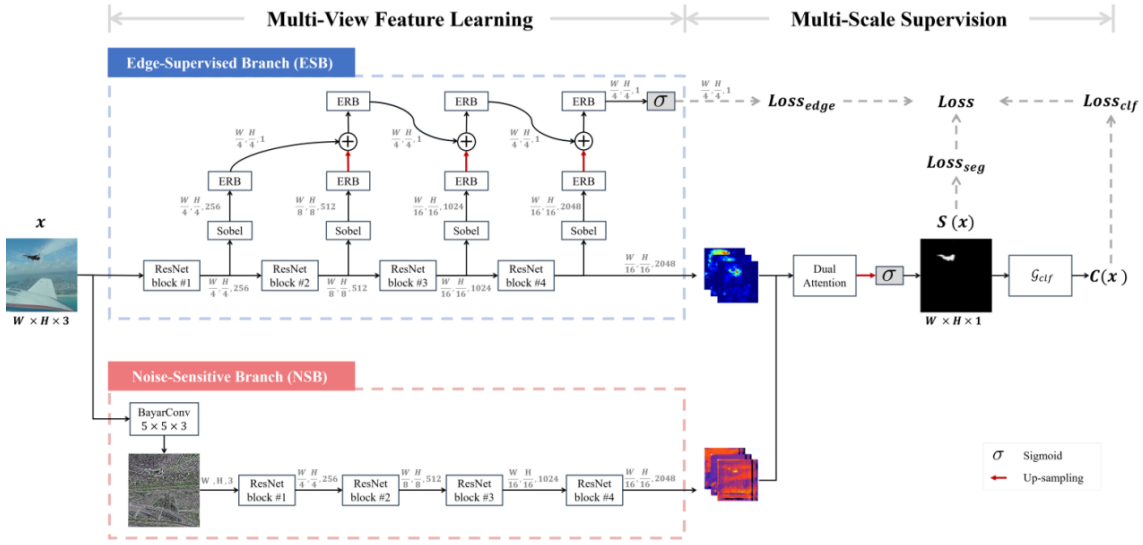


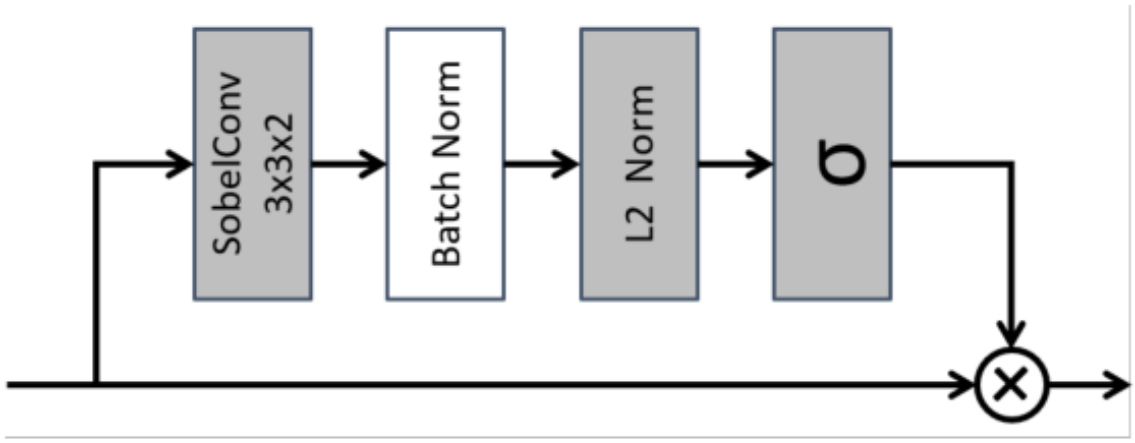
图 2: MVSSNet 模型结构示意图

3.2 多视图特征学习

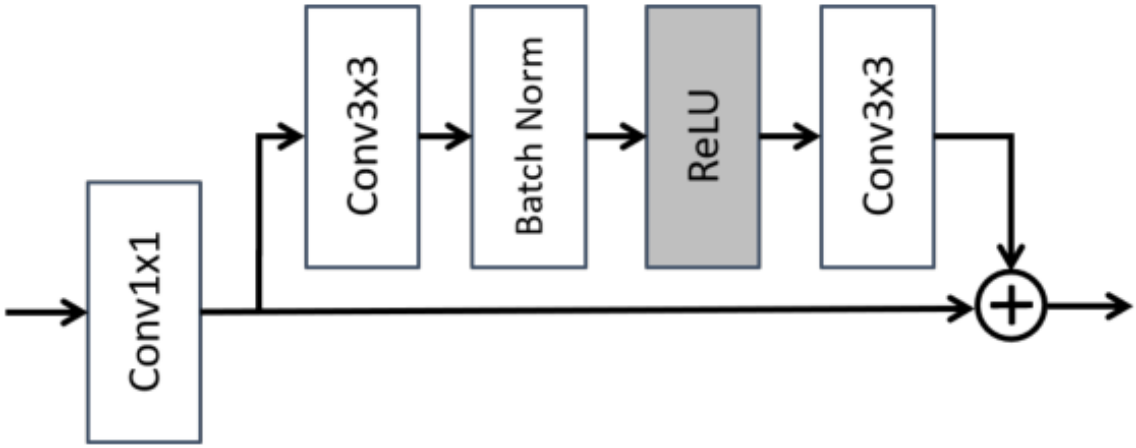
ResNet50 为模型主干，edge-supervised 分支专门设计利用篡改区域周围的细微边界伪影，noise-sensitive 分支旨在捕获篡改区域和真实区域的不一致。两个支路都和语义无关。

3.2.1 边缘监督分支

理想情况下，通过边缘监督期望网络的响应区域更加集中在被篡改的区域。设计这样一个边缘监督网络并非易事，主要挑战是如何为边缘检测头构建适当的输入。如果直接使用最后一个 ResNet 块的特征会导致深层特征捕获低级边缘特征，从而影响操作分割的主要任务。但直接使用来自初始块的特征同样也会出现问题，因为这些浅层特征中包含的细微边缘图案可以在多次深度卷积后轻松消失。因此，有必要同时使用浅层和深层特征，同时考虑到特征是混合的特性，改进之前有工作使用的简单特征连接的方式，论文工作以从浅到深的方式构建边缘头的输入，将来自不同 ResNet 块的特征以渐进方式组合用于操作边缘检测。为了增强与边缘相关的模式，引入了 Sobel 层，如下图。



(a) Sobel Layer



(b) Edge Residual Block (ERB)

图 3: (a)Sobel 层和 (b) 边缘残差块图

第 i 个块的特征首先通过 Sobel 层，然后是边缘残差块 (ERB)，然后将它们与来自下一个块的对应物组合（通过求和）。为了防止累积的影响，组合的特征在下一轮特征组合之前要经过另一个 ERB。这种机制有助于防止边缘头过度监督或完全忽略深层特征的极端情况。通过可视化图中最后一个 ResNet 块的特征图，我们观察到所提出的 ESB 确实在篡改区域附近产生了更集中的响应。

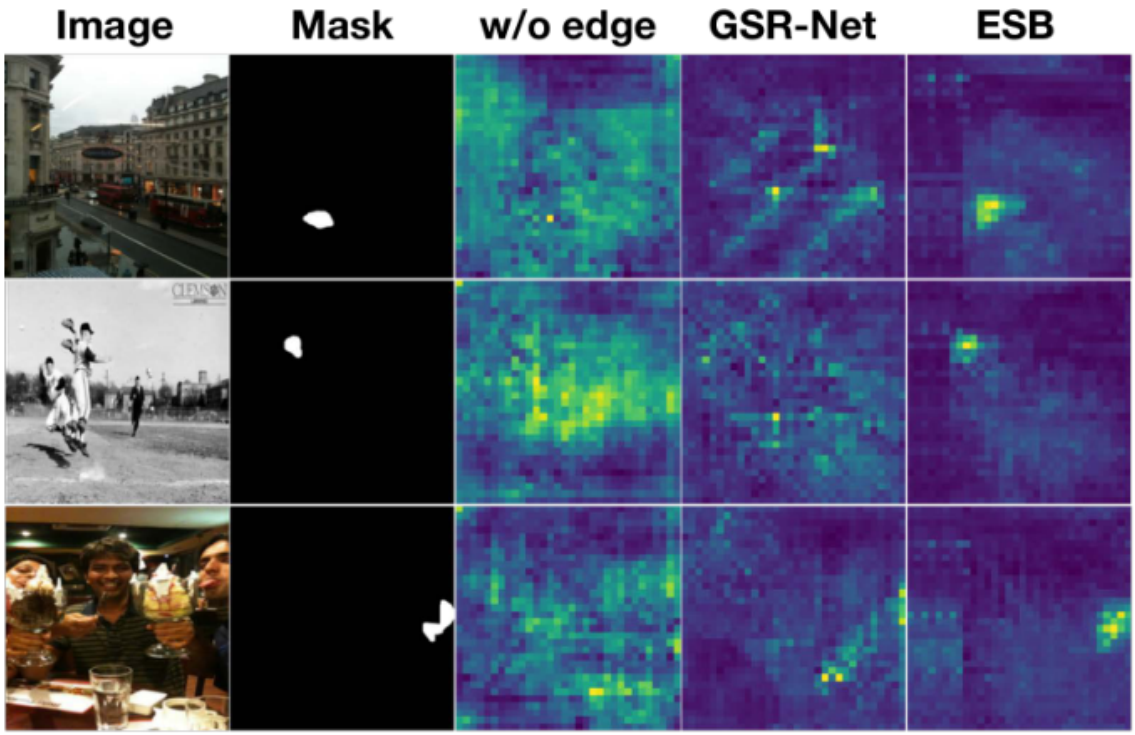


图 4: 最后一个 ResNet 块的平均特征映射的可视化

在图 1 中，ESB 有两个输出，第一个输出经过 sigmoid 函数，是个边缘监督图，第二个是主要的分割图。

$$\left.\begin{matrix}[f_{esb,1},\ldots,f_{esb,k}]\\ \{G_{edge}(x_i)\}\end{matrix}\right\} \leftarrow \text{ERB-ResNet}\,(x) \tag{1}$$

3.2.2 噪声敏感分支

为了充分利用噪声视图，论文构建了一个与 edge-supervised branch 并行的噪声敏感分支，NSB 是一个标准的 FCN，使用 ResNet50，噪声提取选择了 BayarConv，它比 SRM 滤波器更好。

$$\{f_{nsb,1},\ldots,f_{nsb,k}\} \leftarrow \text{ResNet}(\text{BayarConv}(x)) \tag{2}$$

3.2.3 Dual Attention 实现分支融合

通过可训练的 dual attention 模块来融合 ESB 和 NSB 的特征输出图，未使用双线性池化，双流 FasterrCNN 用的是双线性池化，是不用训练的。DA 有两条并行的支路，蓝色是通道，绿色是位置，CA 将通道特征关联起来，以选择性地强调相互依赖的通道特征图。同时，PA 通过所有位置的特征的加权和来选择性地更新每个位置的特征。CA 和 PA 融合之后，通过 1x1 卷积转换为 1 个通道的图，图像大小不变，然后再使用无参数双线性上采样，再是 sigmoid，转为最终的分割图。DA 之前是两个 2048 通道的图的加和，加和之后变成 4096 通道，在经过 DA attention 变成 1 通道。如图 21

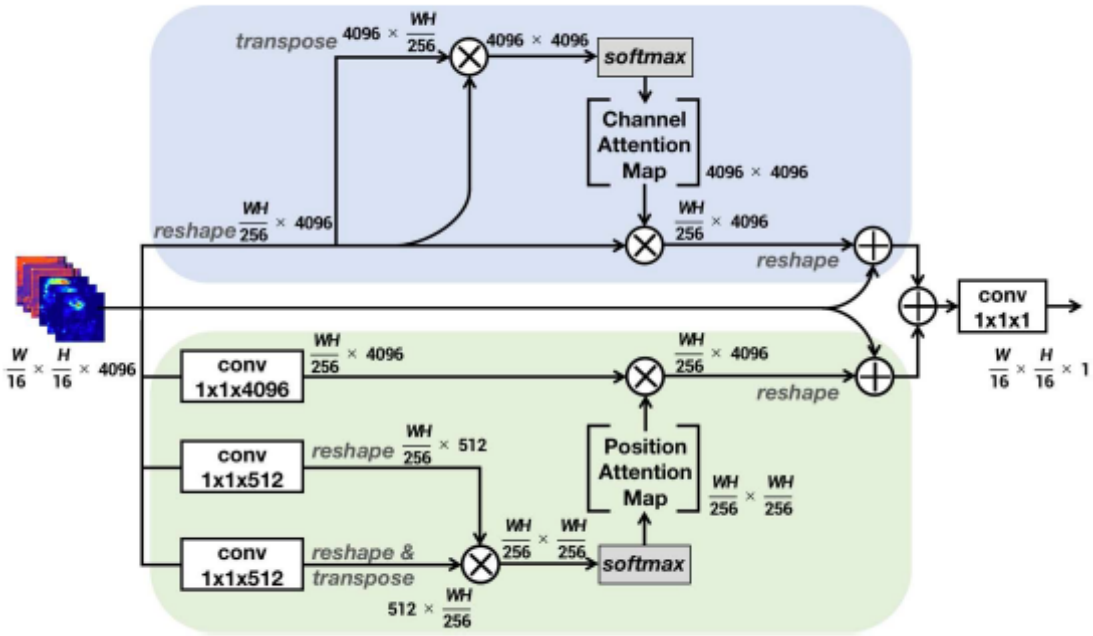


图 5: Dual Attention 分支融合

3.3 多尺度监督

三个尺度的监督，像素级别损失，用于学习语义无关特征的边缘损失和图像级别的损失，像素级别的损失值 (pixel-scale loss) 和边缘监督的损失值 (edge loss) 使用 Dice Loss 计算,Dice Loss 对正负样本严重不平衡的场景有不错的性能表现，被广泛应用在图像分割领域，Dice Loss 通过预测和 GT 的交集除以它们的总体像素进行计算，将一个类别的所有像素作为一个整体进行考量，而且计算交集在总体中的比例，所以不会受大量主流像素的影响，能够提取更好的效果。。其中 edge loss 不在原图尺寸上做计算，在 1/4 图尺寸下计算损失，减少了训练时的计算成本，同时提高了性能。图像级别的损失值 (image-scale loss) 使用 BCE Loss，BCE Loss 是对像素的二分类，图像分割任务中，softmax 交叉熵 loss 是对每一个像素点进行类别预测，然后平均所有的像素点. 其本质上仍是对图片的每个像素进行平等的学习，这就导致如果图像上的多种类别存在不平衡时，模型的训练会由最主流的类别所主导. 网络更偏向于对主流类别的学习，而降低了对非主流类别的特征提取能力，BCE 的话会对正负样本加一下权。组合三个损失公式如下：

$$\text{Loss} = \alpha \cdot \text{loss}_{seg} + \beta \cdot \text{loss}_{clf} + (1 - \alpha - \beta) \cdot \text{loss}_{edg} \tag{3}$$

4 复现细节

4.1 与已有开源代码对比

本次实验使用了论文在 GitHub 上开源的模型代码。模型结构的论文是直接使用开源的代码，个人实现的主要工作包括 MVSS-Net 模型的训练和测试代码和复现其多尺度监督的三个 loss 模块功能以及参考 MVSS-Net++ 模型改进了图像级别的预测中使用新的模块提升模型效果。如图 21是复现模型实现流程，首先是完成数据集的处理，分为篡改图像和 mask 图像；接着是训练模型的实现，包括数据预处理，数据封装，损失函数，学习率等参数的定义，保存验证效果最好的模型进行测试，测试的评估指标包括 Accuracy，IOU，F1score。

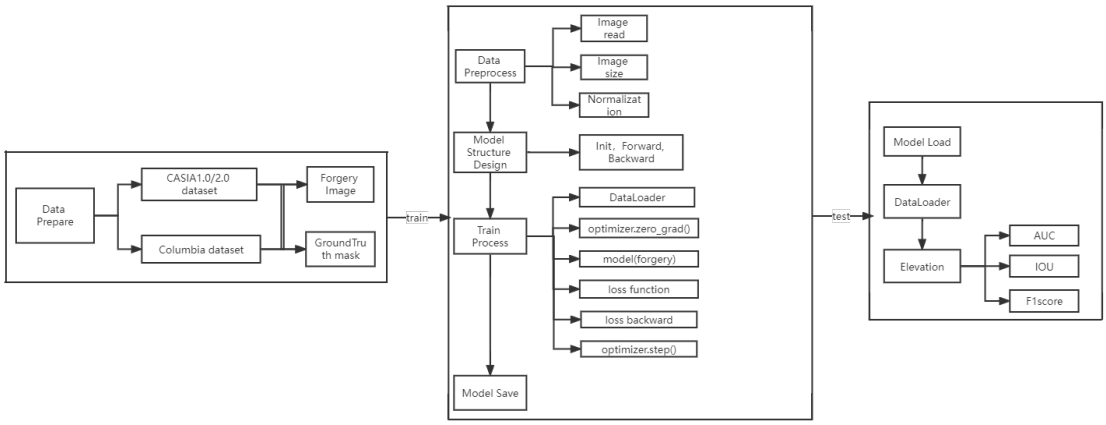


图 6: 复现工作流程

模型训练部分的实现：数据集的封装继承 DataSet，返回篡改图像，mask，edge mask 和表示真假的 label 标签。这里将 MVSSNet 模型在数据集 CASIA2.0 上进行训练，训练的 epoch 为 100 轮，初始学习率设置为 1e-6，受限于 GPU 的大小这里的 batchsize 设置为 4。

代码实现：mvssnetdataset.py

```
class MVSSNetDataset(Dataset):
    def __init__(self, root, txt, val_txt, mode, is_used_edge=False, shuffle=True, seed=None, size=(512, 512)):
        self.size = size
        self.mode = mode
        self.root = root
        self.is_used_edge = is_used_edge
        self.save_list = []
        if self.mode == "val" or self.mode == "test":
            txt = val_txt
        with open(os.path.join(root, txt), "r", encoding='utf-8') as file:
            for line in file:
                line = line.strip('\n')
                line = line.split(' ')
                self.save_list.append(line)
        file.close()
        if shuffle:
            if seed:
                random.seed(seed)
            random.shuffle(self.save_list)
            random.seed(None)
    def __getitem__(self, idx):
        forgery_path = os.path.join(self.root, self.save_list[idx][1])
        mask_path = os.path.join(self.root, self.save_list[idx][2])
        forgery, mask = read(forgery_path, mask_path)
        forgery, mask = process(forgery, mask)
    if self.mode == 'val' or self.mode == 'test':
        self.is_used_edge=False
        return forgery, mask
    if self.is_used_edge:
        label = torch.ones((len(self.save_list),1), dtype=np.float)
        edge_mask = gen_edge_mask(mask)
        edge_mask = cv2.resize(edge_mask, (512 // 4, 512 // 4))
        return forgery, mask, edge_mask, label[idx]
    return forgery, mask

def __len__(self):
    return len(self.save_list)
```

图 7: mvssnetdataset.py

train.py

```python
def train(args):
    model = get_mvss(backbone='resnet50', pretrained_base=True, nclass=1, sobel=True, constrain=True, n_input=3,is_plus=True)
    if len(args.multi_gpu) > 1 and torch.cuda.device_count() > 1:
        model = torch.nn.DataParallel(model)
    model.cuda()

    # ./././ continue to train, if load./././ #
    load_epoch = 0
    if args.load:
        save_file = torch.load(os.path.join(os.path.join(args.weight_save_dir, args.model + args.suffix),
                                            'experiment_{}_'.format(args.flag) + 'epoch_70.pth'), map_location='cpu')
        load_epoch = save_file['epoch']
        model.load_state_dict(save_file['state_dict'])

    # optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=0.9)
    optimizer = optim.Adam(model.parameters(), lr=args.lr)
    if args.load:
        optimizer.load_state_dict(save_file['optimizer'])
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, factor=0.8, patience=10, verbose=True)
    criterion = nn.BCELoss()
    criterion_dice = DiceLoss()


    train_dataset = MVSSNetDataset(root=args.root, txt=args.txt, val_txt=args.val_txt, mode='train', is_used_edge=True, size=args.size,
                                   seed=args.seed)
    train_loader = DataLoader(dataset=train_dataset, batch_size=args.bz, num_workers=args.num_workers, shuffle=True)

    val_dataset = MVSSNetDataset(root=args.root, txt=args.txt, val_txt=args.val_txt, mode='val', is_used_edge=False, size=args.size,
                                 seed=args.seed)
    val_loader = DataLoader(dataset=val_dataset, batch_size=1, shuffle=False)
```

图 8: train.py

```python
    for epoch in range(load_epoch, args.max_epoches):
        model.train()
        losses.reset()
        t = tqdm(train_loader)
        if args.flag == 3:
            for idx, (image, mask, edge_mask, label) in enumerate(t):
                image, mask, edge_mask, label = image.cuda(), mask.cuda(), edge_mask.cuda(), label.cuda()
                # label = label.view(len(label), -1)
                optimizer.zero_grad()
                edge, outputs, clf = model(image, epoch+1)
                loss_seg = criterion_dice(outputs, mask)  # pixel loss
                loss_edge = criterion_dice(edge, edge_mask)  # edge loss
                clf = clf.reshape((len(clf), 1))
                loss_clf = criterion(clf.float(), label.float())
                loss = 0.16 * loss_seg + 0.04 * loss_clf + 0.8 * loss_edge
                loss.backward()
                optimizer.step()
                losses.update(loss.detach().cpu().numpy())
                t.set_description('epoch:{} | losses:{}'.format(epoch + 1, losses.avg))
            scheduler.step(losses.avg)
```

图 9: train.py

模型测试部分的实现：测试部分将训练中验证效果最好的模型保存下来对 CASIA1.0 数据集和 Columbia 数据集进行测试。

代码实现：test.py

```python
def test(args):
    if not os.path.exists(os.path.join(args.result_save_dir, args.model+args.suffix)):
        os.makedirs(os.path.join(os.path.join(args.result_save_dir, args.model+args.suffix), 'forgery'))
        os.makedirs(os.path.join(os.path.join(args.result_save_dir, args.model+args.suffix), 'gt'))
        os.makedirs(os.path.join(os.path.join(args.result_save_dir, args.model+args.suffix), 'mask'))
    assert os.path.exists(args.weight_save_dir)
    model = get_mvss(backbone='resnet50', pretrained_base=True, nclass=1, sobel=True, constrain=True, n_input=3, is_plus=False)
    if len(args.multi_gpu) > 1 and torch.cuda.device_count() > 1:
        model = torch.nn.DataParallel(model)
    model.cuda()
    model_path = "/data/liaogy/weight/mvssnet_casia2.0/experiment_2_epoch_76.pth"
    model.load_state_dict(torch.load(model_path, map_location='cpu')['state_dict'])

    test_dataset = MVSSNetDataset(root=args.root, txt=args.txt, val_txt=args.val_txt, mode='test', size=args.size, is_used_edge=False,
                                  seed=args.seed)
    test_loader = DataLoader(dataset=test_dataset, batch_size=1, shuffle=False)

    print("Now test model2 in ", args.val_txt)
```

图 10: test.py

```
# ././. test ././. #
model.eval()
iou = AverageMeter()
f1 = AverageMeter()
auc = AverageMeter()
t = tqdm(test_loader)
with torch.no_grad():
    for idx, (forgery, mask) in enumerate(t):
        if idx == args.test_num:
            exit()
        forgery, mask = forgery.cuda(), mask.cuda()
        edge, outputs = model(forgery, 1) # mvssnet
        # 像素级别的检测
        trans_mask = mask.detach().cpu().numpy()
        trans_mask = np.expand_dims(trans_mask, axis=1)
        trans_outputs = outputs.detach().cpu().numpy()
        auc.update(cal_auc(trans_outputs, trans_mask))
        f1.update(cal_f1(trans_outputs, trans_mask))
        iou.update(cal_iou(trans_outputs, trans_mask))
        trans_forgery = forgery[0].detach().cpu().numpy() # (c, h, w)
        trans_forgery = np.transpose(trans_forgery, (1, 2, 0)) # (h, w, c)
        trans_forgery = (trans_forgery * 255).astype(np.uint8) # [0, 1]->[0, 255]
        trans_forgery = cv2.cvtColor(trans_forgery, cv2.COLOR_RGB2BGR)
        trans_mask = mask[0].detach().cpu().numpy()  # (h, w)
        trans_mask = (trans_mask * 255).astype(np.uint8)
        trans_outputs = trans_outputs.squeeze()
        trans_outputs = (trans_outputs*255).astype(np.uint8)
        cv2.imwrite(os.path.join(os.path.join(args.result_save_dir, args.model+args.suffix),
                                 'forgery/forgery_{}.png'.format(idx)), trans_forgery)
        cv2.imwrite(os.path.join(os.path.join(args.result_save_dir, args.model+args.suffix),
                                 'gt/gt_{}.png'.format(idx)), trans_mask)
        cv2.imwrite(os.path.join(os.path.join(args.result_save_dir, args.model+args.suffix),
                                 'mask/mask_{}.png'.format(idx)), trans_outputs)
        t.set_description('auc:{:.3f} | f1:{:.3f} | iou:{:.3f} | '.format(auc.avg, f1.avg, iou.avg))
```

图 11: test.py

多尺度监督部分实现：像素级别的 loss 值是将篡改图像 forgery 和 groundTruth mask 通过损失函数 Dice loss 得到，这是目前分割图像最常见的 loss，得到每一个像素点的差值。边缘监督的 loss 值是通过 edge mask 和模型输出的 edge 通过损失函数 Dice loss 计算得到；图像级别的 loss 值计算首先对模型输出的像素级别的预测图进行全局最大池化，得到预测值中最大的预测值和 label(值有 0 和 1) 通过损失函数 BCE loss 计算得到。最后将三个 loss 值按照对应的权重相加起来得到最终的 loss 值进行传播。算法流程如图 21

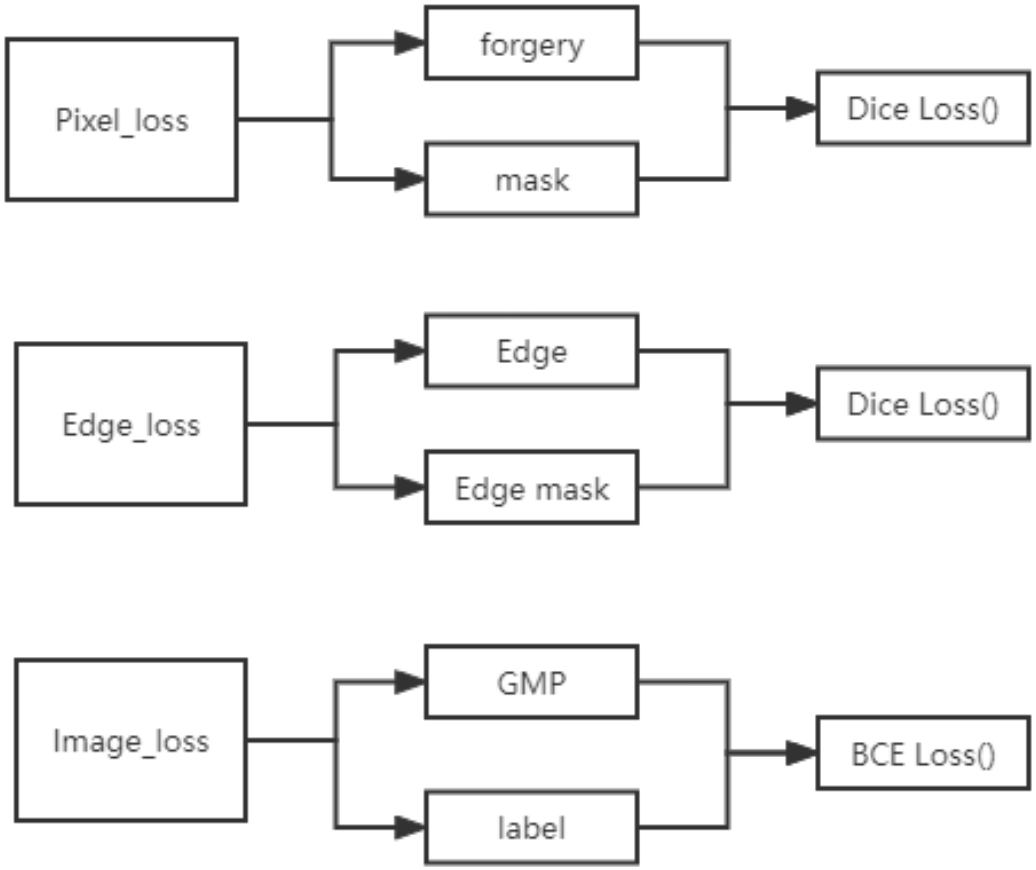


图 12: 多尺度监督的三个 loss 计算

代码实现：

```
loss_seg = criterion_dice(outputs, mask)  # pixel loss
loss_edge = criterion_dice(edge, edge_mask)  # edge loss
clf = clf.reshape((len(clf), 1))
loss_clf = criterion(clf.float(), label.float())
loss = 0.16 * loss_seg + 0.04 * loss_clf + 0.8 * loss_edge
```

图 13: loss 的计算

4.2 实验环境搭建

使用 pycharm 远程连接服务器，1080tiGPU 训练。

4.3 改进工作

论文中直接取像素级别概率图 S(x) 的最大值得到图像级的概率，图像分类损失实际上是基于 S(x) 的每一个像素计算的，损失仅通过唯一点（i）反向传播，这样的瓶颈不仅减慢了分类头的训练，而且阻碍了头部指导整个网络。同时 GMP 对于积极的反应以及空间的变化都是固定不变的，这会降低像素级别的检测效果。这里使用 Gem 对 GMP 进行改进，GeM 是 generalized mean pooling，出自 Fine-tuning CNN Image Retrieval with No Human Annotation，提出的是一种可学习的 pooling layer，可提高检索性能。GeM Pooling 可以看作 Average Pooling 和 Max Pooling 的延申，当 p=1 时，GeM Pooling 退化成 Average Pooling，当 p 无穷大时，GeM pooling 等效于 Max Pooling.GeM 包含可学习的参数 p，对输入样本先求 p 次幂，然后取均值，在进行 p 次开方。GeM 目前已经成为了图像检索池化操作的主流使用

方法，公式如下：

$$\mathrm{GeM}(S(x)) = \frac{1}{W \times H}\left(\sum_{i=1}^{W}\sum_{j=1}^{H} S\left(x_{i,j}\right)^{p}\right)^{\frac{1}{p}} \tag{4}$$

GeM 可以解决 GMP 单一计算的缺点，但是仍然对空间变化是不变的，这里考虑使用卷积解决这个问题。由于卷积自然地捕获像素之间的空间相关性，因此可以考虑提前向 GeM 添加卷积块，由 Conv(S(x)) 表示。因此，获得 C(x) 作为 GeM(Conv(S(x)))。注意，在早期的训练阶段，网络，特别是其分割头部 Gseg，没有得到良好的训练，因此主要产生无意义的 S(x)。Conv 将进一步夸大 Gclf 的这种噪声输入，从而使分类头和整个网络难以训练。为了抑制这种负效应，我们通过由非负超参数 λ 加权的衰减跳跃连接将 GeM(S(x)) 添加到 C(x)。如图 21

$$C(x) = \lambda \cdot \mathrm{GeM}(S(x)) + (1-\lambda) \cdot \mathrm{GeM}(\mathrm{Conv}(S(x))) \tag{5}$$

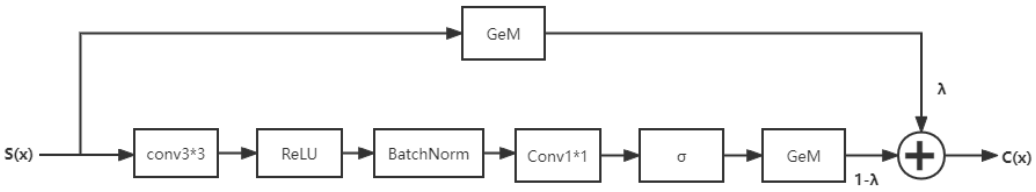


图 14: GEMConv

```python
class GeM(nn.Module):
    def __init__(self, p=3, eps=1e-6):
        super(GeM, self).__init__()
        self.p = nn.Parameter(torch.ones(1) * p)
        self.eps = eps

    def forward(self, x):
        return self.gem(x, p=self.p, eps=self.eps)

    def gem(self, x, p=3, eps=1e-6):
        return F.avg_pool2d(x.clamp(min=eps).pow(p), (x.size(-2), x.size(-1))).pow(1. / p)

    def __repr__(self):
        return self.__class__.__name__ + '(' + 'p=' + '{:.4f}'.format(self.p.data.tolist()[0]) + ', ' + 'eps=' + str(
            self.eps) + ')'
```

图 15: GeM

```python
class ConvGeM(nn.Module):
    def __init__(self, in_channels):
        super(ConvGeM, self).__init__()
        self.conv3 = nn.Conv2d(in_channels, 32, kernel_size=3,padding=1, stride=1)
        self.relu = nn.ReLU()
        self.bn = nn.BatchNorm2d(32)
        self.conv1 = nn.Conv2d(32, 1, kernel_size=1, padding=0, stride=1)
        self.gem1 = GeM()
        self.gem2 = GeM()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x, epoch):
        a = 0.9975**(epoch*epoch)
        res = self.conv3(x)
        res = self.relu(res)
        res = self.bn(res)
        res = self.conv1(res)
        res = self.sigmoid(res)
        res = self.gem2(res)
        x = self.gem1(x)
        return a*x+(1-a)*res
```

图 16: ConvGeM

5 实验结果分析

实验结果将训练中验证效果最好的模型保存下来进行测试，下表展示了改进前和改进后的模型分别在数据集 CASIA 和 Columbia 上的 AUC 和 F1score 的数值。如下表是改进前和改进后的 MVSSNet 模型分别在数据集 CASIA1.0 和 Columbia 上的 AUC 数值，可以看到改进后的模型在 CASIA1.0 上的 AUC 数值比起论文作者得到的 0.753 和改进前复现的结果 0.720 要大一些，数值为 0.814。

Dataset	CASIA1.0	COLUMBIA
改进前 MVSSNet	0.753/0.720	0.703/0.635
改进后 MVSSNet	0.753/0.814	0.703/0.735

如下表是改进前和改进后的 MVSSNet 模型分别在数据集 CASIA1.0 和 Columbia 上的 F1score 数值。

Dataset	CASIA1.0	COLUMBIA
改进前 MVSSNet	0.445	0.327
改进后 MVSSNet	0.427	0.264

```
ssh://liaogy@172.31.224.54:22/home/liaogy/anaconda3/envs/python37/bin/python -u /home/liaogy/lgy/mvssnet3/main.py
load pretrain success
----------use sobel-------------
----------use constrain-------------
load pretrain success
Now test model1 in  casia1.txt
auc:0.814 | f1:0.445 | iou:0.090 | : 100%|████| 920/920 [02:15<00:00,  6.79it/s]
```

图 17

```
ssh://liaogy@172.31.224.54:22/home/liaogy/anaconda3/envs/python37/bin/python -u /home/liaogy/lgy/mvssnet3/main.py
load pretrain success
----------use sobel-------------
----------use constrain-------------
load pretrain success
Now test model1 in  columbia.txt
auc:0.735 | f1:0.264 | iou:0.270 | : 100%|████| 180/180 [00:33<00:00,  5.38it/s]

Process finished with exit code 0
```

图 18: GEMConv

```
ssh://liaogy@172.31.224.54:22/home/liaogy/anaconda3/envs/python37/bin/python -u /home/liaogy/lgy/mvssnet3/main.py
load pretrain success
----------use sobel-------------
----------use constrain-------------
load pretrain success
Now test model2 in  columbia.txt
auc:0.635 | f1:0.427 | iou:0.270 | : 100%|████| 180/180 [00:25<00:00,  7.15it/s]

Process finished with exit code 0
```

图 19: GEMConv

```
ssh://liaogy@172.31.224.54:22/home/liaogy/anaconda3/envs/python37/bin/python -u /home/liaogy/lgy/mvssnet3/main.py
load pretrain success
----------use sobel------------
----------use constrain------------
load pretrain success
Now test model2 in  casia1.txt
auc:0.720 | f1:0.327 | iou:0.090 | : 100%|████| 920/920 [02:26<00:00,  6.30it/s]

Process finished with exit code 0
```

图 20: GEMConv

如图 21 是模型检测定位的效果图，可以从图中看出模型基本能够检测定位出图像的篡改区域：

图 21: 检测定位效果

6 总结与展望

本次复现工作首先调研了图像篡改定位的相关工作，了解到在深度学习之前图像篡改检测主要是靠手工特征和滑动窗口完成篡改图像的检测和定位，深度学习发展之后借助卷积神经网络局部提取特征的特点实现篡改特征的学习从而完成检测和定位。其中全卷积神经网络是为图像篡改定位领域一个表现不错的网络结构，MVSSNet 模型就是在 FCN 的基础上进行改进，考虑到模型的敏感性和特异性，增加了不同的支路来增强模型的效果。其中多视图学习特征支路通过边缘监督分支和噪声敏感分支将容易出现篡改痕迹的位置都进行重点检测；多尺度监督支路通过三个方面的 loss 值，综合考虑到模型的特异性和敏感性。复现工作主要是完成模型的训练测试的实现，初步掌握了神经网络模型训练的代码框架，对神经网络模型的调参有了动手上的体验，同时也在实际的复现过程中更加理解模型提取图像特征以及预测的过程。论文的改进工作是在 MVSSNet++ 基础上展开，考虑到只是简单地用全局最大池化来定义图像级别的预测值过于局限，所以尝试了平均池化和最大池化的延伸 GeM 和卷积操作来计算得到图像级别的预测值。效果相比之前要好一些。未来会继续学习图像篡改定位相关的知识，增强自身的科研能力。

参考文献

[1] HU X, ZHANG Z, JIANG Z, et al. SPAN: Spatial Pyramid Attention Network forImage Manipulation Localization[J]. arXiv: Computer Vision and Pattern Recognition, 2020.

[2] BAMMEY Q, von GIOI R G, MOREL J M. An Adaptive Neural Network for Unsupervised Mosaic Consistency Analysis in Image Forensics[J]. computer vision and pattern recognition, 2020.

[3] YUE W, WAEL A, PREMKUMAR N. ManTra-Net: Manipulation Tracing Network for Detection and Localization of Image Forgeries With Anomalous Features[J]. IEEE Conference Proceedings, 2019.

[4] YANG C, LI H, LIN F, et al. Constrained R-CNN: A general image manipulation detection model[J]. international conference on multimedia and expo, 2019.

[5] YU C, WANG J, PENG C, et al. Learning a Discriminative Feature Network for Semantic Segmentation [J]. computer vision and pattern recognition, 2018.

[6] WEI Q, LI X, YU W, et al. Learn to Segment Retinal Lesions and Beyond[J]. International Conference on Pattern Recognition, 2019.

[7] BIANCHI T, PIVA A. Image Forgery Localization via Block-Grained Analysis of JPEG Artifacts[J]. IEEE Transactions on Information Forensics and Security, 2012.

[8] CHIERCHIA G, POGGI G, SANSONE C, et al. A Bayesian-MRF Approach for PRNU-Based Image Forgery Detection[J]. IEEE Transactions on Information Forensics and Security, 2014.

[9] FAN W, WANG K, CAYRE F. General-purpose image forensics using patch likelihood under image statistical models[J]. international workshop on information forensics and security, 2015.