

- 1. Zero Shot Super Revolution 翻译
  - Abstrac
  - 1. Introduction
  - 2. The Power of Internal Image Statistics
  - 3. Image-Specific CNN
    - 3.1. Architecture & Optimization
    - 3.2. Adapting to the Test Image
  - 4. Experiments & Results
    - 4.1. The ‘Ideal’ Case
    - 4.2. The ‘Non-ideal’ Case
  - 5. Conclusion
- 2. 论文复现
  - 2.1 神经网络结构

# 1. Zero Shot Super Revolution 翻译

## Abstrac

作者在引言中阐述了，虽然深度学习在超分辨率重建方面表现的非常好，但是这些有监督的方法需要大量的无瑕疵的，与原高分辨率图像对应的低分辨率图像。然而最近的许多先进的超分辨率算法在严格意义上并没有达到这一点。在这篇论文中，作者介绍了他们的这种 Zero Shot（零次学习）的方法，他们利用图像内部重复出现的信息，在测试阶段使用从单个图像本身抽取的信息作为样本，训练模型。用这个方法，能够处理类似于老照片，生物信息，噪点图像这样来源未知或质量不理想的图像。对于这类图像的处理，我们的方法表现的比最先进的 CNN 超分辨率方法和之前的无监督学习超分辨率方法都要好。

## 1. Introduction

基于深度学习的超分辨率重建方法在性能上得到了极大提升，近期最先进的方法比之前的非深度学习超分辨率（包括有监督和无监督的）多了几个 dB。性能的巨大提升是通过深入和精心设计 CNN 而获得的，这些 CNN 使用外部数据集进行长时间（几天或几周）彻底训练得到的。然而这些外部有监督方法的极佳性能来源于符合训练条件的数据集，如果数据集不满足训练条件，那么他们的表现就非常糟糕了。比如基于 CNN 的 超分辨率重建算法就是基于大量高质量图像训练而成的，这些图像的低分辨率图像是通过预先定义的 downscaling kernel （通常使用 bicubic kernel 加抗锯齿 —— MATLAB 默认的 imresize 命令）以及预定义的 SR scaling-factor （假定两个维度均相同的情况下，通常是  $\times 2$ ,  $\times 3$  或者  $\times 4$ ）无瑕疵（例如传感器噪声、不理想的 PSF、图像压缩等等）处理得到的。图例 2 展示了在非理想 downscaling kernel 处理，或包含锯齿，或包含锯齿，或包含传感器噪声，或包含压缩瑕疵的情况下，CNN SR 和 ZSSR 模型的表现，如下图

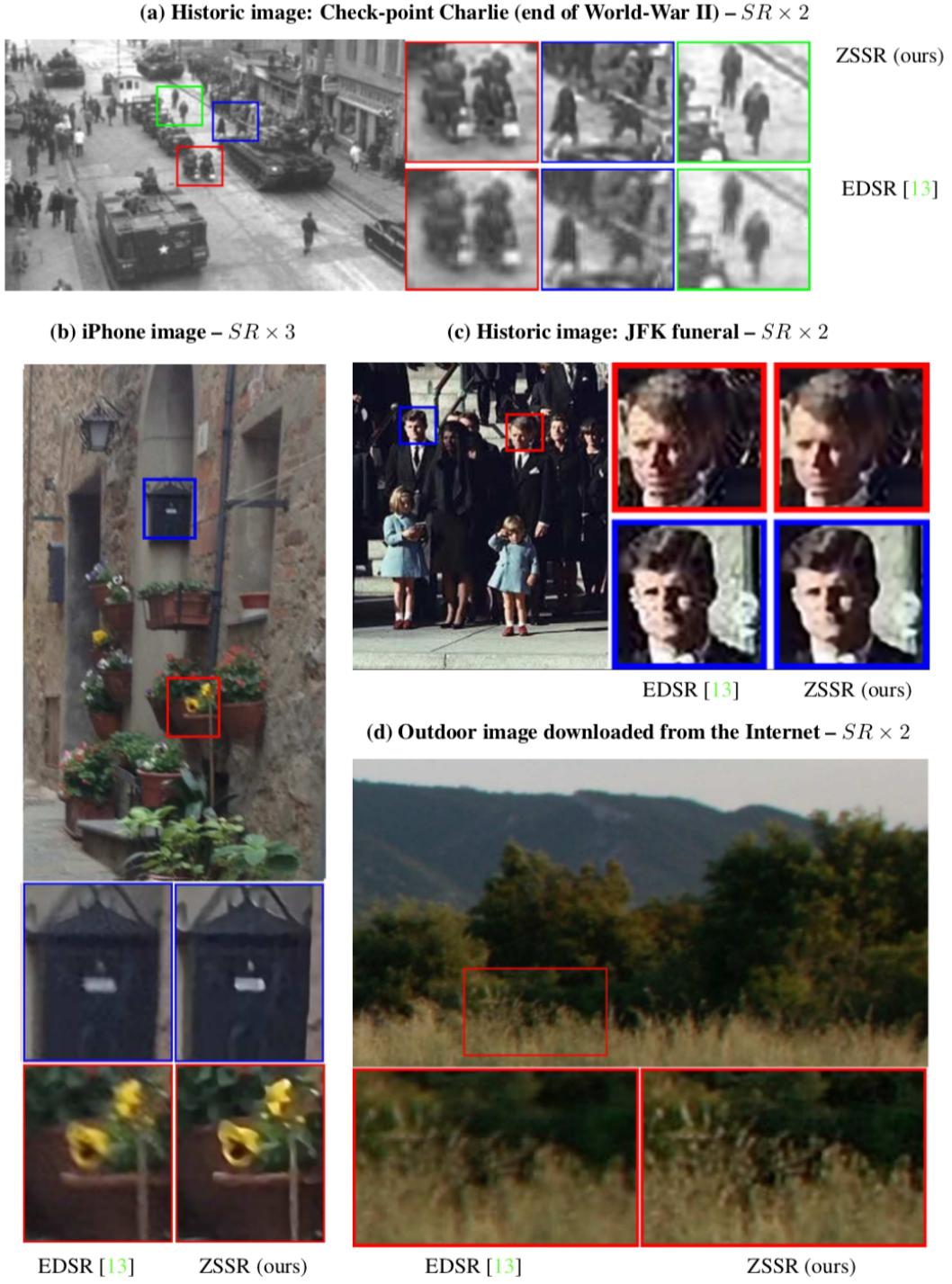
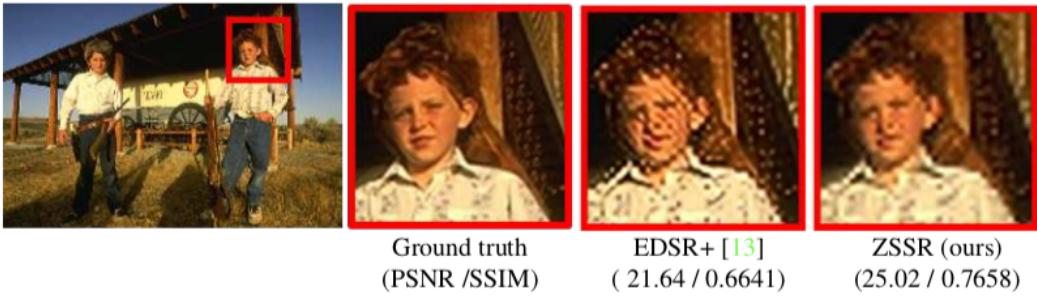


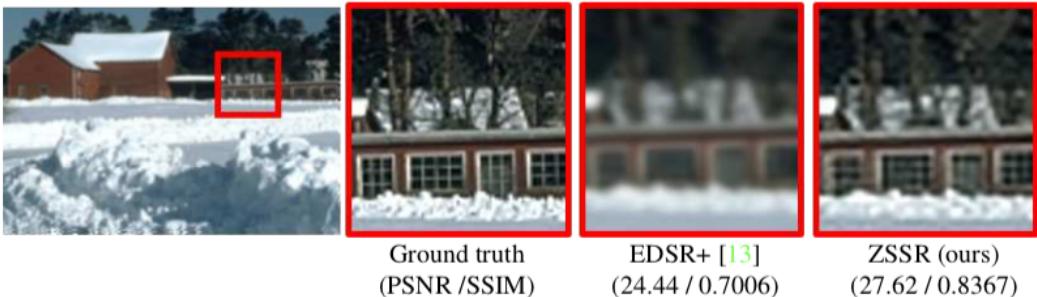
Figure 1: **SR of real images (unknown LR acquisition process).** Real-world images rarely obey the ‘ideal conditions’ assumed by supervised SR methods. For example, old historic photos (a,c), images taken by smartphones (b), random images on the Internet (d), etc. Since ZSSR trains at test time on examples extracted from the test image, it is better at performing SR ‘In-the-Wild’ (i.e., in unconstrained and unknown settings). Full sized images can be found on our [project website](#).

图例 1 进一步表明上述效果不是人为干预的，而是在处理现实中的低分辨率图像上经常发生的，这些图像的来源有网络、iPhone 和历史图像。在这种非理想的情况下，先进的 SR 方法得到的结果并不好。

(a) SR under aliasing:



(b) SR under unknown *non-ideal* downscaling kernel:



单个图像在各个尺度上重复出现的小图像信息是自然图像的强大属性。它构成了例如无监督 SR, Blind-SR (downscaling kernel 未知) , Blind-Deblurring (blind 去模糊) , Blind-Dehazing (Blind 去雾) 等等这些无监督图像增强方法。尽管这些无监督方法不想上述有监督方法那样受到限制, 但它们需要使用预定义尺寸 (通常为  $5 \times 5$ ) 的小图像 patches 的欧几里得相似性, 使用 K-nearest-neighbours 搜索。因此, 这些方法不能推广到 LR (low resolution) 中不存在的 patches, 也不能推广到隐式学习相似度衡量, 也不能够适应图像内部重复结构的不均匀尺寸。我们的 image-specific CNN 利用了 image-specific 的跨尺度内部信息重现的特点 (原文是 power, 这里不太清楚翻译成什么好) , 该方法不受限于限制 patch-based 方法的条件。我们用 LR 图像及其 downscaling 版本 (自我监督) 训练了一个 CNN 用来推断从 HR 到 LR 的关系, 接着我们将这些训练过的 relations 应用到输入的 LR 图像来生成 HR 图像。这个方法大幅度的胜过了无监督的 patch-based SR 模型。

由于单个图像内部的视觉熵比一般的外部图像集合要小得多, 因此小型的简单 CNN 能够满足这个任务。因此即使我们的神经网络是在测试阶段训练的, 其运行时间也可以与最先进的有监督 CNN 在测试阶段的训练时间相媲美。有趣的是, 在使用最先进的监督方法使用的基准数据集测试我们的 image-specific CNN 时, 产生了令人印象深刻的结果 (尽管不是最优秀的结果, 我们的模型也较小且未经训练) , 并且在非理想图像上的处理结果大大超过了有监督的最先进的 SR。我们提供上述实验在视觉和经验上的证据。

文中的“Zero-Shot”属于来源于识别/分类领域。然而这里需要注意的是, 我们的方法不像 Zero-Shot 学习或者 One-Shot, 我们的方法不需要任何辅助信息或是额外的图像。我们手头可能仅仅拥有一张测试图像。除此之外, 当提供了可用的额外信息 (例如, 使用 Nonparametric blind super-resolution 可以直接从测试集中估算 downscale kernel) , 我们的 image-specific CNN 能够充分利用测试时间, 极大提高处理效果。本论文在以下几个方面做出贡献:

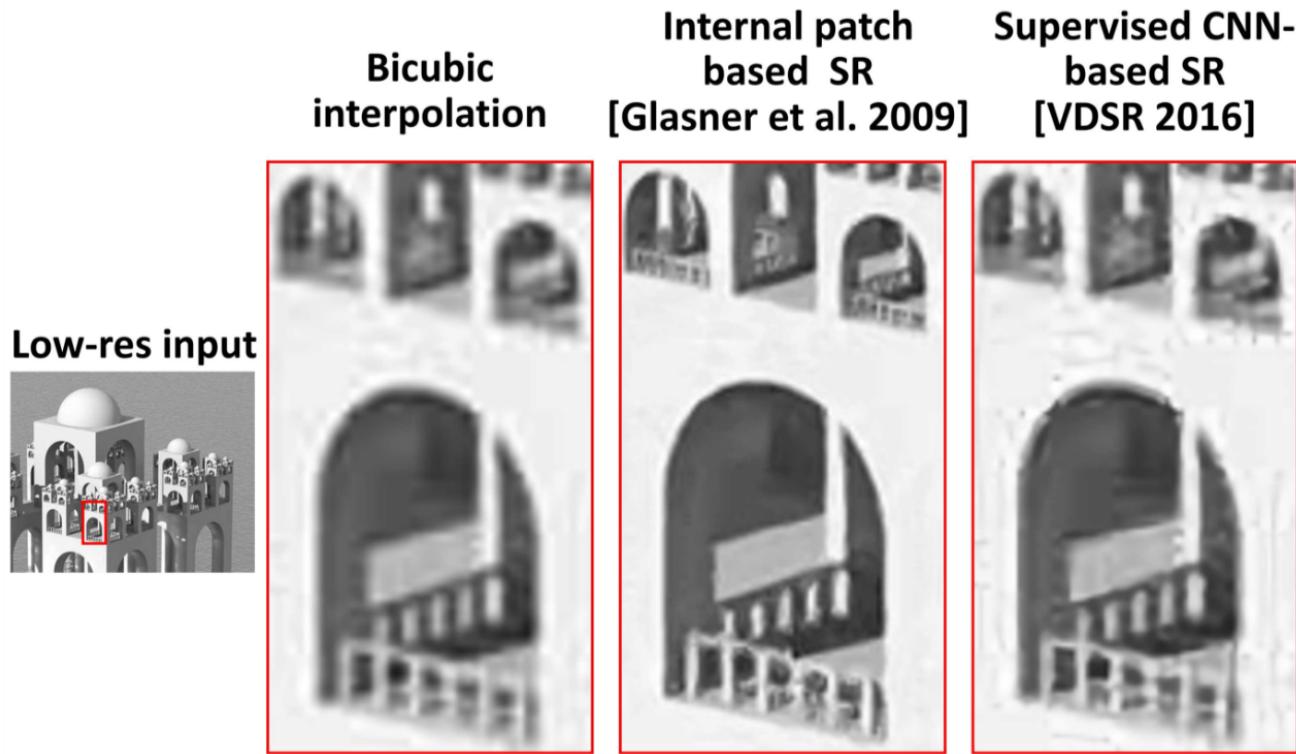
1. 这是第一个无监督的 CNN-based SR 方法

2. 该模型能够处理非理想成像条件以及广泛的图像和数据类型（即使是第一次处理）
3. 不需要与处理，占用的计算资源较少
4. 能够应用于任何大小的 SR，理论上具有任何长宽比
5. 能够适应于已知和未知的成像条件（测试阶段）
6. 对于非理想的条件，它能够提供最先进的结果，对于理想的条件，与有监督且训练好的最先进模型也有的一拼

## 2. The Power of Internal Image Statistics

我们这个方法的基础是自然图像内部的数据重复性很高。比如，单个图像内部的小图像 patches (例如 $5 \times 5$ ,  $7 \times 7$ ) 重复了很多次。《Super-resolution from a single image》和《Internal statistics of a single natural image》两篇论文使用了数百个自然图像对上述结论进行了验证，几乎任何自然图像的小 patches 都是正确的。

图例 3 展示了《Super-resolution from a single image》中基于内部 patch 重现的简单单个图像 SR。



注意，它能够恢复图中小阳台的小扶手，因为在较大阳台之一图像中的其他位置发现了它们存在的证据。事实上这些小扶手存在的唯一证据在图像中不通缩放尺度的不同位置。这在任何外部数据集上是找不到的，无论这个数据集有多大。因为能够观察到，最先进的 SR 方法在依赖于外部数据库时不能够恢复 image-specific 信息。虽然在这里使用了类分形图像举例说明了强大的内部预测能力，但在任何自然图像中都能分析并表现出强大的内部预测能力。

有证据表明，单个图像，其 patches 的内部熵比一般自然图像合集中 patches 的外部熵要小。这进

一步引起了这样的观测：内部图像数据能够提供比从通用图像集合中获取的外部统计数据更强的预测能力。在不确定性增强以及图像质量下降的条件下，上述情况尤为明显。

### 3. Image-Specific CNN

我们的 image-specific CNN 结合了内部 image-specific 信息的预测能力和低熵以及神经网络的泛化能力。给定一个测试图像  $I$ ，在没有任何外部可用样本的情况下训练，据此我们构建了一个 image-specific CNN，旨在解决特定图像的 SR 任务。我们从测试图像本身抽取样本来训练我们的 CNN。我们是通过对 downscale LR 图像  $I$  来获取上述样本，从而获得其本身分辨率更低的 version,  $I \downarrow s$  (图例 4b 的上半部分) 这里的  $s$  表示需要的 SR 比例因子。接着，我们用训练好的 CNN 去测试图像  $I$ ，现在使用  $I$  作为 LR 输入到神经网络中，用来构建 HR 输出  $I \uparrow s$  (图例 4b 的下半部分)。

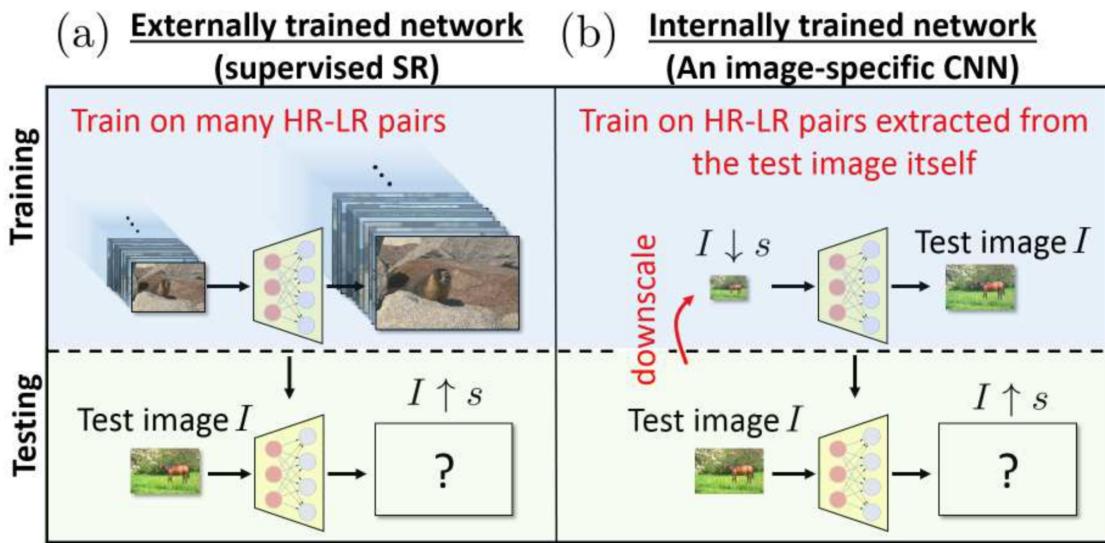


Figure 4: **Image-Specific CNN – “Zero-Shot” SR.** (a) *Externally-supervised CNNs are pre-trained on large external databases of images. The resulting very deep network is then applied to the test image  $I$ .* (b) *Our proposed method (ZSSR): a small image-specific CNN is trained on examples extracted internally, from the test image itself. It learns how to recover the test image  $I$  from its coarser resolutions. The resulting self-supervised CNN is then applied to the LR image  $I$  to produce its HR output.*

请注意训练好的 CNN 是完全卷积化的，因此适用于不同尺寸的图像。

由于我们的训练集只由一个样本构成，我们在输入图像  $I$  上进行数据扩展来提取更多的 LR-HR 样本对儿来训练神经网络。这些扩展的数据通过 downscale 输入图像  $I$  到许多分辨率更小的 versions ( $I = I_0, I_1, \dots, I_n$ )。它们起到 HR supervision 的作用，被称作 “HR father”。每一个 HR father 都使用所需的 SR 缩放因子被 downsampled，来获得 “LR son”，这些数据构成了训练数据集。这些最后得到的训练集是由许多 image-specific LR-HR 样本对构成的，神经网络可以随机训练这些对儿。

我们还通过旋转每一个 LR-HR 对儿到四个角度 ( $0^\circ, 90^\circ, 180^\circ, 270^\circ$ )，旋转后的再分别进行垂直和水平镜像。这样就得到了原先数据集 8 倍的数据量。

由于缺少鲁棒性和即使 LR 图像很小也能允许使用较大的 SR 缩放因子的缘故，SR 执行缓慢。我们的算法适用于几个中间的缩放因子 ( $s_1, s_2, \dots, s_m = s$ )。在每个中间尺度  $s_i$  上，我们将利用这个尺度  $s_i$  生成的 SR 图像和 HR 图像及其 downscale/旋转 version (这里我个人理解为一个尺度生成一组训

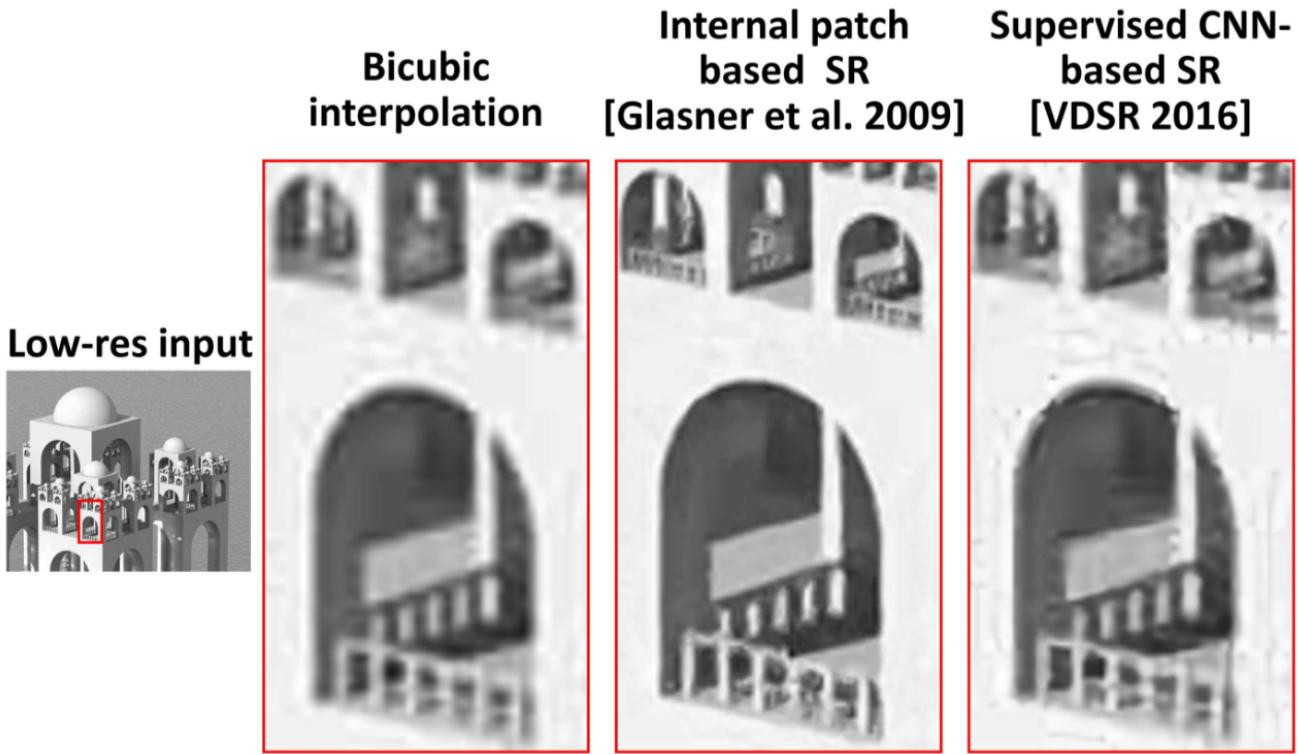
练集) 作为新的 HR father 添加到我们逐渐扩充的训练集中。我们使用下一个缩放因子  $s_{i+1}$  downscale 这些数据 (以及之前较小的“HR 样本”) 并得到一个新的 LR-HR 训练样本对。重复这个步骤直到分辨率到达  $s$ 。

### 3.1. Architecture & Optimization

通过大量不同的 LR-HR 外部图像样本集来训练的有监督的 CNN 必须在其学习过的权重中找到所有可能的 LR-HR 关系中的多样性。由于这个原因，这些网络变得非常深，并且非常复杂。相对的，单张图像的 LR-HR 关系就非常的小，因此能够通过更小更简单的 image-specific 网络进行硬编码。

我们使用一个含有 8 层的简单全卷积网络，每一层含有 64 个通道 (channels)。每一层的激活函数是 ReLU。网络输入被插值到输出尺寸。正如之前基于 CNN 的 SR 方法 (《Deeply-recursive convolutional network for image super-resolution》、《Accurate image super-resolution using very deep convolutional networks》以及《Learning a deep convolutional network for image super-resolution》) 所做的那样，我们仅仅学习被插值的 LR 和它的 HR parent 之间的残差。我们使用  $L_1$  损失函数和 ADAM 优化器，以及值为 0.001 的学习率。我们周期性的对重构误差使用线性拟合，如果标准差比线性拟合的斜率大了一个 factor，我们就将学习率除以 10。当学习率到达  $10^{-6}$  时就停下来。

注意，尽管可视范围有限，ZSSR 也有能力捕获测试图像内部信息的非本地重现。比如，当 ZSSR 被应用于图例 3 的 LR 图像时，它训练了一个CNN，在其他扶手没有出现在可视范围内时，将较低分辨率 version 中的扶手恢复到 LR 测试图像中的扶手。



当这个 CNN 被应用于测试图像本身时，由于使用了相同的 image-specific 过滤器，它能将恢复其他地方的新扶手。

为了加速训练阶段，使得运行时间与测试图像  $I$  尺寸无关，我们在每次迭代中对随机选取的 father-

*son* 样本对进行一次随机的固定大小裁剪 (crop)。裁剪尺寸通常为  $128 \times 128$  (除非被采样的图像对比较小)。在每一次迭代中, 采样一个 LR-HR 样本对的可能性被设置为非均匀的并且与 HR father 的尺寸成比例。尺寸比例 (HR father 和 测试图像  $I$  的比例) 越接近 1, 被采样的可能性就越大。这反映了非合成的 HR 样本比合成的 HR 样本具有更高的可信度。

最后, 我们使用了类似于《Enhanced deep residual networks for single image super-resolution》中的几何自集成的方法 (将测试图像  $I$  通过 8 种旋转+翻转生成 8 个不同的输出, 然后组合他们)。我们取这 8 个输出的中位数而不是他们的均值。我们将它与《Improving resolution by image registration》和《Super-resolution from a single image》中的反投影技术结合, 这导致 8 个输出图像要经过几次反投影迭代并且最终中位数图像也会通过反投影进行矫正。

运行时间: 虽然在测试阶段就完成了训练, 但是  $\text{SR} \times 2$  的运行时间在 Tesla V100 GPU 上只有 9s, 在 K-80 上只有 54s (BSD100 上取平均数据)。这个运行时间几乎独立于图像尺寸或者相对于 SR 缩放因子 独立 (这是由于在训练集中同等尺寸的 crop)。最终的测试运行时间在训练迭代方面是微不足道的。

对于理想的情况, 我们逐步增加分辨率。例如, 使用 6 个比例因子逐渐增加通常会在 PSNR 上提高约 0.2 dB, 但是每个图像会增加约 1 分钟 (在 V100 上)。这里就有一个运行时间和输出质量的取舍, 由用户自己来决定。

为了进行比较, leading EDGE[《Enhanced deep residual networks for single image super-resolution》] 的测试时间和图像大小成平方增长。虽然该方法在小尺寸图像上很快, 但对  $800 \times 800$  的图像上, 五次执行时间都要比我们这个训练+测试模型慢 (或者与使用 6 个中间缩放因子逐渐增加的方法所花费的时间相当)。

### 3.2. Adapting to the Test Image

当从 HR 得到的 LR 图像的获取参数对所有图像固定之后 (例如, 同样的 downscaling kernel, 高质量图像条件), 当前的有监督 SR 方法的表现令人难以置信。然而在实际过程中, 由于相机/传感器不同个人成像条件的原因 (例如, 拍摄照片时相机会发生轻微的非自愿晃动, 能见度差的情况等), 获取的过程往往随着图像的变化而变化 (例如, 不同的镜头类型和 PSF)。这会导致不同的 downscaling kernels, 不同的噪声特征以及多种压缩 artifacts 等等。人们几乎不能训练所有可能的图像采集配置/设置。此外, 对于所有可能的 degradations/settings 类型, 单监督的 CNN 不太可能表现的很好。为了获取好的性能, 人们需要许多不同的特定监督的 SR 网络, 每一个都在不同类型的 degradations/settings 上训练了几天或几周。

这就是 image-specific 网络的优势所在。我们的神经网络在测试阶段, 能够适应手头测试图像的特定 degradations/settings。我们的神经网络能够在测试时接受来自用户输入的以下参数:

1. 所需要的 downscaling kernel (若不提供 kernel, 则将 bicubic kernel 作为默认选择)
2. 所需要的 SR 缩放因子  $s$
3. 所需要的 gradual scale increases 数量 (在速度和质量之间进行权衡)
4. 是否在 LR 和 HR 图像之间进行反投影增强 (默认选择 是)
5. 是否需要为每一个从测试图像中提取出的 LR-HR 样本对儿中的 LR sons 增加“噪声” (默认选择 否)

最后两个参数（取消反投影和增加噪声）允许处理低质量 LR 图像的 SR（无论噪声来源于传感器噪声，还是 JPEG 压缩 artifacts，或其他噪声）。我们发现，增加少量的高斯噪声（均值为 0 并且标准差在  $\sim 5$  灰阶）能够提升多种 degradations 的性能（高斯噪声，斑点噪声，JPEG artifacts 以及其他更多 degradations）。我们将这种现象归结于这样一个事实，image-specific 信息倾向于在各个 scales 上重复，但是噪声 artifacts 则不会。为 LR sons 添加小部分合成噪声（但不要给 HR fathers 添加）能让网络学会忽略不相关的跨尺度（cross-scale）信息（噪声），与此同时使网络学会提高相关信息（信号细节）的分辨率。

的确，我们的实验展示了对于低质量 LR 图像以及各种 degradation 类型来说，image-specific CNN 比最先进的 EDSR+ 在 SR 上能够取得更好的结果（详见第四节）。相似的，在非理想 downscaling kernels 的情况下，image-specific CNN 相对于最先进的方法有了显著提升（甚至没有任何噪声）。当 downscaling kernel 已知的情况下（例如，已知 PSF 的传感器），可以将其提供给我们的网络。当 downscaling 未知的情况下（这很常见），对于 kernel 的粗糙估计能够直接通过测试图像本身被计算出来。这样的 rough kernel 估计足以在非理想 kernel 上获得比 EDSR+ 多 1dB 的提升（参阅 Figs.1 和 2 以及第四节的经验评估）。

注意，在测试阶段为外部最先进的有监督 SR 方法提供被估计的 downscaling kernel 没有用处。它们需要用这个特定的（非参数化）downscaling kernel，在新的 LR-HR 集合对儿中完全重新训练一个新的神经网络。

## 4. Experiments & Results

我们的方法（ZSSR - “Zero-Shot SR”）主要面向通过现实的（未知和变化）采集设置的真实 LR 图像。现实的 LR 图像没有 HR *ground truth*，因此需要通过视觉评估（如 Fig. 1 所示）。为了定量地评估 ZSSR 的性能，我们在多种设置上进行了几个受控实验。有趣的是，在使用先进的有监督方法训练和专门化的理想基准数据集时，ZSSR 产生了具有竞争力的结果（although not SotA，尽管我们的 CNN 比较小，并且没有经过预训练）。然而，在非理想数据集下，ZSSR 大大超过了最先进的 SR。报告中所有的数值结果都是使用了《Accurate image super-resolution using very deep convolutional networks》和《Deeply-recursive convolutional network for image super-resolution》的评估脚本所得出的。

### 4.1. The ‘Ideal’ Case

虽然理想情况并不是 ZSSR 的目标，我们仍旧在理想 LR 图像的标准 SR 基准中测试了我们的模型。在这些基准中，通过使用 MATLAB 的 imresize 命令（一个使用 抗锯齿 downsample 的 bicubic kernel），对 HR versions 进行 downscale 之后理想的得到了 LR image。Table 1 表明，在与根据这些条件被彻底训练的外部有监督方法相比，我们的 image-specific ZSSR 获得了具有竞争力的结果。

		Supervised				Unsupervised	
Dataset	Scale	SRCNN [4]	VDSR [9]	EDSR+ [13]	SRGAN [12]	SelfExSR [7]	ZSSR (ours)
Set5	$\times 2$	36.66 / 0.9542	37.53 / 0.9587	38.20 / 0.9606	-	36.49 / 0.9537	37.37 / 0.9570
	$\times 3$	32.75 / 0.9090	33.66 / 0.9213	34.76 / 0.9290	-	32.58 / 0.9093	33.42 / 0.9188
	$\times 4$	30.48 / 0.8628	31.35 / 0.8838	32.62 / 0.8984	29.40 / 0.8472	30.31 / 0.8619	31.13 / 0.8796
Set14	$\times 2$	32.42 / 0.9063	33.03 / 0.9124	34.02 / 0.9204	-	32.22 / 0.9034	33.00 / 0.9108
	$\times 3$	29.28 / 0.8209	29.77 / 0.8314	30.66 / 0.8481	-	29.16 / 0.8196	29.80 / 0.8304
	$\times 4$	27.49 / 0.7503	28.01 / 0.7674	28.94 / 0.7901	26.02 / 0.7397	27.40 / 0.7518	28.01 / 0.7651
BSD100	$\times 2$	31.36 / 0.8879	31.90 / 0.8960	32.37 / 0.9018	-	31.18 / 0.8855	31.65 / 0.8920
	$\times 3$	28.41 / 0.7863	28.82 / 0.7976	29.32 / 0.8104	-	28.29 / 0.7840	28.67 / 0.7945
	$\times 4$	26.90 / 0.7101	27.29 / 0.7251	27.79 / 0.7437	25.16 / 0.6688	26.84 / 0.7106	27.12 / 0.7211

Table 1: Comparison of SR results for the ‘ideal’ case (bicubic downscaling).

事实上，ZSSR 显然比老旧的 SRCNN 表现要好，在一些样本中的表现与 VDSR（在去年之前一直是最先进的方法）相比，有的一拼或者更胜一筹。在无监督 SR 情况下（regime），ZSSR 的表现比 leading method SelfExSR 要好很多。

此外，在图像带有很强的内部重复性结构的情况下，即使这些 LR 图像是使用理想的有监督设置生成的，ZSSR 也有胜过 VDSR 的趋势，并且有时候与 EDSR+ 相比，也有这种趋势。上述的一个案例在 Fig. 5 中有所表现。

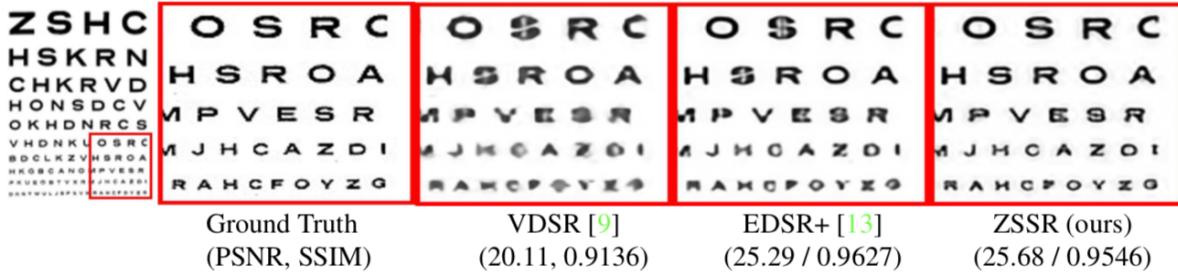


Figure 5: In images with strong internal repetitive structures, ZSSR tends to surpass VDSR, and sometimes also EDSR+, even though the LR image was generated using the ‘ideal’ supervised setting (i.e., bicubic downscaling).

虽然这个图像并不是典型的自然图像，但是进一步的分析表明，Fig.5 所展示的内部学习性能（通过 ZSSR），不仅存在于“分形”图像中，也存在于普通自然图像。Fig. 6 也展示了蕾丝的例子。



Figure 6: **Internal vs. External preference.** *Green: pixels that favor Internal-SR (i.e., pixels where ZSSR obtains lower error with respect to the ground-truth HR image); Red: pixels that favour External-SR (EDSR+).*

正如上图所示，图像中的一些像素（被标记为绿色的）能够利用被内部学习的数据重现（ZSSR）中获益更多，而被深度学习的外部信息则做不到这一点，而其他（用红色标记的）像素则能从被学习的外部数据中获益更多（EDSR+）。正如所期望的那样，内部方法（ZSSR）在信息高度重现的图像区域中是有很大优势的，尤其是那些特征非常小的区域（有着极低分辨率），像是楼房上的小窗户。如此小的特征可以在同样一张图像（的不同位置和尺度）中的其他地方找到更大的（高分辨率）的相同特征。这暗示着，在一个可计算的框架内，通过结合内部学习和外部学习的能力，可能存在进一步提升 SR 的潜力（即使是理想的 bicubic 案例）。这也为我们未来打算做的工作。

#### 4.2. The ‘Non-ideal’ Case

现实中的图像往往不能够被理想化地生成。我们对以下非理想化情况进行了实验：(i) 非理想 downscaling kernels（偏离了 bicubic kernel）以及 (ii) 低分辨率 LR images（例如，由于噪点，压缩 artifacts 等等原因导致的低分辨率），并获得了结果。在非理想条件下，image-specific ZSSR 提供了比最先进的 SR 方法更好的结果（好了 1-2 个dB）。这些大量的实验会在之后进行描述。Fig.2 展示了这样的可视化结果。

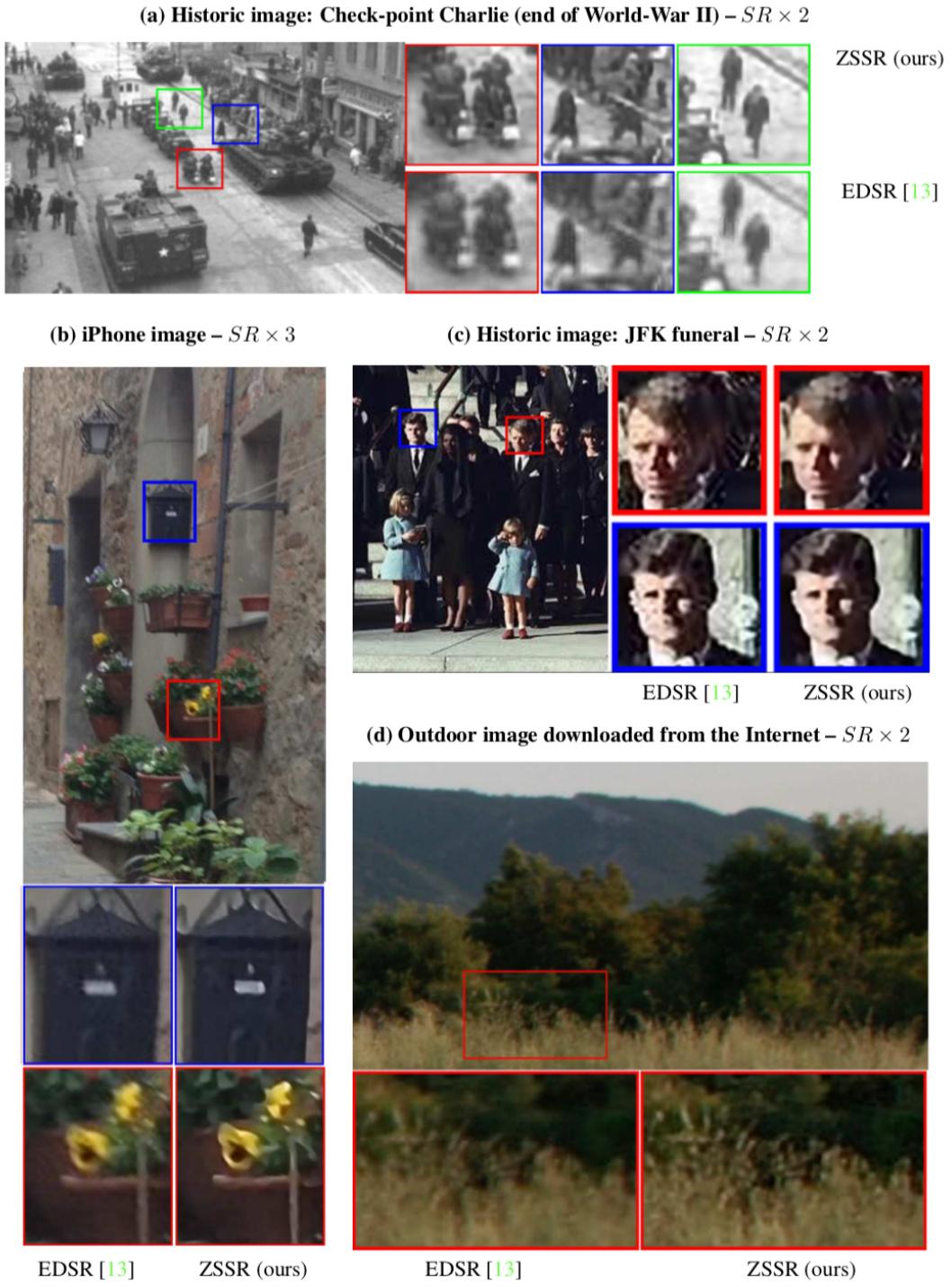


Figure 1: **SR of real images (unknown LR acquisition process).** Real-world images rarely obey the ‘ideal conditions’ assumed by supervised SR methods. For example, old historic photos (a,c), images taken by smartphones (b), random images on the Internet (d), etc. Since ZSSR trains at test time on examples extracted from the test image, it is better at performing SR ‘In-the-Wild’ (i.e., in unconstrained and unknown settings). Full sized images can be found on our [project website](#).

其他可视化结果和所有的图像可以在我们的[项目网站](#)上找到。

**(A) 非理想 downscaling kernel:** 这个实验的目的是测试更多具有数值评估结果能力的真实模糊 kernels。为了实现这个目标，我们通过使用随机高斯 kernel downscaling HR 图像的方法，从 BSD100 中构建了一个新的数据集。对于每个图像，它downscaling kernel 的协方差矩阵  $\sum$  被选择为每个轴上

具有随机角度  $\theta$  和随机长度  $\lambda_1, \lambda_2$ :

$$\lambda_1, \lambda_2 \sim U[0, s^2], \theta \sim U[0, \pi], \Lambda = \text{diag}(\lambda_1, \lambda_2), U = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \Sigma = U \Lambda U^t,$$

这里的  $s$  表示 HR-LR 的 downscaling factor。因此，每一个 LR 图像能够通过不同的随机 kernel 进行下采样。Table 2 比较了我们模型与 leading 外部有监督的模型的性能。

VDSR [9]	EDSR+ [13]	Blind-SR [15]	ZSSR [estimated kernel] (ours)	ZSSR [true kernel] (ours)
27.7212 / 0.7635	27.7826 / 0.7660	28.420 / 0.7834	28.8118 / 0.8306	29.6814 / 0.8414

Table 2: **SR in the presence of unknown downscaling kernels.** LR images were generated from the BSD100 dataset using random downscaling kernels (of reasonable size). SR $\times 2$  was then applied to those images. Please see text for more details.

我们同时将我们模型与无监督的 Blind-SR 方法的性能惊醒了比较。采用 ZSSR 时，我们考虑了以下两种情况：(i) 未知 downscaling kernel 的更现实的情况。对于这种模式，我们使用《Nonparametric blind super-resolution [15]》直接从测试图像评估 kernel 并将其传给 ZSSR。我们使用 [15] 通过寻找非参数化的 downscaling kernel 来估计的未知 SR kernel，这种 downscaling kernel 最大化了 LR 测试图像中跨尺度 patches 的相似度。(ii) 我们将带有正确的 downscaling kernel 的 ZSSR 用于构建 LR image。对于已知规格的传感器所获取的图像，这种情况很可能有用。

注意，外部监督方法不能从已知的测试图像（无论是估计的还是真实的）的模糊 kernel 获益，因为他们完全是使用特定的 kernel 来进行训练和优化的。Table 2 展示了，当提供 true kernels 时，比起最先进的方法，ZSSR 要好很多：对于未知（估计） kernels 为 +1dB，对于 true kernels 为 +2dB。

VDSR [9]	EDSR+ [13]	Blind-SR [15]	ZSSR [estimated kernel] (ours)	ZSSR [true kernel] (ours)
27.7212 / 0.7635	27.7826 / 0.7660	28.420 / 0.7834	28.8118 / 0.8306	29.6814 / 0.8414

Table 2: **SR in the presence of unknown downscaling kernels.** LR images were generated from the BSD100 dataset using random downscaling kernels (of reasonable size). SR $\times 2$  was then applied to those images. Please see text for more details.

视觉上，通过最先进的 SR 方法生成的图像非常的模糊（参阅 Fig.2 和[项目网站](#)）有意思的是，[15] 的无监督算法并没有使用深度学习，但同样比最先进的 SR 算法要出色。上述现象支撑了《Accurateblur models vs. image priors in single image super-resolution [18]》的分析和观察：(i)一个精确的 downscaling 模型比复杂的图像先验 (image priors) 更重要。(ii) 使用错误的 downscaling kernel 会导致过于平滑的 SR 结果。

非理想 kernel 的一个特殊例子是会导致锯齿 (aliasing) 的  $\delta$  kernel。最先进的方法也不能够很好的处理上述的例子（参阅 Fig. 2）

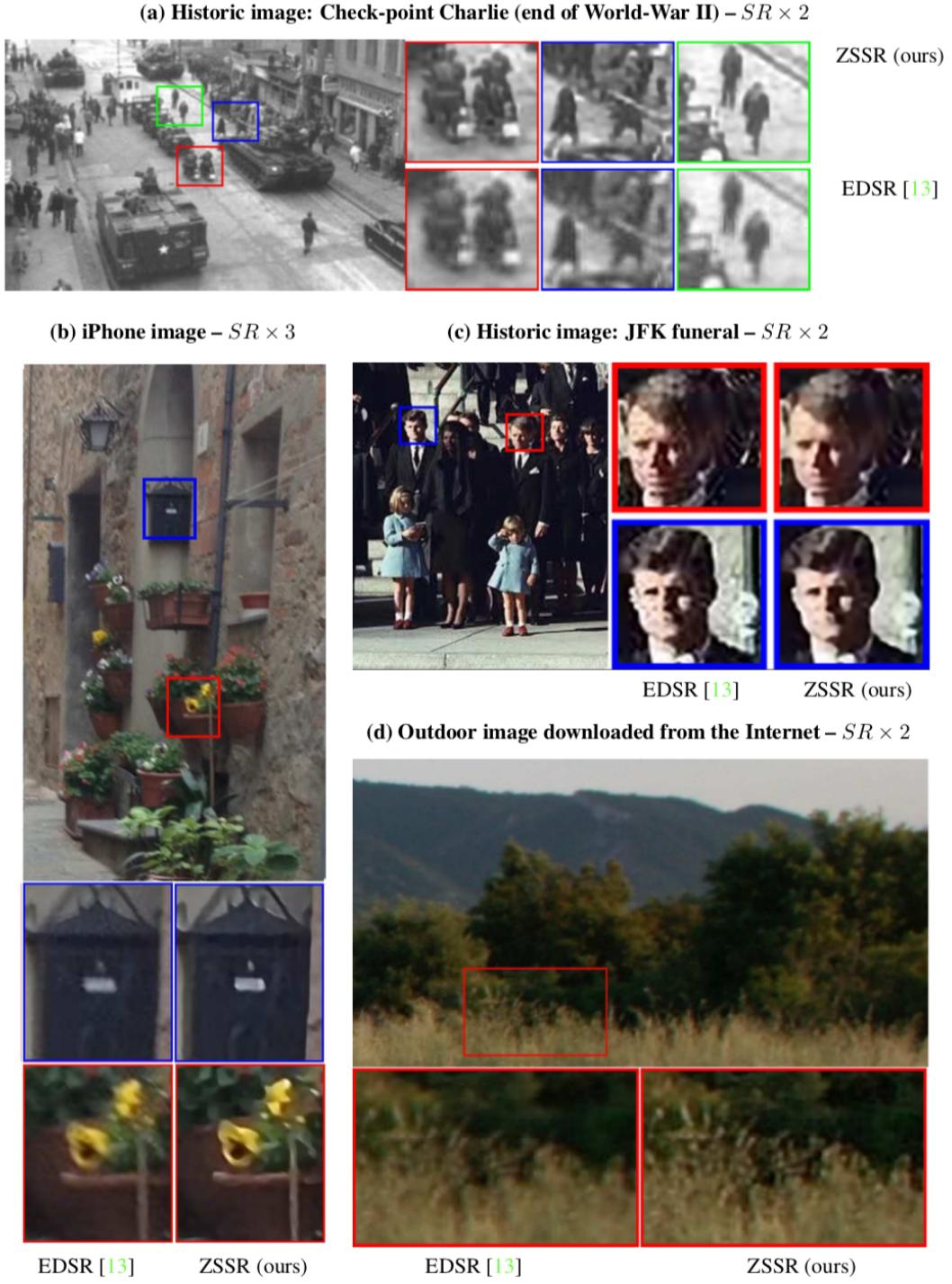


Figure 1: **SR of real images (unknown LR acquisition process).** Real-world images rarely obey the ‘ideal conditions’ assumed by supervised SR methods. For example, old historic photos (a,c), images taken by smartphones (b), random images on the Internet (d), etc. Since ZSSR trains at test time on examples extracted from the test image, it is better at performing SR ‘In-the-Wild’ (i.e., in unconstrained and unknown settings). Full sized images can be found on our [project website](#).

**(B) 低质量 LR 图像：** 在这个实验中，我们测试了不同质量 degradation 类型的图像。为了测试 ZSSR 的在应对未知损坏时的鲁棒性，我们从 BSD100 中挑选的每一张图像都符合三种 degradations 中随机一种：(i) 高斯噪声  $[\sigma = 0.05]$ ，(ii) 斑点噪声  $[\sigma = 0.05]$ ，(iii) JPEG 压缩 [质量=45 (MATLAB 标准)]。Table 3 展示了对于未知的 degradation 类型，ZSSR 是 robust 的，然而这些 degradation 类型

通常会破坏有监督的 SR 算法，使 bicubic 算法优于当前的最先进的 SR 方法。

**与 SRGAN 比较：** SRGAN 也是为理想样本设计的。在这种情况下，SRGAN 方法往往会产生视觉上令人愉悦的幻觉，因此分数会比 ZSSR 的低。在非理想样本中，该模型获得了非常差的视觉表现（参考 Fig. 7）。



Figure 7: **Visual comparison of ZSSR to SRGAN [12]** (*using the code of [1]*). SRGAN obtains poor visual quality on ‘non-ideal’ LR images – Please zoom-in on screen.

## 5. Conclusion

我们介绍了“Zero-Shot”SR，它利用了深度学习的优势，不依赖任何外部样本或预训练。它通过一个小的 image-specific CNN 来获取 SR 预测，这种 CNN 在测试阶段仅使用从 LR 测试图像中提取的内部样本来训练。它将提供对真实图像的 SR 处理，这些图像的获取过程是非理想的，未知的，以及随图像的变化而变化的（例如 image-specific 设置）。在这样一个真实世界“非理想的”设置下，我们的方法在质量上和数量上都远远胜于最先进的 SR 方法。据我们所知，这是第一个无监督的，基于 CNN 的 SR 算法。

## 2. 论文复现

### 2.1 神经网络结构

本论文的项目地址为：<https://github.com/assafshocher/ZSSR>，环境为 python 2.7, tensorflow v1.x。代码针对不同的使用情况，提供了以下几种配置：

### Quick usage examples (applied on provided data examples):

Usage example to test 'Set14', Gradual SR (~0.3dB better results, 6x Runtime)

```
python run_ZSSR.py X2_GRADUAL_IDEAL_CONF
```

Usage example to test 'Set14' (Non-Gradual SR)

```
python run_ZSSR.py X2_ONE_JUMP_IDEAL_CONF
```

Visualization while running (Recommended for one image, interactive mode, for debugging)

```
python run_ZSSR.py X2_IDEAL_WITH_PLOT_CONF
```

Applying a given kernel

```
python run_ZSSR.py X2_GIVEN_KERNEL_CONF
```

Run on a real image

```
python run_ZSSR.py X2_REAL_CONF
```

为了简便以及考虑到自身能力有限，这里只探讨 `python run_ZSSR.py X2_REAL_CONF` 的运行结果，并以此构建神经网络结构。

通过论文 3.1. Architecture & Optimization 中的下面这段内容

We use a simple, fully convolutional network, with 8 hidden layers, each has 64 channels. We use ReLU activations on each layer. The network input is interpolated to the output size. As done in previous CNN-based SR methods [10, 9, 4], we only learn the *residual* between the interpolated LR and its HR parent. We use  $L_1$  loss with ADAM optimizer [11]. We start with a learning rate of 0.001. We periodically take a linear fit of the reconstruction error and if the standard deviation is greater by a factor than the slope of the linear fit we divide the learning rate by 10. We stop when we get to a learning rate of  $10^{-6}$ .

可以看出，论文中构建的卷积层由 8 个隐藏层构成，每层 64 个通道（意味着我们需要构建 64 个过滤器），每一层都对于线性输出都有一个 ReLU 激活函数，没有池化层，整体使用 ADAM 优化器，开

始的学习率为 0.001，学习到  $10^{-6}$  为止。通过阅读源代码，能够进一步了解网络结构。

通过阅读在项目中的 [ZSSR.py](#) 中的 build\_network() 函数，

```
def build_network(self, meta):
    with self.model.as_default():

        # Learning rate tensor
        self.learning_rate_t = tf.placeholder(tf.float32, name='learning_rate')

        # Input image
        self.lr_son_t = tf.placeholder(tf.float32, name='lr_son')

        # Ground truth (supervision)
        self.hr_father_t = tf.placeholder(tf.float32, name='hr_father')

        # Filters
        self.filters_t = [tf.get_variable(shape=meta.filter_shape[ind], name='filter_%d' % ind,
                                           initializer=tf.random_normal_initializer(
                                               mean=0, stddev=np.sqrt(meta.init_variance / np.prod(meta.filter_shape[ind][0:3]))))
                          for ind in range(meta.depth)]

        # Activate filters on layers one by one (this is just building the graph, no training here)
        self.layers_t = [self.lr_son_t] + [None] * meta.depth
        for l in range(meta.depth - 1):
            self.layers_t[l + 1] = tf.nn.relu(tf.nn.conv2d(self.layers_t[l], self.filters_t[l],
                                                          [1, 1, 1, 1], "SAME", name='layer_%d' % (l + 1)))

        # Last conv layer (Separate because no ReLU here)
        l = meta.depth - 1
        self.layers_t[-1] = tf.nn.conv2d(self.layers_t[l], self.filters_t[l],
                                         [1, 1, 1, 1], "SAME", name='layer_%d' % (l + 1))

        # Output image (Add last conv layer result to input, residual learning with L1 loss)
        self.net_output_t = self.layers_t[-1] + self.conf.learn_residual * self.lr_son_t

        # Final loss (L1 loss between label and output layer)
        self.loss_t = tf.reduce_mean(tf.reshape(tf.abs(self.net_output_t - self.hr_father_t), [-1]))

        # Apply adam optimizer
        self.train_op = tf.train.AdamOptimizer(learning_rate=self.learning_rate_t).minimize(self.loss_t)
        self.init_op = tf.initialize_all_variables()
```

在下段代码中

```

self.layers_t = [self.lr_son_t] + [None] * meta.depth
for l in range(meta.depth - 1):
    self.layers_t[l + 1] = tf.nn.relu(tf.nn.conv2d(self.layers_t[l], self.filters_t[l],
                                                [1, 1, 1, 1], "SAME", name='layer_%d' % l))

```

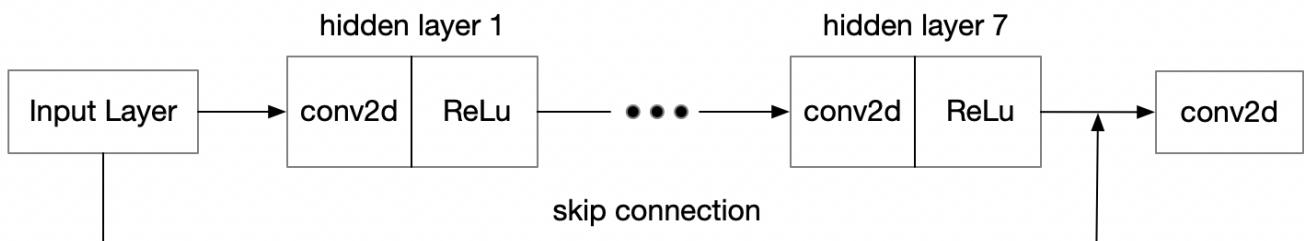
能发现，实际上卷积层的最后一层隐藏层是没有使用 ReLU 函数处理的。并且，下段代码

```

# Output image (Add last conv layer result to input, residual learning with global skip
self.net_output_t = self.layers_t[-1] + self.conf.learn_residual * self.lr_son_t

```

表明最后一层会进行 skip connection (ResNet 跳远连接)，以此我确定了卷积层的模型如下图：



论文中关于鲁棒性的探讨以及使用的方法我没有弄明白。

For the sake of robustness, as well as to allow large SR scale factors  $s$  even from very small LR images, the SR is performed gradually [5, 21]. Our algorithm is applied for several intermediate scale-factors ( $s_1, s_2, \dots, s_m = s$ ). At each intermediate scale  $s_i$ , we add the generated SR image  $\text{HR}_i$  and its downsampled/rotated versions to our gradually growing training-set, as new HR fathers. We downscale those (as well as the previous smaller ‘HR examples’) by the next gradual scale factor  $s_{i+1}$ , to generate the new LR-HR training example pairs. This is repeated until reaching the full desired resolution increase  $s$ .

于是我去看了代码文件 [ZSSR.py](#) 中 run() 中的 train() 函数，代码如下：

```

def train(self):
    # main training loop
    for self.iter in xrange(self.conf.max_iters):
        # Use augmentation from original input image to create current father.
        # If other scale factors were applied before, their result is also used (hr_
        self.hr_father = random_augment(ims=self.hr_fathers_sources,
                                         base_scales=[1.0] + self.conf.scale_factors,
                                         leave_as_is_probability=self.conf.augment_le_
                                         no_interpolate_probability=self.conf.augmen_
                                         min_scale=self.conf.augment_min_scale,
                                         max_scale=(1.0] + self.conf.scale_factors)
                                         allow_rotation=self.conf.augment_allow_rota_
                                         scale_diff_sigma=self.conf.augment_scale_di_
                                         shear_sigma=self.conf.augment_shear_sigma,
                                         crop_size=self.conf.crop_size)

        # Get lr-son from hr-father
        self.lr_son = self.father_to_son(self.hr_father)

        # print '%%%%%%%%%%%%%%'
        print self.hr_father
        new_data = np.ceil(self.hr_father*256)
        test_im = Image.fromarray(np.uint8(new_data))
        test_im.save('father.jpg')
        print ""
        print self.lr_son
        new_data = np.ceil(self.lr_son*256)
        test_im = Image.fromarray(np.uint8(new_data))
        test_im.save('son.jpg')
        # print '%%%%%%%%%%%%%%'

        # run network forward and back propagation, one iteration (This is the heart)
        self.train_output = self.forward_backward_pass(self.lr_son, self.hr_father)

        # Display info and save weights
        if not self.iter % self.conf.display_every:
            print 'sf:', self.sf*self.base_sf, ', iteration: ', self.iter, ', loss: ', self.train_output

        # Test network
        if self.conf.run_test and (not self.iter % self.conf.run_test_every):
            self.quick_test()

        # Consider changing learning rate or stop according to iteration number and
        self.learning_rate_policy()

        # stop when minimum learning rate was passed
        if self.learning_rate < self.conf.min_learning_rate:
            break

```

论文中的下段话

To accelerate the training stage and make the *runtime independent of the size* of the test image  $I$ , at each iteration we take a *random crop of fixed size* from a randomly-selected father-son example pair. The crop is typically  $128 \times 128$  (unless the sampled image-pair is smaller). The probability of sampling a LR-HR example pair at each training iteration is set to be non-uniform and proportional to the size of the HR-

father. The closer the size-ratio (between the HR-father and the test image  $I$ ) is to 1, the higher its probability to be sampled. This reflects the higher reliability of non-synthesized HR examples over synthesize ones.

表明为了加速训练，我们会对每一个 father-son 样本对进行随机范围的裁剪，大小为  $128*128$ ，而我在使用下段代码查看 father-son 样本对的时候

```
print '%%%%%%%%%%%%%%'
print self.hr_father
new_data = np.ceil(self.hr_father*256)
test_im = Image.fromarray(np.uint8(new_data))
test_im.save('father.jpg')
print ""
print self.lr_son
new_data = np.ceil(self.lr_son*256)
test_im = Image.fromarray(np.uint8(new_data))
test_im.save('son.jpg')
print '%%%%%%%%%%%%%%'
```

发现，father 是在原有图像的基础上进行随机裁剪，大小为  $128 \times 128$ ，而 son 是在 father 的基础上缩小到  $64 \times 64$  获得的。按照 father\_to\_son() -> imresize() 的过程参阅源代码会发现，作者在缩小图像的过程中，提供了不止一种缩放方法，这点可以在 [imresize.py](#) 文件中 imresize() 中的下段代码中看到

```
method, kernel_width = {
    "cubic": (cubic, 4.0),
    "lanczos2": (lanczos2, 4.0),
    "lanczos3": (lanczos3, 6.0),
    "box": (box, 1.0),
    "linear": (linear, 2.0),
    None: (cubic, 4.0) # set default interpolation method as cubic
}.get(kernel)
```

默认使用的是 cubic 方式。在下采样的时候，作者还会为图像进行抗锯齿处理，代码如下：

```

# Antialiasing is only used when downscaling
antialiasing *= (scale_factor[0] < 1)
# Sort indices of dimensions according to scale of each dimension. since we are going down
sorted_dims = np.argsort(np.array(scale_factor)).tolist()
# Iterate over dimensions to calculate local weights for resizing and resize each time
out_im = np.copy(im)
for dim in sorted_dims:
    # No point doing calculations for scale-factor 1. nothing will happen anyway
    if scale_factor[dim] == 1.0:
        continue
    # for each coordinate (along 1 dim), calculate which coordinates in the input image
    # weights that multiply the values there to get its result.
    weights, field_of_view = contributions(im.shape[dim], output_shape[dim], scale_factor,
                                            method, kernel_width, antialiasing)
    # Use the affecting position values and the set of weights to calculate the result
    out_im = resize_along_dim(out_im, dim, weights, field_of_view)

```

论文 2. The Power of Internal Image Statistics 中有下段描述：

We further enrich the training set by transforming each LR-HR pair using 4 rotations ( $0^\circ$ ,  $90^\circ$ ,  $180^\circ$ ,  $270^\circ$ ) and their mirror reflections in the vertical and horizontal directions. This adds  $\times 8$  more image-specific training examples.

会发现作者为了扩充数据集，对原先的图像进行了四个角度的旋转，以及四个角度的垂直，水平镜像处理。[ZSSR.py](#) 中 final\_test() 中的代码体现了这一点：

```

def final_test(self):
    # Run over 8 augmentations of input - 4 rotations and mirror (geometric self ensemble)
    outputs = []
    # The weird range means we only do it once if output_flip is disabled
    # We need to check if scale factor is symmetric to all dimensions, if not we will do
    for k in range(0, 1 + 7 * self.conf.output_flip, 1 + int(self.sf[0] != self.sf[1])):
        # Rotate *k degrees and mirror flip when k>=4
        test_input = np.rot90(self.input, k) if k < 4 else np.fliplr(np.rot90(self.input, k))
        print '%%%%%%%%%%%%%%'
        print test_input.shape
        print '%%%%%%%%%%%%%%'
        # Apply network on the rotated input
        tmp_output = self.forward_pass(test_input)
        # Undo the rotation for the processed output (mind the opposite order of the flip)
        tmp_output = np.rot90(tmp_output, -k) if k < 4 else np.rot90(np.fliplr(tmp_output), -k)
        # fix SR output with back projection technique for each augmentation
        for bp_iter in range(self.conf.back_projection_iters[self.sf_ind]):
            tmp_output = back_projection(tmp_output, self.input, down_kernel=self.kernel,
                                         up_kernel=self.conf.upscale_method, sf=self.sf)
        # save outputs from all augmentations
        outputs.append(tmp_output)
    # Take the median over all 8 outputs
    almost_final_sr = np.median(outputs, 0)
    # Again back projection for the final fused result
    for bp_iter in range(self.conf.back_projection_iters[self.sf_ind]):
        almost_final_sr = back_projection(almost_final_sr, self.input, down_kernel=self.kernel,
                                           up_kernel=self.conf.upscale_method, sf=self.sf)
    # Now we can keep the final result (in grayscale case, colors still need to be added
    # because it is done before saving and for every other purpose we use this result)
    self.final_sr = almost_final_sr
    # Add colors to result image in case net was activated only on grayscale
    return self.final_sr

```

这里我没弄清楚 `self.sf` 代表的含义，以及 for 循环中的 range 的计算方式。我在实验之前猜想是对裁剪后的样本对进行四个角度的反转，以及翻转后的垂直、水平镜像处理，为了验证这个想法，我使用

```

print '%%%%%%%%%%%%%%'
print test_input.shape
print k
print '%%%%%%%%%%%%%%'

```

进行输出，结果如下图（并未对 `real_example` 中的图像进行修改，这里的图像仍旧是项目中自带的图像）：

```
ZSSR --bash -- 80x24
sf: [2. 2.] , iteration: 1400 , loss: 0.016697695
iteration: 1400 reconstruct mse: 0.0005674512 , true mse: None
sf: [2. 2.] , iteration: 1420 , loss: 0.016247483
sf: [2. 2.] , iteration: 1440 , loss: 0.019970056
iteration: 1450 reconstruct mse: 0.00056859 , true mse: None
sf: [2. 2.] , iteration: 1460 , loss: 0.01871764
sf: [2. 2.] , iteration: 1480 , loss: 0.023358941
sf: [2. 2.] , iteration: 1500 , loss: 0.018704847
iteration: 1500 reconstruct mse: 0.00056617835 , true mse: None
sf: [2. 2.] , iteration: 1520 , loss: 0.01751154
sf: [2. 2.] , iteration: 1540 , loss: 0.0253068
iteration: 1550 reconstruct mse: 0.0005690336 , true mse: None
sf: [2. 2.] , iteration: 1560 , loss: 0.025547614
sf: [2. 2.] , iteration: 1580 , loss: 0.020803662
sf: [2. 2.] , iteration: 1600 , loss: 0.025214514
iteration: 1600 reconstruct mse: 0.0005695878 , true mse: None
slope: 9.433715604245834e-09 STD: 8.319366069255324e-09
learning rate updated: 1e-06
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
(346, 550, 3)
0
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
** Done training for sf= [2.0, 2.0] **
MacBook-Pro:ZSSR outro$
```

能够明显看出这里只跑了一次循环，而且针对原图做了一次（或三次） $90^\circ$  旋转，未进行镜像，**这里仍旧不太清楚原因**，但在之后构建神经网络的时候我会按照论文中所说的，对随机 crop 的图像块进行旋转并镜像，用这些数据来训练神经网络，根据效果再调整策略。在执行了下行代码：

```
tmp_output = self.forward_pass(test_input)
```

对被处理的图像进行一次前向传播之后，如下段代码所示：

```
# Undo the rotation for the processed output (mind the opposite order of the flip and t
tmp_output = np.rot90(tmp_output, -k) if k < 4 else np.rot90(np.fliplr(tmp_output), -k)
# fix SR output with back projection technique for each augmentation
for bp_iter in range(self.conf.back_projection_iters[self.sf_ind]):
    tmp_output = back_projection(tmp_output, self.input, down_kernel=self.kernel,
                                 up_kernel=self.conf.upscale_method, sf=self.sf)
```

作者将图像还原，并使用反投影方法（例如将图像 A 放大至图像 B 的处理，反投影的思路为：处理 A 进行一次前向传播得到 B 之后，将 B 下采样处理得到 C，计算 C 和 A 之间的残差 D，然后再将残差 D 前向传播处理得到放大的残差 E，最后将 E 和 B 相加并输出，这里参考了

<https://zhuanlan.zhihu.com/p/50192019> 中对 DBPN 的描述）

综上，可以简单的总结出代码对数据的大致处理：

1. 读取输入图像 A, 对图像随机进行 crop, 得到图像 B
2. 对 B 进行 cubic 缩小, 进行抗锯齿操作得到图像 C
3. 将 B 和 C 作为 input 和 label 丢进神经网络
4. 重复步骤 1~3 n 次循环, 代码中的建议最低次数为 256
5. 对输入图像 A 根据某种条件  $\beta$  进行旋转镜像处理得到图像 D
6. 将 D 丢进前向传播得到输出 E
7. 对 E 根据某种条件  $\alpha$  进行反投影得到输出 F
8. 将 F 加入输出集合 output\_set
9. 重复步骤 5~8, 循环次数为某种条件  $\theta$
10. 对输出集合 output\_set 取中位图像 G
11. 再根据条件  $\alpha$  对 G 进行反投影
12. 输出被处理后的图像 G

上述被标记为红色的步骤均有不确定条件, 在执行代码后, 我查看了函数 final\_test() 中的条件  $\alpha$  (`(self.conf.back_projection_iters[self.sf_ind])`) 的输出, 值为 0。因此, 我决定在复现的模型中使用下述步骤:

1. 读取输入图像 A, 对图像随机进行 crop, 得到图像 B
2. 对 B 进行 cubic 缩小, 进行抗锯齿操作得到图像 C
3. 将 B 和 C 作为 input 和 label 丢进神经网络
4. 重复步骤 1~3, n 次, 代码中的建议最低次数为 256
5. 对输入图像 A 进行旋转镜像处理得到图像 D
6. 将 D 丢进前向传播得到输出 E
7. 将 F 加入输出集合 output\_set
8. 重复步骤 5~6, 直到做完 4 次旋转及其垂直水平镜像共八次变换
9. 对输出集合 output\_set 取中位图像 G
10. 输出图像

整体框架如下图所示:

