图像篡改检测

一、背景介绍

在篡改技术中，拼接（splicing）、复制-移动（copy-move）和移除（removal）是最常见的操作。图像拼接从真实图像中复制区域并将其粘贴到其他图像中。复制-移动操作选取某一图像中的区域，复制粘贴给自身。移除操作从真实图像中消除区域并进行绘制。图 1 是这三种操作的简单示例，其中每行对应一种篡改方法；第一列为真实图像，第二列为篡改后的图像，第三列为篡改区域的掩码。

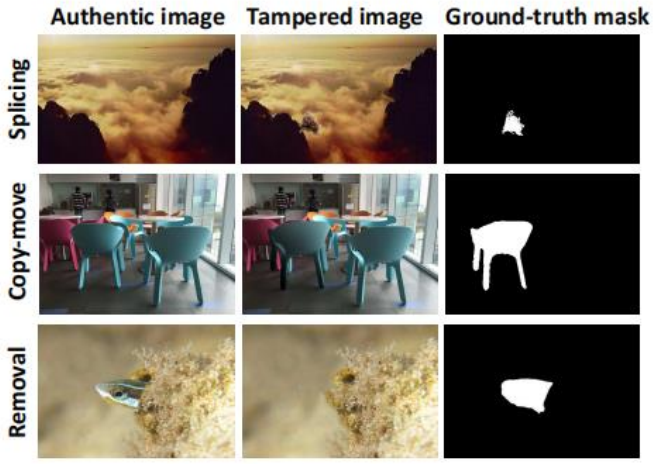


图 1 图像篡改示例

随着多媒体采集设备的快速发展和众多操作简单的图像编辑软件的出现，普通人也能很轻易地对图像进行加工和修改。图 2 所示为利用 IpasOS16 自动提取图像主体[1]，从而实现图像拼接的示例。

图 2 利用 IpadOS16 实现图像拼接

随着计算机和互联网的飞速发展，数字图像的存储和传递也变得简单，多媒体已经成为信息承载与共享的重要途径。日常生活中人们对图像进行修改，往往是出于美化、娱乐的目

的；但是在有些情况下，被恶意篡改的图像经过传播，会影响人们对客观事物的判断，有时甚至会对社会和国家造成不良的影响。

2005 年韩国首尔大学的黄禹锡干细胞造假事件的调查结果表明，黄从 11 个病人干细胞株上所衍生的干细胞，实际存在的只有两个，另外九个干细胞的照片是由这两个干细胞照片修改而来，随后《science》正式宣布撤回黄禹锡 2004 年和 2005 年发表的两篇论文。

2008 年 BBC 新闻报道伊朗火箭发射的境况，其中只有两只火箭是真实的，其它的都经过了图像篡改，不仅夸大了事实，而且给人们造成了不必要的恐慌。

图 3 伊朗火箭发射图像篡改(左为原图)

在没有经过专业训练的情况下，人们可能很难识别出一张图片是否经过篡改以及篡改的区域。自动地进行图像篡改检测可以帮助人们识别虚假消息，一定程度上遏制学术不端、新闻造假等现象，减轻图像篡改带来的不良影响。

二、算法选取与实现

Zhou 等人[2]提出了一种双流图像篡改检测框架 RGB-N，如图 4 所示，它不仅建模了视觉篡改，而且还捕获了局部噪声特征的不一致性。其中视觉信息为 RGB 格式的图像信息，噪声指一个像素的值和该像素的估计值（使用周围像素产生的插值）之间的残差。

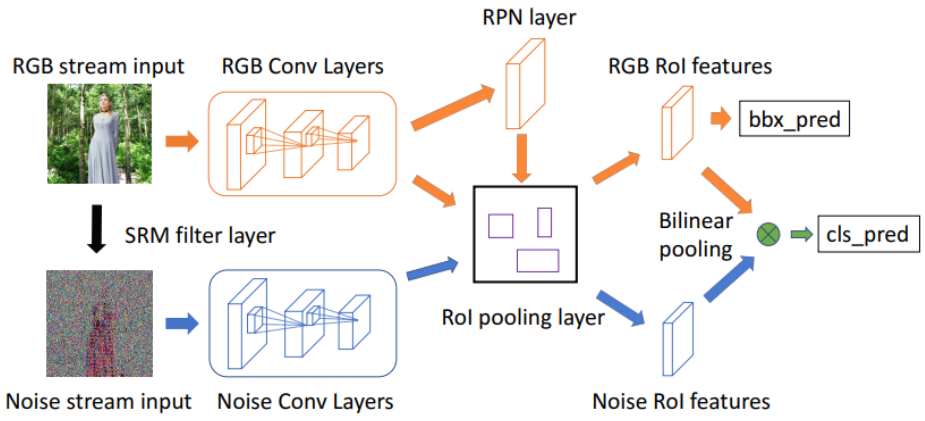


图 4 双流图像篡改检测框架 RGB-N

RGB-N 基于 Faster R-CNN 框架，该框架常用于目标检测，主要分为四个阶段：（1）提取图片特征的卷积层；（2）找出物体区域的 RPN（Region Proposal Network）层；（3）RoI（Region of Interest）池化层接受卷积层和 RPN 层的输出，获得框选出的物体的特征；（4）RGB-N 使用双线性池化来结合视觉流和噪声流的空间共现特征，随后使用全连接层和 softmax 估计篡改概率。

表 1 展示了代码主要文件及说明。

表 1 代码主要文件及说明

路径		文件名	说明
./		main.py	训练模型
./		test.py	调用模型并可视化预测结果
_lib	config	config.py	参数设置
	datasets	-	数据准备
	layer_utils	anchor_target_layer.py	为 RPN 层生成监督数据
		generate_anchors.py	生成 anchor
		proposal_layer.py	根据 RPN 的输出选择物体框，使用 NMS
		proposal_top_layer.py	根据 RPN 的输出选择物体框，不使用 NMS
		proposal_target_layer.py	训练时用，限制与 ground-truth mask 的 IoU 值过低的框的数目
		roi_data_layer.py	生成训练数据
	nets	network.py	模型基类
		vgg16.py	采用 VGG16 的 RGB-N 模型
./default/gene_2007_trainval/default		-	模型参数保存位置
./data		-	数据集，VGG16 的参数

(1) 卷积层

RGB 卷积层与噪声卷积层都使用 VGG 网络[3]，对应代码分别为_lib → nets → vgg16.py 中 vgg 类的 build_head_for_RGB 函数和 build_head_for_Noise 函数。

```python
def build_head_for_RGB(self, is_training):

    # Main network
    # Layer  1
    net = slim.repeat(self._image, 2, slim.conv2d, 64, [3, 3], trainable=False, scope='conv1')
    net = slim.max_pool2d(net, [2, 2], padding='SAME', scope='pool1')

    # Layer 2
    net = slim.repeat(net, 2, slim.conv2d, 128, [3, 3], trainable=False, scope='conv2')
    net = slim.max_pool2d(net, [2, 2], padding='SAME', scope='pool2')

    # Layer 3
    net = slim.repeat(net, 3, slim.conv2d, 256, [3, 3], trainable=is_training, scope='conv3')
    net = slim.max_pool2d(net, [2, 2], padding='SAME', scope='pool3')

    # Layer 4
    net = slim.repeat(net, 3, slim.conv2d, 512, [3, 3], trainable=is_training, scope='conv4')
    net = slim.max_pool2d(net, [2, 2], padding='SAME', scope='pool4')

    # Layer 5
    net = slim.repeat(net, 3, slim.conv2d, 512, [3, 3], trainable=is_training, scope='conv5')

    # Append network to summaries
    self._act_summaries.append(net)

    # Append network as head layer
    self._layers['head'] = net

    return net
```

图 5 RGB 卷积层代码

```python
def build_head_for_Noise(self, is_training, initializer, initializer_srm):

    def truncate_2(x):
        neg = ((x + 2) + abs(x + 2)) / 2 - 2
        return -(2 - neg + abs(2 - neg)) / 2 + 2

    # Main network
    # Layer SRM
    net = slim.conv2d(self._image, 3, [5, 5], trainable=False, weights_initializer=initializer_srm,
                      activation_fn=None, padding='SAME', stride=1, scope='srm')
    net = truncate_2(net)

    # Layer  1
    net = slim.repeat(net, 2, slim.conv2d, 64, [3, 3], trainable=is_training, weights_initializer=initializer, scope='conv1n')
    net = slim.max_pool2d(net, [2, 2], padding='SAME', scope='pool1n')

    # Layer 2
    net = slim.repeat(net, 2, slim.conv2d, 128, [3, 3], trainable=is_training, weights_initializer=initializer, scope='conv2n')
    net = slim.max_pool2d(net, [2, 2], padding='SAME', scope='pool2n')

    # Layer 3
    net = slim.repeat(net, 3, slim.conv2d, 256, [3, 3], trainable=is_training, weights_initializer=initializer, scope='conv3n')
    net = slim.max_pool2d(net, [2, 2], padding='SAME', scope='pool3n')

    # Layer 4
    net = slim.repeat(net, 3, slim.conv2d, 512, [3, 3], trainable=is_training, weights_initializer=initializer, scope='conv4n')
    net = slim.max_pool2d(net, [2, 2], padding='SAME', scope='pool4n')

    # Layer 5
    net = slim.repeat(net, 3, slim.conv2d, 512, [3, 3], trainable=is_training, weights_initializer=initializer, scope='conv5n')

    # Append network to summaries
    self._act_summaries.append(net)

    # Append network as head layer
    self._layers['head2'] = net

    return net
```

图 6 噪声卷积层代码

噪声卷积层比 RGB 卷积层多一个 SRM 滤波器[4]子层，得到的噪声特征图包含的信息为局部噪声。在 SRM 滤波器子层中我们使用 3 个滤波器核，如图 7 所示，其定义和初始化代码在_lib → nets → vgg16.py 中 vgg 类的 build_network 函数中。

$$\frac{1}{4}\begin{bmatrix}0&0&0&0&0\\0&-1&2&-1&0\\0&2&-4&2&0\\0&-1&2&-1&0\\0&0&0&0&0\end{bmatrix}\quad\frac{1}{12}\begin{bmatrix}-1&2&-2&2&-1\\2&-6&8&-6&2\\-2&8&-12&8&-2\\2&-6&8&-6&2\\-1&2&-2&2&-1\end{bmatrix}\quad\frac{1}{2}\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&1&-2&1&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}$$

图 7 SRM 滤波器核

(2) RPN 层

我们选取尺寸为 8px、16px、32px，长宽比为 1:2、1:1 和 2:1 的 anchor。PRN 层仅接受 RGB 卷积输出的特征作为输入，首先通过一个 3*3*512 的卷积核，然后通过两个并行的 1*1*2k，1*1*4k 的卷积内核，其中 k 为 Anchor 数量。1*1*2k 的卷积核负责分类，以预测框内是否含有物体；1*1*4k 的卷积核负责预测 anchor 的偏移值，即 $(\Delta x_{center}, \Delta y_{center}, \Delta width, \Delta height)$。

PRN 层的实现代码为_lib → nets → vgg16.py 中 vgg 类的 build_rpn 函数。

```
def build_rpn(self, net, is_training, initializer):

    # Build anchor component
    self._anchor_component()

    # Create RPN Layer
    rpn = slim.conv2d(net, 512, [3, 3], trainable=is_training, weights_initializer=initializer, scope="rpn_conv/3x3")

    self._act_summaries.append(rpn)
    rpn_cls_score = slim.conv2d(rpn, self._num_anchors * 2, [1, 1], trainable=is_training, weights_initializer=initializer, padding='VALID', activation_fn=None, scope='rpn_cls_score')

    # Change it so that the score has 2 as its channel size
    rpn_cls_score_reshape = self._reshape_layer(rpn_cls_score, 2, 'rpn_cls_score_reshape')
    rpn_cls_prob_reshape = self._softmax_layer(rpn_cls_score_reshape, "rpn_cls_prob_reshape")
    rpn_cls_prob = self._reshape_layer(rpn_cls_prob_reshape, self._num_anchors * 2, "rpn_cls_prob")
    rpn_bbox_pred = slim.conv2d(rpn, self._num_anchors * 4, [1, 1], trainable=is_training, weights_initializer=initializer, padding='VALID', activation_fn=None, scope='rpn_bbox_pred')
    return rpn_cls_prob, rpn_bbox_pred, rpn_cls_score, rpn_cls_score_reshape
```

图 8 RPN 层代码

由于我们使用的是有监督学习，如果模型处于训练状态，我们还需自动给 anchor 打上（物体，前景）的标签。计算 anchor 与 ground-truth mask 的 IoU 值，将 IoU < 0.3 的标记为“前景”，将 IoU > 0.7 的标记为“物体”，忽略 0.3 至 0.7 的 anchor。然后再对有标签的 anchor 进行采样，平衡两种标签的占比，保存最后得到的信息。对应代码为_lib → layer_utils → anchor_target_layer.py 中的 anchor_target_layer 函数，该函数被_lib → nets → vgg16.py 中 vgg 类的 build_proposals 函数调用。

(3) RoI 池化层

经过了 RPN 网络，我们得到了一些框及其含有物体的概率，接下来我们需要选择一些框并获取它们的特征。特征的获取方式为：将得到的建议框对应的特征图裁剪出来，使用双线性插值调整到固定大小，再使用最大池化进行调整。

使用_lib → layer_utils → proposal_ layer.py 中的 proposal_ layer 函数或_lib → layer_utils → proposal_ top_layer.py 中的 proposal_top_ layer 函数对 RPN 输出的框进行排序和选择，并得到格式为$(x_{min}, y_{min}, x_{max}, y_{max})$的框。其中 proposal_ layer 函数使用了非极大值抑制（Non-Maximum Suppression, NMS）[5]。两函数被_lib → nets → vgg16.py 中 vgg 类的 build_proposals 函数调用。

再使用_lib → nets → network.py 中 Network 类的_crop_pool_layer 函数实现特征的抽取，特征图的大小为 7*7。该函数被_lib → nets → vgg16.py 中 vgg 类的 build_predictions 函数调用。

```
def _crop_pool_layer(self, bottom, rois, name):
    with tf.variable_scope(name):
        batch_ids = tf.squeeze(tf.slice(rois, [0, 0], [-1, 1], name="batch_id"), [1])
        # Get the normalized coordinates of bboxes
        bottom_shape = tf.shape(bottom)
        height = (tf.to_float(bottom_shape[1]) - 1.) * np.float32(self._feat_stride[0])
        width = (tf.to_float(bottom_shape[2]) - 1.) * np.float32(self._feat_stride[0])
        x1 = tf.slice(rois, [0, 1], [-1, 1], name="x1") / width
        y1 = tf.slice(rois, [0, 2], [-1, 1], name="y1") / height
        x2 = tf.slice(rois, [0, 3], [-1, 1], name="x2") / width
        y2 = tf.slice(rois, [0, 4], [-1, 1], name="y2") / height
        # Won't be backpropagated to rois anyway, but to save time
        bboxes = tf.stop_gradient(tf.concat([y1, x1, y2, x2], axis=1))
        pre_pool_size = cfg.temps['roi_pooling_size'] * 2
        crops = tf.image.crop_and_resize(bottom, bboxes, tf.to_int32(batch_ids), [pre_pool_size, pre_pool_size], name="crops")

    return slim.max_pool2d(crops, [2, 2], padding='SAME')
```

图 9 RoI 特征抽取

(4) 双线性池化和预测

由 RoI 输出的特征分别为视觉流特征和噪声流特征，因此引入双线性池化将两个特征进行外积（outer product），把 RGB 流和噪声流结合到一起的同时保留了空间信息。这里我们使用 Compact Bilinear Pooling[6]，实现代码为_lib → utils → compact_bilinear_pooling_layer.py，被_lib → nets → vgg16.py 中 vgg 类的 build_predictions 函数调用。

预测部分为 3 层全连接层+softmax，位于_lib → nets → vgg16.py 中 vgg 类的 build_predictions 函数中。

三、效果展示

RGB-N 可以理解为目标检测的变体，我们将其生成的框和预测值进行了可视化演示。可以使用--pic_format 设置图片格式，默认为全部文件。使用--conf_thresh 设置可视化部分显示框的最低篡改评估分数，低于该值的篡改预测分数框不被显示，默认值为 0.2。使--gpu 设置是否使用 GPU，没有则使用 CPU。

(1) 数据集自动生成的图片

数据集自动生成的图片为 JPG 格式，所以你可以这样调用：

python test.py --pic_format=".jpg"

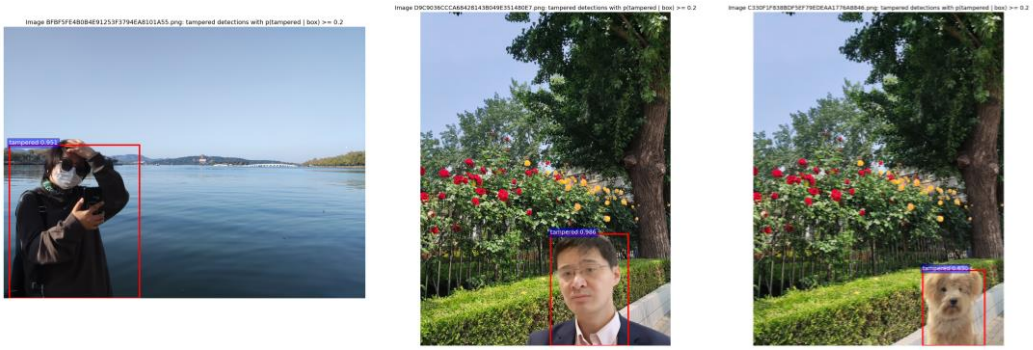
组图 10 对数据集自动生成的图片进行篡改预测

由于是自动生成的图片，所以人眼看可能也会比较明显。最右侧的图表示该模型具有应对真实图片的能力（未框选物体）。

(2) 我们拼接得到的图片

利用 IpadOS 的自动提取物体的功能，我们生成了一些拼接图片。这些图片的格式均为 PNG 格式，所以你可以这样调用：

python test.py --pic_format=".png"



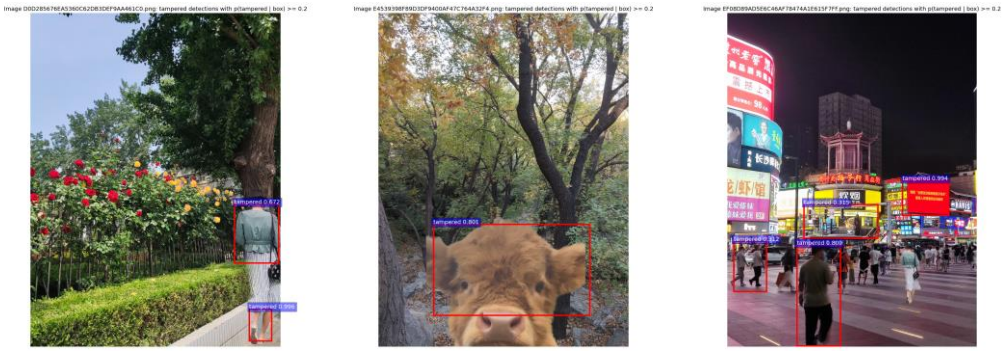
组图 11 对我们拼接得到的图片进行篡改预测

我们使用的图片并未出现在数据集中，可见模型具有一定的鲁棒性。

(3) 错误分析

RGB-N 基于目标检测框架 Faster R-CNN，使用视觉流和噪声流特征对图像篡改进行预测，难免会出现目标检测精度不够、预测错误的情况，如组图 12 所示。

最左边的一张图和中间一张图均为篡改预测框为篡改区域的一部分的情况，可以尝试通过改进 RPN 与 RoI 之间的框选算法来改进。最右边的图则展现了视觉流可能误导图像篡改检测，检测出错误的结果，这可能是原图像中人物的残影、物体模糊等导致的。



组图 12 错误分析

四、总结与感悟

选择的文章发自 2018 年，是较早的技术。图像篡改技术如今已有很大的改变，包括润饰、变种、增强、计算机生成、AI 绘制等。许多安全问题的发展都是攻防对抗史，图像篡改技术与图像篡改检测也不例外。因此，在图像篡改领域总是还有许多问题值得研究，期待看到这项技术投入使用的那天！

参考文献

[1] iPadOS 16 - Apple (中国大陆)[DB/OL].[2022-11-22].https://www.apple.com.cn/ipados/ipados-16/

[2] Zhou P, Han X, Morariu V I, et al. Learning rich features for image manipulation detection[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 1053-1061.

[3] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

[4] Fridrich J, Kodovsky J. Rich models for steganalysis of digital images[J]. IEEE Transactions on information Forensics and Security, 2012, 7(3): 868-882.

[5] Neubeck A , Gool L J V . Efficient Non-Maximum Suppression[C]// 18th International Conference on Pattern Recognition (ICPR 2006), 20-24 August 2006, Hong Kong, China. IEEE Computer Society, 2006.

[6] Lin T Y, RoyChowdhury A, Maji S. Bilinear CNN models for fine-grained visual recognition[C]//Proceedings of the IEEE international conference on computer vision. 2015: 1449-1457.

附录

代码参考：WaLittleMoon/Learning-Rich-Features-for-Image-Manipulation-Detection: 基于双流 Faster R-CNN 网络的 图像篡改检测 (github.com)

代码链接：https://pan.baidu.com/s/1VgPNvpcWnMqeTLXP1ZjN_A?pwd=mv1c 提取码：mv1c