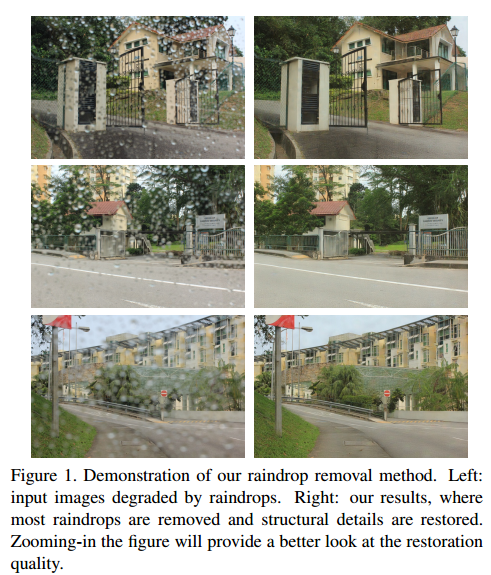
论文题目：Attentive Generative Adversarial Network for Raindrop Removal from A Single Image

论文下载地址：<https://arxiv.org/abs/1711.10098>

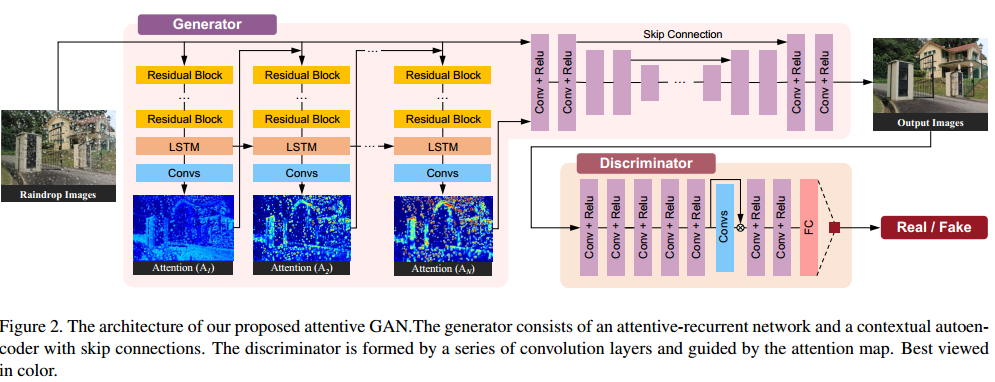
Github代码：<https://github.com/MaybeShewill-CV/attentive-gan-derainnet>

数据集下载地址：

## <https://drive.google.com/drive/folders/1e7R76s6vwUJxILOcAsthgDLPSnOrQ49K>



这是一篇结合attention机制和GAN去除雨滴的文章。从上图可以看出这个方法去除雨滴的效果非常好。下面介绍一下这个网络。



该网络共包含三个部分，分别为： Attention-recurrent Network，Context Autoencoder 和 Dicriminator Network。前两个部分构成了生成器。

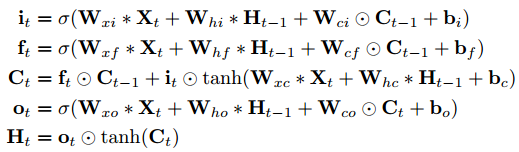
第一部分主要的工作是做检测（即检测雨滴在图片中的位置），然后生成 attention map，它由先前的图片中得出，每一层由若干个Residual Block，一个LSTM和一个Convs组成。需要说明的是，图中的每一个“Block”由5层ResNet组成，其作用是得到输入图片的精确特征与前一个Block的模。具体做法是首先使用 Residual block 从雨滴图片中抽取 feature，渐进式地使用 Convs 来检测 attentive 的区域。训练数据集中图片都是成对的，所以可以很容易计算出相应的 mask（M），由此可以构建出 Loss 函数；由于不同的 attention 网络刻画 feature 的准确度不同，所以给每个 loss 一个指数的衰减。相应的 loss 函数如下：

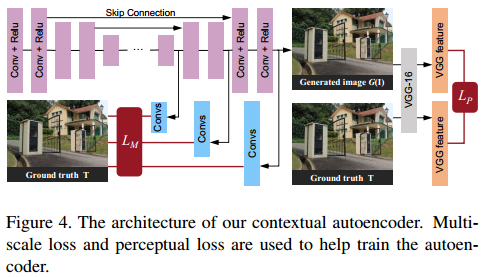






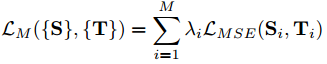
LSTM公式如下：





随后将 attention map 和雨滴图像一起送给 autoencoder，生成去雨滴的超分辨率图像。由于attention map的给出，这一部分的工作类似于将attention map中attention值较高的部分通过该部分周围的图片信息形成的新的色块进行替换，从而实现图片信息的还原。autoencoder 的结构用了 16 个 Con和 Relu。为了避免网络本身造成的 blur，作者使用了 skip connection，因为在低级层次会带来很好的效果。在构建 loss 方面，除了多尺度的考虑，还加上了一个高精度的 loss，分别是：Multi-scale loss 和perceptual loss.

Multi-scale loss:

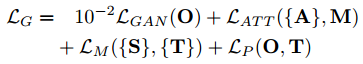


Perceptual loss:

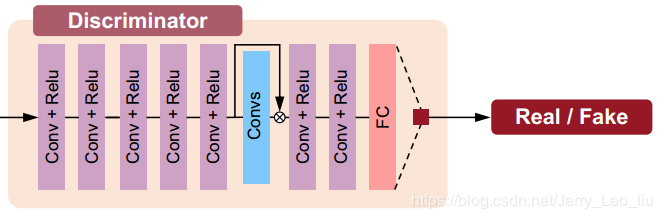




结合两个部分，得到生成器的损失，如下：



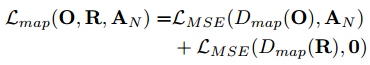




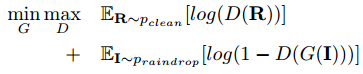
最后一个是 discriminator。这个步骤有两部分，一种是只使用 autoencoder 生成的无雨滴图像，进行判断；另一种则是加入 attention map 作为指导。

判别器的损失函数是：





综上，整体损失为：



下面说一下实验部分。

生成器中残差块的实现：

1. def \_residual\_block(self, input\_tensor, name):
2. """
3. attentive recurrent net中的residual block
4. :param input\_tensor:
5. :param name:
6. :return:
7. """
8. output = None
9. with tf.variable\_scope(name):
10. inputs = input\_tensor
11. shortcut = input\_tensor
12. **for** i **in** range(5):
13. **if** i == 0:
14. inputs = self.conv2d(inputdata=inputs,out\_channel=32,

kernel\_size=3,padding='SAME',stride=1,use\_bias=False,

name='block\_{:d}\_conv\_1'.format(i))

1. # TODO reimplement residual block
2. inputs = self.lrelu(inputdata=inputs, name='block\_{:d}\_relu\_1'.format(i + 1))
3. output = inputs
4. shortcut = output
5. **else**:
6. inputs = self.conv2d(inputdata=inputs,out\_channel=32,
7. kernel\_size=1, padding='SAME', stride=1,
8. use\_bias=False, name='block\_{:d}\_conv\_1'.format(i))
9. inputs = self.lrelu(inputdata=inputs, name='block\_{:d}\_conv\_1'.format(i + 1))
10. inputs = self.conv2d(inputdata=inputs, out\_channel=32,
11. kernel\_size=1, padding='SAME', stride=1,
12. use\_bias=False, name='block\_{:d}\_conv\_2'.format(i))
13. inputs = self.lrelu(inputdata=inputs, name='block\_{:d}\_conv\_2'.format(i + 1))
15. output = self.lrelu(inputdata=tf.add(inputs, shortcut),
16. name='block\_{:d}\_add'.format(i))
17. shortcut = output
18. **return** output

## 注意力机制的实现：

1. **def** build\_attentive\_rnn(self, input\_tensor, name, reuse=False):
2. """
3. Generator的attentive recurrent部分, 主要是为了找到attention部分
4. :param input\_tensor:
5. :param name:
6. :param reuse:
7. :return:
8. """
9. [batch\_size, tensor\_h, tensor\_w, \_] = input\_tensor.get\_shape().as\_list()
10. with tf.variable\_scope(name, reuse=reuse):
11. init\_attention\_map = tf.constant(0.5, dtype=tf.float32,
12. shape=[batch\_size, tensor\_h, tensor\_w, 1])
13. init\_cell\_state = tf.constant(0.0, dtype=tf.float32,
14. shape=[batch\_size, tensor\_h, tensor\_w, 32])
15. init\_lstm\_feats = tf.constant(0.0, dtype=tf.float32,
16. shape=[batch\_size, tensor\_h, tensor\_w, 32])
17. attention\_map\_list = []
19. **for** i **in** range(4):
20. attention\_input = tf.concat((input\_tensor, init\_attention\_map), axis=-1)
21. conv\_feats = self.\_residual\_block(input\_tensor=attention\_input,   name='residual\_block\_{:d}'.format(i + 1))
22. lstm\_ret = self.\_conv\_lstm(input\_tensor=conv\_feats,
23. input\_cell\_state=init\_cell\_state,
24. name='conv\_lstm\_block\_{:d}'.format(i + 1)
25. init\_attention\_map = lstm\_ret['attention\_map']
26. init\_cell\_state = lstm\_ret['cell\_state']
27. init\_lstm\_feats = lstm\_ret['lstm\_feats']
28. attention\_map\_list.append(lstm\_ret['attention\_map'])
29. ret = {
30. 'final\_attention\_map': init\_attention\_map,
31. 'final\_lstm\_feats': init\_lstm\_feats,
32. 'attention\_map\_list': attention\_map\_list
33. }
34. **return** ret

## 自编码器的实现：

1. def build\_autoencoder(self, input\_tensor, name, reuse=False):
2. """
3. Generator的autoencoder部分, 负责获取图像上下文信息
4. :param input\_tensor:
5. :param name:
6. :param reuse:
7. :**return**:
8. """
9. with tf.variable\_scope(name, reuse=reuse):
10. conv\_1 = self.conv2d(inputdata=input\_tensor, out\_channel=64, kernel\_size=5,  padding='SAME', stride=1, use\_bias=False, name='conv\_1')
11. relu\_1 = self.lrelu(inputdata=conv\_1, name='relu\_1')
12. conv\_2 = self.conv2d(inputdata=relu\_1, out\_channel=128, kernel\_size=3, padding='SAME', stride=2, use\_bias=False, name='conv\_2')
13. relu\_2 = self.lrelu(inputdata=conv\_2, name='relu\_2')
14. conv\_3 = self.conv2d(inputdata=relu\_2, out\_channel=128, kernel\_size=3, padding='SAME',  stride=1, use\_bias=False, name='conv\_3')
15. relu\_3 = self.lrelu(inputdata=conv\_3, name='relu\_3')
16. conv\_4 = self.conv2d(inputdata=relu\_3, out\_channel=128, kernel\_size=3,  padding='SAME', stride=2, use\_bias=False, name='conv\_4')
17. relu\_4 = self.lrelu(inputdata=conv\_4, name='relu\_4')
18. conv\_5 = self.conv2d(inputdata=relu\_4, out\_channel=256, kernel\_size=3, padding='SAME',  stride=1, use\_bias=False, name='conv\_5')
19. relu\_5 = self.lrelu(inputdata=conv\_5, name='relu\_5)
20. conv\_6 = self.conv2d(inputdata=relu\_5, out\_channel=256, kernel\_size=3, padding='SAME',  stride=1, use\_bias=False, name='conv\_6')
21. relu\_6 = self.lrelu(inputdata=conv\_6, name='relu\_6')
22. dia\_conv1 = self.dilation\_conv(input\_tensor=relu\_6, k\_size=3, out\_dims=256, rate=2, padding='SAME', use\_bias=False, name=' dia\_conv\_1')
23. relu\_7 = self.lrelu(dia\_conv1, name='relu\_7')
24. dia\_conv2 = self.dilation\_conv(input\_tensor=relu\_7, k\_size=3, out\_dims=256, rate=4,  padding='SAME', use\_bias=False, name=' dia\_conv\_2')
25. relu\_8 = self.lrelu(dia\_conv2, name='relu\_8')
26. dia\_conv3 = self.dilation\_conv(input\_tensor=relu\_8, k\_size=3, out\_dims=256, rate=8, padding='SAME', use\_bias=False, name=' dia\_conv\_3')
27. relu\_9 = self.lrelu(dia\_conv3, name='relu\_9')
28. dia\_conv4 = self.dilation\_conv(input\_tensor=relu\_9, k\_size=3, out\_dims=256, rate=16,  padding='SAME', use\_bias=False, name=' dia\_conv\_4')
29. relu\_10 = self.lrelu(dia\_conv4, name='relu\_10')
30. conv\_7 = self.conv2d(inputdata=relu\_10, out\_channel=256, kernel\_size=3,  padding='SAME', use\_bias=False, stride=1,name=' conv\_7')
31. relu\_11 = self.lrelu(inputdata=conv\_7, name='relu\_11')
32. conv\_8 = self.conv2d(inputdata=relu\_11, out\_channel=256, kernel\_size=3,  padding='SAME', use\_bias=False, stride=1,name=' conv\_8')
33. relu\_12 = self.lrelu(inputdata=conv\_8, name='relu\_12')
34. deconv\_1 = self.deconv2d(inputdata=relu\_12, out\_channel=128, kernel\_size=4,  stride=2, padding='SAME', use\_bias=False, name=' deconv\_1')
35. avg\_pool\_1 = self.avgpooling(inputdata=deconv\_1, kernel\_size=2, stride=1, padding='SAME',  name='avg\_pool\_1')
36. relu\_13 = self.lrelu(inputdata=avg\_pool\_1, name='relu\_13')
37. conv\_9 = self.conv2d(inputdata=tf.add(relu\_13, relu\_3), out\_chan nel=128, kernel\_size=3,padding='SAME', stride=1, use\_bias=False,name=' conv\_9')
38. relu\_14 = self.lrelu(inputdata=conv\_9, name='relu\_14')
39. deconv\_2 = self.deconv2d(inputdata=relu\_14, out\_channel=64, kernel\_size=4, stride=2, padding='SAME', use\_bias=False, name='deconv\_2')
40. avg\_pool\_2 = self.avgpooling(inputdata=deconv\_2, kernel\_size=2, stride=1, padding='SAME',    name='avg\_pool\_2')
41. relu\_15 = self.lrelu(inputdata=avg\_pool\_2, name='relu\_15')
42. conv\_10 = self.conv2d(inputdata=tf.add(relu\_15, relu\_1), out\_channel=32, kernel\_size=3, padding='SAME', stride=1, use\_bias=False,  name='conv\_10')
43. relu\_16 = self.lrelu(inputdata=conv\_10, name='relu\_16')
44. skip\_output\_1 = self.conv2d(inputdata=relu\_12, out\_channel=3, kernel\_size=3,  padding='SAME', stride=1, use\_bias=False,name='skip\_ouput\_1')
45. skip\_output\_2 = self.conv2d(inputdata=relu\_14, out\_channel=3, kernel\_size=3, padding='SAME', stride=1, use\_bias=False,name='skip\_ouput\_2')
46. skip\_output\_3 = self.conv2d(inputdata=relu\_16, out\_channel=3, kernel\_size=3, padding='SAME', stride=1, use\_bias=False,name='skip\_ouput\_3')
47. # 传统GAN输出层都使用tanh函数激活
48. skip\_output\_3 = tf.nn.tanh(skip\_output\_3, name='skip\_output\_3\_tanh')
50. ret = {
51. 'skip\_1': skip\_output\_1,
52. 'skip\_2': skip\_output\_2,
53. 'skip\_3': skip\_output\_3
54. }
55. **return** ret

## 判别器的实现：

1. with tf.variable\_scope(name, reuse=reuse):
2. conv\_stage\_1 = self.\_conv\_stage(input\_tensor=input\_tensor, k\_size=5,  stride=1, out\_dims=8,  group\_size=0, name='conv\_stage\_1')
3. conv\_stage\_2 = self.\_conv\_stage(input\_tensor=conv\_stage\_1, k\_size=5,  stride=1, out\_dims=16, group\_size=0, name='conv\_stage\_2')
4. conv\_stage\_3 = self.\_conv\_stage(input\_tensor=conv\_stage\_2, k\_size=5, stride=1, out\_dims=32, group\_size=0, name='conv\_stage\_3')
5. conv\_stage\_4 = self.\_conv\_stage(input\_tensor=conv\_stage\_3, k\_size=5, stride=1, out\_dims=64, group\_size=0, name='conv\_stage\_4')
6. conv\_stage\_5 = self.\_conv\_stage(input\_tensor=conv\_stage\_4, k\_size=5, stride=1, out\_dims=128, group\_size=0, name='conv\_stage\_5')
7. conv\_stage\_6 = self.\_conv\_stage(input\_tensor=conv\_stage\_5, k\_size=5,  stride=1, out\_dims=128, group\_size=0, name='conv\_stage\_6')
8. attention\_map = self.conv2d(inputdata=conv\_stage\_6, out\_channel=1, kernel\_size=5,  padding='SAME', stride=1, use\_bias=False, name='attention\_map')
9. conv\_stage\_7 = self.\_conv\_stage(input\_tensor=attention\_map \* conv\_stage\_6, k\_size=5,  stride=4, out\_dims=64, group\_size=0, name='conv\_stage\_7')
10. conv\_stage\_8 = self.\_conv\_stage(input\_tensor=conv\_stage\_7, k\_size=5,stride=4, out\_dims=64, group\_size=0, name='conv\_stage\_8')
11. conv\_stage\_9 = self.\_conv\_stage(input\_tensor=conv\_stage\_8, k\_size=5,  stride=4, out\_dims=32, group\_size=0, name='conv\_stage\_9')
12. fc\_1 = self.fullyconnect(inputdata=conv\_stage\_9, out\_dim=1024, use\_bias=False, name='fc\_1')
13. fc\_2 = self.fullyconnect(inputdata=fc\_1, out\_dim=1, use\_bias=False, name='fc\_2')
14. fc\_out = self.sigmoid(inputdata=fc\_2, name='fc\_out')
15. fc\_out = tf.where(tf.not\_equal(fc\_out, 1.0), fc\_out, fc\_out - 0.0000001)
16. fc\_out = tf.where(tf.not\_equal(fc\_out, 0.0), fc\_out, fc\_out + 0.0000001)
17. **return** fc\_out, attention\_map, fc\_2

## 想要了解更多代码请看github