Example-Based Image Super-Resolution Techniques

Mark Sabini – `msabini` & Gili Rusak – `gili`

December 17, 2016

1 Introduction

With the current surge in popularity of image-based applications, improving content quality is vital. While hardware-based solutions do exist, an approach called image super-resolution adopts a more software-based approach. Specifically, the goal of image super-resolution is to sharpen or improve the quality of a low-resolution (LR) image input by outputting a super-resolved (SR) high-resolution (HR) image output. As this problem is inherently ill-poised (as there may be multiple reconstructions of the same image), there have been many efforts over the past thirty years to develop high-performance image super-resolution techniques.

2 Literature Review

The seminal work on image-super resolution comes from Tsai & Huang [10], who detailed a frequency domain approach which reconstructed a HR image from a collection of LR images of the same source with the aid of signal processing techniques. Unfortunately, frequency domain-based methods are sensitive to model errors, so other approaches often operate within the spatial domain instead. One example of such an approach is detailed by Glasner et al. [5], who combine classical multiframe image super-resolution and patch redundancy within the same image. Glasner's approach effectively unites the two methods and can obtain super-resolution from a single input image.

Current state-of-the-art methods leverage deep neural networks to create faster and more precise image super-resolution outcomes [6]. For example, a recent work by Johnson et. al. presents an architecture that is extremely effective at reconstructing images with fine detail by utilizing a perceptual loss function instead of the Euclidean loss [6]. Other papers have since added to this technique by experimenting with different networks.

In this project, we have experimented with three different implementations of image super-resolution algorithms. The first method, inspired by Freeman et al. [4], utilizes k-nearest neighbors (k-NN) as a rote learning baseline in order to generate a mapping from low-resolution patches to high-resolution patches. The second method, based on one given by Li & Simske [7], uses support vector regression to learn LR patch to HR pixel mappings. The final method uses a super-resolution convolutional neural network (SRCNN) [2] to learn transformations from entire LR images to HR images.

The most common metric for measuring a meaningful increase in the resolution of an image is the peak signal-to-noise ratio (PSNR), computed for an $m \times n$ input image I relative to the ground truth image T as follows:

$$\mathrm{PSNR}_T(I) = 20 \cdot \log_{10}\left(\frac{255}{\sqrt{mn/\ell_2(T,I)}}\right)$$

We adopt the PSNR as the primary metric for measuring super-resolution performance.

3 Dataset

Our training dataset, which we denote D_{91}, consisted of 91 color images from ImageNet, as described by Chao et al. [2]. These images ranged in content from flowers and animals to cars and other household objects. These same images ranged in size from 75×75 to 375×375 pixels. Some examples of images from D_{91} are shown below:

Our testing dataset consisted of single, color images, which we varied for different experiments.

4 Algorithms

For the sake of clarity, we will refer to H and H_0 as HR training and test images, respectively, and L and L_0 as LR training and test images, respectively. In addition, we introduce the operators $\downarrow_s$ and $\uparrow_s$, which downscale and upscale images by a scale factor of s, respectively. Unless otherwise noted, it is assumed that bicubic interpolation is used.

We used MATLAB for our implementations of k-NN and SVR. We trained our SRCNN models on the ICME GPU cluster using Caffe.

4.1 k-Nearest Neighbors

k-nearest neighbors (k-NN) is a non-parametric rote learning algorithm which is well-known for its simplicity. At a high level, given a training set $\{(x^{(1)}, y^{(1)}), \cdots, (x^{(m)}, y^{(m)})\}$ and test example x, k-NN finds the indices $i \in \{i_1, \cdots, i_k\}$ of the k training examples that best minimize the Euclidean distance $\ell_2(x, x^{(i)})$. The predicted y is then $\frac{1}{k}\sum_{j=1}^{k} y^{(i_j)}$.

We base our work on an existing paper by Freeman et al. [4] that describes an example-based image super-resolution technique. In order to reduce noise and artifacts, we used a k-nearest neighbors approach as suggested by Yang & Huang [11].

Using a set of training images, we first blur and downsample each input image H to derive a low-resolution image $L = (H * B) \downarrow_2$ (B is the chosen blur kernel), and then upscale and interpolate L to derive $L_{\text{interp}} = L \uparrow_2$, a low-resolution image with the same dimensions as H. Then, for each 5×5 patch p_L in L, we compute the difference d between the corresponding 10×10 patch in H and L_{interp}. Afterwards, p_L and d are flattened and concatenated to form a training example, and then stored. In the prediction phase, we first use bicubic interpolation to upscale the low-resolution image L_0 to an image $L_0' = L_0 \uparrow_2$ with the same size as the high resolution image. We then search the training set for the k-nearest neighbors of each patch and add the average of the corresponding differences to the respective patch in L_0'. The final high-resolution output is then L_0'.

4.2 Support Vector Regression

Support vector regression (SVR) utilizes the same techniques as support vector classification (SVC), but instead attempts to learn a function approximating the output. SVR, which was first proposed by Vapnik et al. [9], solves the following optimization problem [1]:

$$\begin{aligned} \min_{\mathbf{w}, b, \xi, \xi^*} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ \text{subject to} \quad & \mathbf{w}^T\phi(\mathbf{x}_i) + b - z_i \leq \epsilon + \xi_i, \\ & z_i - \mathbf{w}^T\phi(\mathbf{x}_i) - b \leq \epsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \geq 0,\ i = 1, \cdots, l \end{aligned}$$

For our second approach, we applied ϵ-SVR from LIBSVM with the radial basis function (RBF) kernel to image super-resolution. We based our approach on a paper by Li & Simske [7]. In this paper, the authors again take advantage of patch-based local similarities in training images. The high-level difference between this approach and the k-NN approach is that the mapping learned is now patch-to-pixel, rather than patch-to-patch.

To begin, each high-resolution color training image H is again blurred and downsampled to derive a low-resolution training image $L = (H * B) \downarrow_2$. This is then upscaled and interpolated to derive $L_{\text{interp}} = L \uparrow_2$. For each channel in L_{interp}, we construct a training set for SVR, by mapping each $2p + 1 \times 2p + 1$ patch to the corresponding center pixel of the corresponding channel of H. We then train an ϵ-SVR model for each channel using the RBF kernel, and found the optimal hyperparameters to be $\epsilon = 0.2$, $C = 362$. In the prediction phase, we first upsample the low-resolution image L_0 to get $(L_0)_{\text{interp}} = L_0 \uparrow_2$. Then, we generate the high resolution image L_0' by iterating over the $2p+1 \times 2p+1$ patches in $(L_0)_{\text{interp}}$ to predict corresponding high resolution pixels for each channel.

Stock SVR was infeasible to train locally on MATLAB, due to memory and time constraints.[1] Thus, we experimented with several variations to shrink the training set including random patch sampling and k-means clustering. For random patch sampling, we collected all LR patches from D_{91} and randomly selected k of them on which to train our SVR. For k-means clustering, we clustered all LR patches from D_{91} into k clusters. We then formed a reduced-size training set by mapping each cluster centroid to the average intensity of the HR pixels in that cluster.

We also experimented with single-image SVR, in a manner similar to Glasner et al. [5]. Essentially, given an LR input image L_0, we first train an SVR model using $(L_0) \downarrow_2$ as a LR training image and L_0 as a surrogate for the HR ground truth. Then, we apply our model to L_0 to predict the supposed original HR image, L_0'.

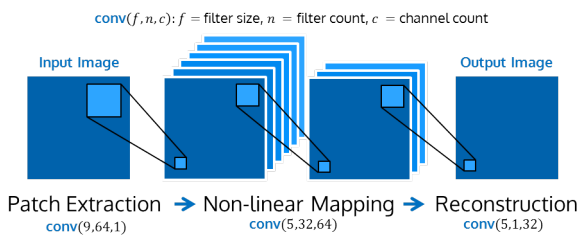
4.3 Super-Resolution Convolutional Neural Network

Convolutional neural networks (CNNs) have been used for many image processing tasks such as classification, and have recently been applied to image super-resolution. CNNs typically include convolutional layers, pooling layers, and fully-connected layers, but super-resolution literature tends to rely on architectures that only include the first type. A convolutional layer, denoted $\text{conv}(f, n, c)$, describes a filter of size $f \times f$. This filter describes an affine transformation $G(Y) = W * Y + B$ for a weights matrix W and bias matrix B. A non-linearity is applied to the output $G(Y)$ through a Rectified Linear Unit (ReLU) layer, resulting in $F(Y) = \max(0, G(Y))$.

[1] Just a single 200×200 image alone produces $196^2 = 38,416$ LR patches.

In addition to f, the numbers n and c describe the depth of the output and input, respectively. When put together, these convolutional layers form a CNN.

We explored the SRCNN approach detailed by Dong et al. [2], which aims to learn a mapping from LR images directly to their HR versions. The model's goal is to recover $F(Y)$ from the input image Y that is as close to the original high resolution image as possible. For training, we first build a training set that maps LR patches to HR patches. From here, we can use Caffe to train the SRCNN and determine the model weights. The architecture of the SRCNN proposed by Dong et al. [2] is:



This architecture, referred to as 9-1-5 (due to the filter sizes), consists of a three-step process comprising patch extraction, non-linear mapping, and reconstruction. The paper minimizes the mean squared error loss between the reconstructed images and the original ground-truth images. Given a LR image L_0, we predict the HR version by first upscaling and interpolating to get $(L_0)_{\text{interp}} = L_0 \uparrow_3$. We then feed $(L_0)_{\text{interp}}$ through the SRCNN, and derive our HR output $L_0' = F((L_0)_{\text{interp}})$.

We varied the SRCNN architecture by retraining our model on 9-3-5, 9-5-5, and 11-1-5 architectures to determine the effect of different filter sizes on image super-resolution performance. We also modified the existing algorithm to super-resolve color images by learning three different models - one for each color channel.

5 Results

5.1 k-Nearest Neighbors

For the k-NN approach, we train on all 5×5 patches from D_{91} and test on a flower image. As shown in Figure 1, k-NN outperforms bicubic interpolation. Furthermore, increasing k leads to an increase in PSNR. Qualitatively, all super-resolved versions using k-NN look sharper than the original LR image. One shortcoming of k-NN is the risk of artifacts, as multiple identical LR patches may actually map to different HR patches. This is especially apparent when $k = 1$, but increasing k can be shown to mitigate this effect. On average, k-NN was the quickest out of all methods to run, taking 1 minute for training and 2 minutes for prediction.

5.2 Support Vector Regression

For the SVR approach, we wanted to incorporate patches from all of D_{91} as opposed to a single image in order to include textures from a variety of sources. We found that using k-means to reduce the training set size increased PSNR, and allowed us to train on

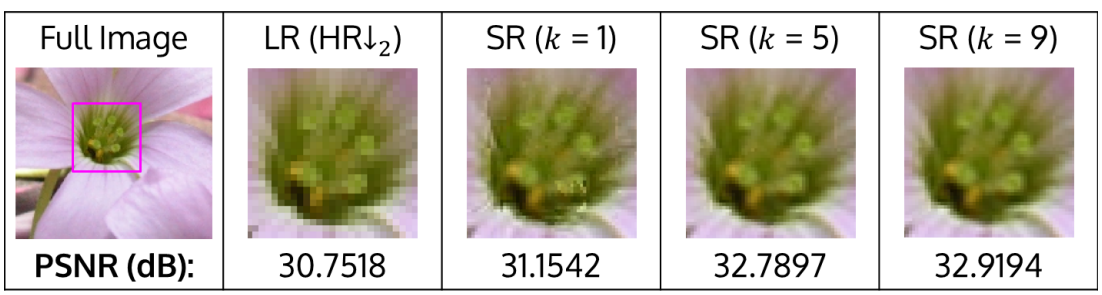


Figure 1: Performance of k-NN with varying values of k

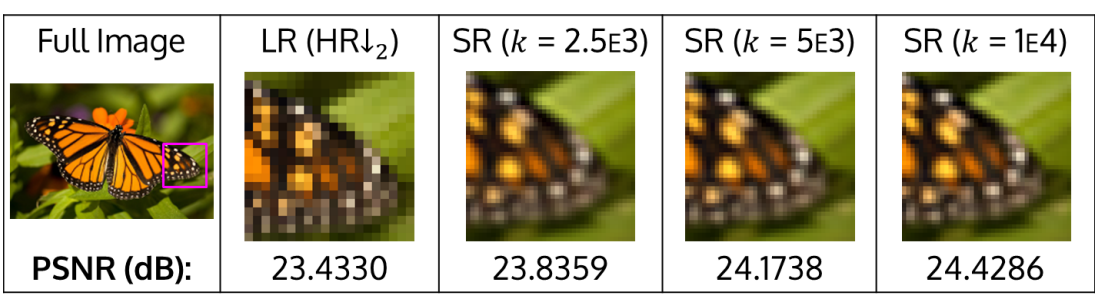


Figure 2: Performance of SVR with varying number of training examples k

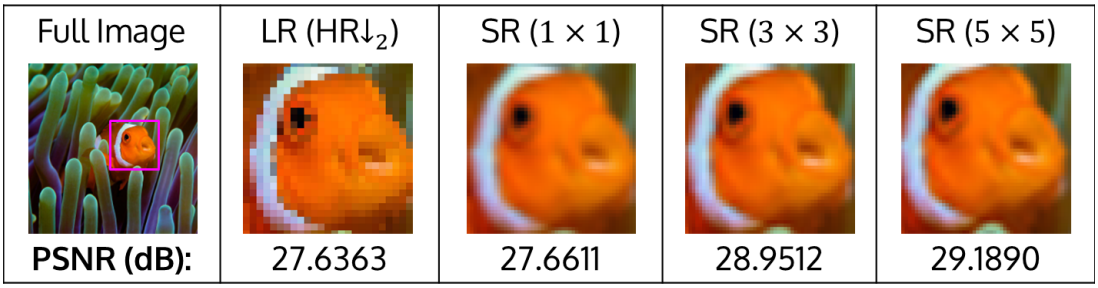


Figure 3: Performance of SVR with varying LR input patch size

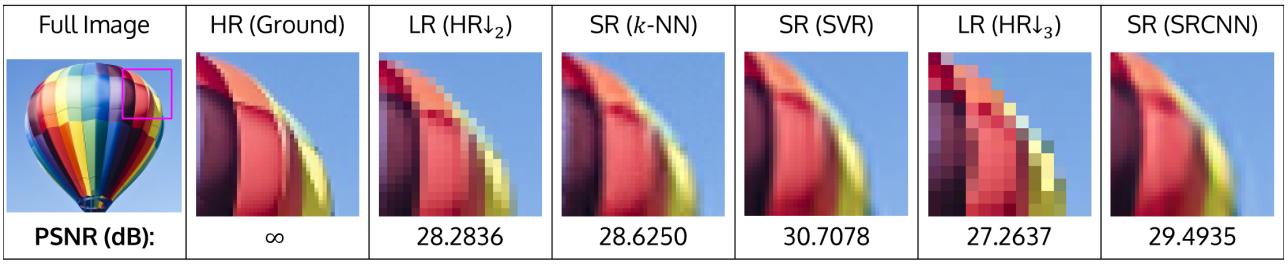


Figure 4: Side-by-side comparison of image SR techniques

all of D_{91}. Specifically, on the butterfly image, k-means SVR with 10000 clusters and a patch size of 7×7 increased PSNR from 23.4330 dB to 24.5241 dB. We theorize that this is so because the cluster centroids from k-means may effectively capture the actual relationship between similar LR patches and their HR pixels. We also found increases in PSNR by randomly sampling patches, as we preserve the original mapping, as shown in Figure 2. One advantage of the latter approach over the former, though, is a much lower runtime due to the speed of random sampling over clustering. On average, SVR with random sampling takes 40 seconds for training and 1 minute for prediction. For comparison, k-means SVR takes 1 minute for training, 3 minutes for clustering, and 1 minute for prediction.

Despite the capabilities of k-means and random sampling to make SVR training more efficient, both methods still potentially incorporate training examples that may not be directly relevant to the image at hand. Following the single-image SVR approach described above, we were able to obtain an increase in PSNR of 1.488 dB on butterfly. This is likely because LR patches may still contain similar characteristics as HR patches of the same image.

We see an improvement in performance of SVR with an increase in number of training examples, as is expected since SVR is provided with a wider variety of patches to learn on. In addition, increasing LR patch size had a similar effect (as shown in Figure 3), since SVR now has access to more spatially local information.

5.3 SRCNN

We trained four different architectures of SRCNN with varying filter sizes but constant depth parameters. For all architectures, we noticed an increase of PSNR with an increase in number of SRCNN training iterations. According to our results in Figure 5(a), a larger filter size for the second convolutional layer corresponded to better super-resolution performance. In addition, increasing the filter size of the first convolutional layer seemed to have little effect. When varying the number of images in the training set, we found that SRCNN is susceptible to overfitting if trained on a single image. As shown in Figure 5(b), the PSNR of the test image for SRCNN trained on the cameraman image actually decreased with the number of iterations. Contrary to this, the SRCNN trained on all of D_{91} actually demonstrated a continued increase in testing PSNR during training. On average, SRCNN required approximately 20 hours to train a million iterations on the ICME GPU cluster and 3 seconds for prediction. Thus, SRCNN is extremely useful for applications where training time is abundant, but predictions must be made quickly.

The SRCNN approach, whose results are shown in Figure 4, yielded comparable increases in PSNR to the SVR approach, even though the former took much longer to train than the latter.

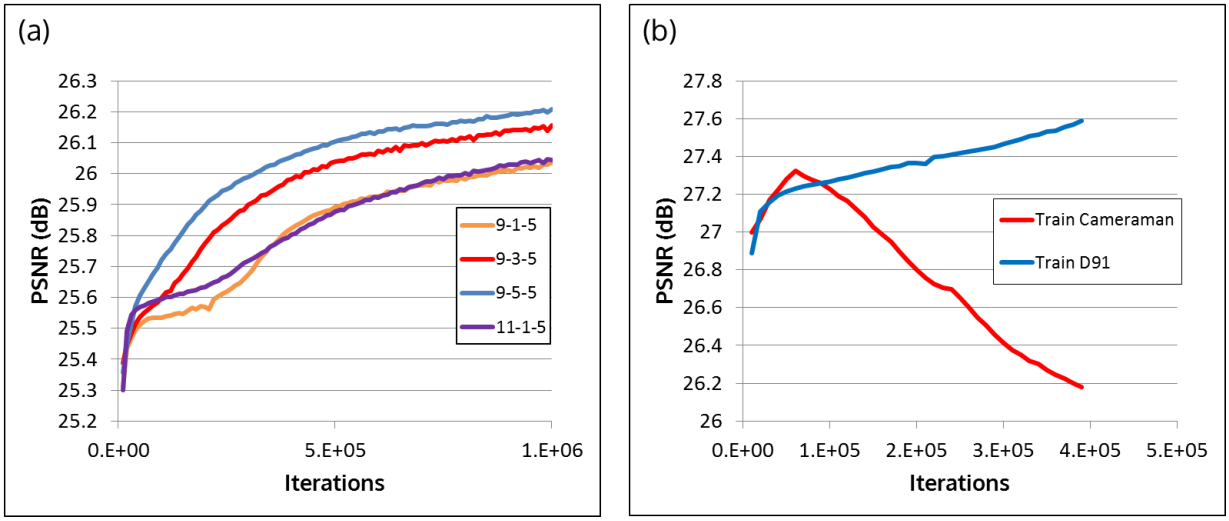


Figure 5: (a) Comparison of SRCNN architectures (b) Effects of training set size on PSNR

5.4 Additional Results

In addition to those presented here, supplementary results are available in an interactive demo at marksabini.com/cs229-imageSR.

6 Future Work

A possible extension to the SRCNN approach includes implementing new architectures that make use of additional layers in our neural network in order to reduce the training time required for a good model. One such example of this is FSRCNN, which incorporates deconvolutional layers and can thus speed up training by a factor of up to 41.3 [3]. For SVR, a feasible next step is to train on larger datasets utilizing GPUs. As an additional method, we plan to investigate sparse coding and its applications to image super-resolution [12].

7 Conclusion

In this project, we have implemented and modified various machine learning-based image super-resolution techniques. The methods all showed a noticeable increase in PSNR, although they stemmed from different intuitions. Regardless, each of the three methods is a testament to the power of software to address hardware limitations.

8 Acknowledgements

We would like to thank Andrew Ng, John Duchi, and Michael Zhu for their continued guidance throughout the course of this project. We would also like to thank Brian Tempero (ICME), Justin Johnson, and Michael Chang for their advice and resources.

References

[1] Chang, Chih-Chung, and Chih-Jen Lin. "LIBSVM: a library for support vector machines." *ACM Transactions on Intelligent Systems and Technology (TIST)* 2.3 (2011): 27.

[2] Dong, Chao, et al. "Image super-resolution using deep convolutional networks." *IEEE transactions on pattern analysis and machine intelligence* 38.2 (2016): 295-307.

[3] Dong, Chao, Chen Change Loy, and Xiaoou Tang."Accelerating the super-resolution convolutional neural network." *European Conference on Computer Vision.* Springer International Publishing, 2016.

[4] Freeman, William T., Thouis R. Jones, and Egon C. Pasztor. "Example-based super-resolution." *IEEE Computer graphics and Applications* 22.2 (2002): 56-65.

[5] Glasner, Daniel, Shai Bagon, and Michal Irani. "Super-resolution from a single image." *2009 IEEE 12th International Conference on Computer Vision.* IEEE, 2009.

[6] Johnson, Justin, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution." *arXiv preprint arXiv:1603.08155* (2016).

[7] Li, Dalong, and Steven Simske. "Example based single-frame image super-resolution by support vector regression." *Journal of Pattern Recognition Research* 5.1 (2010): 104-118.

[8] Ni, Karl S., and Truong Q. Nguyen. "Image superresolution using support vector regression." *IEEE Transactions on Image Processing* 16.6 (2007): 1596-1610.

[9] Smola, Alex, and Vladimir Vapnik. "Support vector regression machines." *Advances in neural information processing systems* 9 (1997): 155-161.

[10] Tsai, R. Y., and Thomas S. Huang. "Multiframe image restoration and registration." *Advances in computer vision and Image Processing* 1.2 (1984): 317-339.

[11] Yang, Jianchao, and Thomas Huang. "Image super-resolution: Historical overview and future challenges." *Super-resolution imaging* (2010): 1-34.

[12] Yang, Jianchao, et al. "Image super-resolution via sparse representation." IEEE transactions on image processing 19.11 (2010): 2861-2873.