Paper Reading

林昭炜 3170105728

Papers:

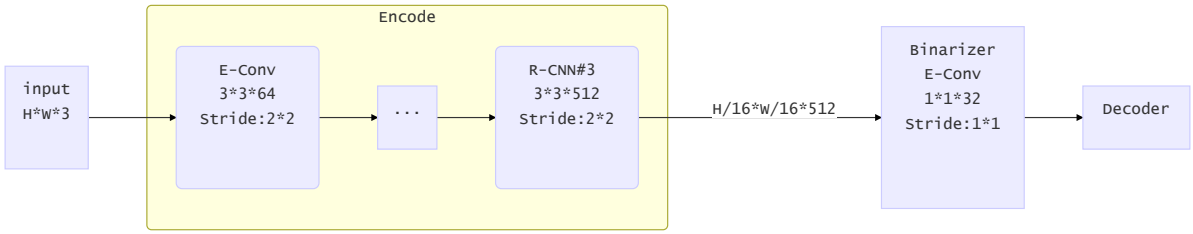
Full Resolution Image Compression with Recurrent Neural Networks [major]	**1608.05148.pdf**	**George Toderici et al.**
Variable Rate Image Compression With Recurrent Neural Networks	1511.06085.pdf	George Toderici et al.
Associative Long Short-Term Memory	1602.03032.pdf	Ivo Danihelka et al.

Intro

JPEG 是使用DCT/DWT，量化，熵编码最终实现对图片的压缩，它的压缩能力是超强的，很少有办法能超越JPEG，但是 George Toderici et al. 在他们15年的论文里提出JPEG 对缩略图的压缩能力比较弱，所以他们打算使用神经网络去压缩图片，图像被限制在32×32, 但从实验数据来看，在和JPEG对比也没占到什么便宜，所以他们在16年再接再厉，发表了一篇论文旨在解决全幅画面的压缩。

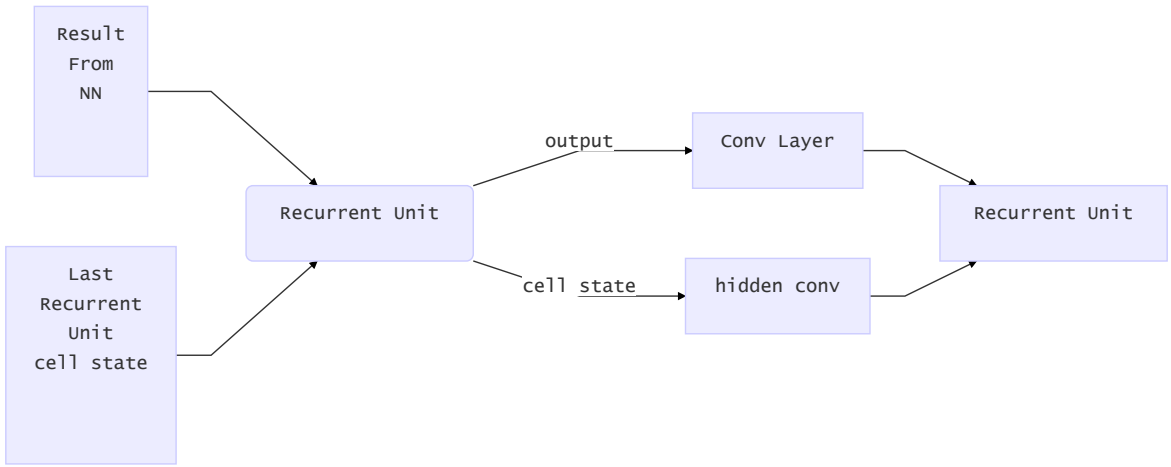
NN Architecture

下图是神经网络的简单示意：



图片经过几个R-CNN单元的Encoder, 高度和宽度变为原来的$1/16$, 因为加上一个Binarizer的作用，深度变成了32层，因为最后输出的其实是比特，所以原图大小变成 $(H/16) \times (W/16) \times 32 \times (1/8)$, 相当于原来图片的 $1/192$, 而一般的JPEG大约在$1/60$左右。

这是我对一个R-CNN单元的理解：



hidden convolution 层是对 Recurrent Unit 的卷积，用的是所谓的Hidden Kernel, 主要是保证输出的列数和卷积层是一致的，而行数不一致的话完全可以在 Recurrent Unit 内矩阵相乘的时候调整。当然，最省力的方式还是和图片的卷积核保持一致，这样子矩阵最后的维度一致。Recurrent Unit会在下章讲。

整个网络可以多次迭代，每次输出一串比特流，所以公式如下:

$$b_t = B(E_t(r_{t-1})), \quad \hat{x}_t = D_t(b_t) + \gamma \hat{x}_{t-1}$$
$$r_t = x - \hat{x}_t, \quad r_0 = x, \quad \hat{x}_0 = 0$$

E_t 是第t次迭代的Encoder, B是Binerizer, b_t 是最后压缩的结果, $\hat{x}_t$ 就是每次预测出来的图像。γ 取 0 或者 1， 取值的不同的情况会在 Reconstruction 里讲。利用Recurrent Unit 记忆的特性，每次只要告诉神经网络预测的结果和实际相差多少（残差 Residual ），再算出新的一堆比特加在后面，理论上解码器每次接受越多的，图像重建也越准确， 因为每次迭代的时候残差会减小。

Recurrent Unit

Recurrent Unit 是为了能保留每次传递的信息，让神经网络自己选择要记住那些东西。作者把市面上的三种常用的Recurrent Unit 都试了一遍。

LSTM

Long-Short Term Memory最早1997年提出，公式如下

$$[f, i, o, j]^T = [\sigma, \sigma, \sigma, tanh]^T (W x_t + U h_{t-1} + b),$$
$$c_t = f \odot c_{t-1} + i \odot j,$$
$$h_t = o \odot tanh(c_t).$$

f, i, o 分别是forget gate, input gate, output gate, j 是这次生成的记忆信息。

c_t 就是更新后的LSTM 的状态，它记住多少之前的状态由forget gate 决定，它保存多少当前的记忆信息由 input gate 决定， 而最终的输出由 output gate 决定。σ就是sigmoid 函数，$\sigma, \tanh$的作用都是element wise的。这里的W, U 是需要两个训练的矩阵，作者号称是对输入做convolutional linear transformation，但我不清楚具体是什么，仅仅知道需要 Toeplitz matrice 取做变换。我认为f, i, o, j 应该是有自己对应的W, U矩阵，而作者似乎是用同一个矩阵对其进行变换, 感觉这和正常的LSTM出入有点大。这样子的话三种Gate根本区分不出来。

之前提到矩阵维度要一样，否则两个矩阵无法相加。

Associate LSTM

这是由Ivo Danihelka et al. 提出的，想法很简单，就是把 LSTM 输出的结果自己和自己上下堆在一起放进新的矩阵里，产生冗余信息的效果。bnd是把它堆起来之后，对所有超过1的元素投影到0到1区间内，这样子它就相当于一个激活函数了:

$$
\begin{gathered}
[f, i, o, j, r_i, r_o]^T = [\sigma, \sigma, \sigma, bnd, bnd, bdn]^T (Wx_t + Uh_{t-1} + b), \\
c_t = f \odot c_{t-1} + r_i \odot i \odot j, \\
h_t = o \odot bnd(r_0 \odot c_t), \\
h_t = (\mathrm{Re}h_t, \mathrm{Im}h_t).
\end{gathered}
$$

GRU

最简单的GRU单元:

$$
\begin{gathered}
z_t = \sigma(W_z x_t + U_z h_{t-1}), \\
r_t = \sigma(W_r x_t + U_r h_{t-1}), \\
h_t = (1 - z_t) \odot h_{t-1} + z_t \odot tanh(Wx_t + U(r_t \odot h_{t-1})) + \alpha_h W_h h_{t-1}, \\
h_t^o = h_t + \alpha_x W_{ox} x_t.
\end{gathered}
$$

它讲LSTM单元Input 和 forget gate 简化成 $z_t, (1 - z_t)$, 而r_t是一个relevance矩阵。作者认为GRU同时记录了空间和每次迭代的信息，所以在后面加了一个$\alpha_h W_h h_{t-1}$参数。

Reconstruction Framework

回忆一下迭代公式:

$$
\begin{gathered}
b_t = B(E_t(r_{t-1})), \quad \hat{x}_t = D_t(b_t) + \gamma \hat{x}_{t-1} \\
r_t = x - \hat{x}_t, \quad r_0 = x, \quad \hat{x}_0 = 0
\end{gathered}
$$

One-Shot

这里γ 取0，每次传入的是和上一次图片的残差，依靠自己的记忆每次都是去重建完整的图像。

Additive

γ 取1， 重建出来的是仅仅是残差，最后结果是几次迭代之和。

Residual Scaling

我们希望每一次迭代残差能快速接近0，这样子我们很快能逼近原图。论文作者觉得对于不同图片残差下降的速度不同，有些可能很慢，所以既然山不过来，我到山那边去，干脆用直接去缩放这个残差:

$$
\begin{gathered}
g_t = G(\hat{x}_t), \quad b_t = B(E_t(r_t - 1) \odot ZOH(g_{t-1})) \\
\hat{r}_{t-1} = D_t(b_t) \oslash ZOH(g_{t-1}), \\
\hat{x}_t = \hat{x}_{t-1} + \hat{r}_{t-1}, \quad r_t = x - \hat{x}_t, \\
g_0 = 1, \quad r_0 = x
\end{gathered}
$$

ZOH 作者说是spatial upsampling by zero-order hold，我不是很清楚，总之是以一种方式把g_t扩大成一个矩阵, 这样子在编码的时候减小残差，$G()$是由4层3×3卷积+ELU+一层2×2的卷积层+ELU， ELU是激活函数:

$$ELU(x) = \begin{cases} e^x - 1, & x < 0 \\ x, & x > 0 \end{cases}$$

最后加上2使其变为$(1, +\infty)$

感想

优点

最让我佩服这一篇论文的是作者能把CNN和RNN结合起来，普通的CNN只能输出和原始图像成比例的数据，加上全连接层之后又只能输出固定长度的数据，CNN与RNN结合之后可以根据需求决定迭代次数，从而按需生成不同质量图片。

缺点

首先是人们可能对神经网络的理解不深入，多数靠trial and error来测试不同架构的性能，正如本文作者用来不同的RNN单元去测试。我个人认为在没有真正理解神经网络之前，或者有数学证明，我们也无法确定哪一种是最优化的方案。

在Matthew D. Zeiler et al. 的Visualizing and Understanding Convolutional Networks指出CNN会关注图片的特征，往往越深层的网络会提取高阶的特征，比如前几层可能关注线条，后面几层会看到比如头部等特征，但是压缩图片的话意味着高阶的特征可能无穷无尽，所以只能用比较浅的网络。而且可能输出的图片质量不稳定，正如作者所说，有些图片的可能残差很难下降，所以它对图片的选择性比较强。

展望

神经网络的压缩图片的思路和传统算法完全不一样。CNN 会提取图片的特征，也许在之后的能训练网络去关注有用的特征，就是图片里我们想要看到细节的地方。除此之外，如果真的广泛应用这种技术，我觉得往往是那些可能需要存档的照片，特别是服务端，这些照片往往不需要频繁读取，但是必须得存着以后有一天来看，这样子的话会对压缩率的要求比较高。还有一种应用场景可能是有大量图的网站，加载图片很耗时，这种技术可以加快图片的加载。

我认为这样的格式在文件头里会存一些参数，比如固定所有的参数只训练一层的参数记录在文件里，解码的时候根据这个参数，加上图片格式规定的固定参数去解码。这样子可以针对每一张图片进行优化。

这种技术对不同情况网络也很友好，因为每次传递数据都会使图片质量更高，图片可以像视频一样随意转换清晰度，通过叠加层来增加清晰度。

除此之外，这项技术还可以应用到视频里，CNN不停地接受每一帧的数据，利用LSTM记录上一帧的信息然后去提取特征，好处是不需要人工去找一个匹配的像素点，而且LSTM可以记住之前好几帧的内容，根据需求选择和那一帧去匹配，加大压缩率，让神经网络去发现匹配的区域。每一层有一个CUTTOFF的卷积，生成一个预测值，用来预测是否发生了场景变换，如果变了那么清空记忆单元重新去根据新的一帧提取特征。也可以有一个Global LSTM，由几个LSTM组成，实现几个类似场景帧之间的匹配，进一步压缩空间。但是最大问题在于算力，视频一般使30fps, 为了真实感有达到60fps，意味着每秒要计算出60张图片，肯定需要GPU去加速。这种超高压缩率的技术未来应该用在4k视频流等，家庭影院甚至电影院都会受益于这个技术。

这项技术也可以用于图像加密，对于特别是加密要求特别高的应用环境下，神经网络的参数就是解密图片的密钥，而这些密钥为了完全可以用Probabilistic Encryption[1] 去加密，因为这个加密方式会让图片至少扩大128倍，但是如果只加密参数的话，数据量就小很多了，这也要求对神经网络进行改进，比如将conditional number[2] 加入Cost Function, 让神经网络尽可能扩大它，从而保证即便攻击者找到一个和实际参数比较接近的估算值，也无法还原信息。

画完大饼以后，我觉得这项技术局限也挺大的，如果要作为一个通用的格式，光是超参数的确定也会有很大的争议，很难达成一个统一的标准，而且这种技术需要大量的算力，我在MATTLAB上对一个4583×3055 的图片做一个rgb2yuv转换都要花几秒的时间，更别说需要层层卷积的神经网络了。在小图片领域，JPEG虽然做得不是特别好，但是神经网络也无法有太大的提高，用户层面一定没有太大动力换成新的格式。

大图片速度慢，小图片提高不明显，是神经网络在用于压缩图片上的局限。

---

1. S. Goldwasser, S. Micali, Probabilistic Encryption, Journal of Computerand System Sciences, 28, pp. 270–299, 1984 ↩

2. https://en.wikipedia.org/wiki/Condition_number↩