Text Super-Resolution and Deblurring using Multiple Support Vector Regression

Roy Blankman Sean McMillan Ross Smith

December 15^{th}, 2011

1 Introduction

In a world that is increasingly digital, there is a significant push to digitize documents of cultural or intellectual value. A large portion of the data to be digitized is text, which poses some challenges to massive digitization efforts. The problem has historically been solved by expensive or complex equipment, such as the document scanners employed by Google or through crowdsourcing the transcription of text that is too low-quality to be transcribed via optical character recognition (OCR), e.g. reCAPTCHA. In order to make scanning large quantities of text cost-effective, the price and complexity of the equipment would need to be reduced significantly. The lower-quality equipment would produce lower-resolution images, and thus, in order to have near-perfect transcription we would require robust methods to super-resolve the low-resolution scans into something that could easily be interpreted by OCR software.

In particular, this paper is concerned with the challenge of reproducing the sharp boundries present in text when trying to super resolve blurred low-resolution text images. We accomplish super resolution by first training multiple support vector machines on corresponding patches of low- and high- resolution images and then using the trained SVM's to predict higher-resolution versions of input images. Finding a method to preserve sharp edges when enhancing images would solve the problem faced with most interpolation methods that produce rather smooth output with poorly defined edges. Additionally, Ni et al. [4] have found that the use of an SVM with a guassian kernel trained on natural images (i.e. images of nature scenes, people, or anything not computer-generated) preserves edges very poorly.

Although images of text have been used previously for training algorithms for super-resolution, these images were taken with a camera and the blurring added to the images was minimal. We attempted text super resolution using synthetic images whose edges were very well defined, and are the only salient features. We investigated how SVR would perform given a heavily blurred version of these synthetic images, and how we might choose parameters to optimize the image enhancement.

2 Previous Work

Image super-resolution consists of producing a clearer version of a low-resolution image, which is quite different from interpolation methods which aim to enlarge and often smooth an image. Linear or even cubic spline interpolation create a softer image that may be good for certain applications, but is horribly suited, for example, to enhance text images. It is obvious that a local image is not enough to produce a higher-resolution version of the image, and in general super resolution is an ill-posed problem. This is obvious since there are many high-resolution images that could generate a low-resolution one after blurring or

downsampling. Most machine learning approaches to super resolution involve a training set of low- and high-resolution image pairs to train an algorithm to predict a clearer version of an image. Additionally, the high correlation of neighboring pixels in natural images is often utilized by image-enhancing algorithms. One of the most successful algorithms for image super-resolution is belief propogation where an image's high-resolution equivalent is treated like a Markov network [9].

Finally, support vector regression has been used for super resolution applications, most notably by Ni et al. [4]. Like our approach, multiple support vector machines are trained on low- and high-resolution image pairs, but the algorithm was applied to mainly natural images. Support vector regression with a Gaussian kernel was shown to outperform polynomial kernels on natural images. It was also noted that in regions of natural images near sharp edges the Gaussian kernels performed especially poorly. Support vector regression was also applied in the discrete cosine domain with marginally better results.

3 Background

3.1 Support Vector Regression

Support vector regression is a technique developed by Vapnik et al. [6] in 1997 which uses a similar formulation as support vector classification except it is used to find an approximating function to a set of data. SVR uses an ϵ-insensitive loss function, meaning at each data point $\mathbf{x}_i$ there is ϵ error allowed without penalty.

$$\epsilon\text{-insensitive loss function: } |y - f(x)|_\epsilon = max\{0, |y - f(x)| - \epsilon\}$$

The first form of support vector regression presented here is known as ϵ-SVR, whose definition follows. Given a data set of the form $(x_1, y_1), \ldots, (x_l, y_n)$, where $x_i \in \mathbb{R}^N$ is a feature vector, $y_i \in \mathbb{R}$ is the corresponding output, $C > 0$ and $\epsilon > 0$ are parameters, we can write our objective function as follows

$$\begin{aligned} \min_{\mathbf{w},\xi_i,\xi_i^*,\rho,b} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C(\sum_{i=1}^{n}(\xi_i^* + \xi)) \\ \text{such that} \quad & \mathbf{w}^T\phi(\mathbf{x}_i) + b - y_i \leq \epsilon + \xi_i, \\ & y_i - \mathbf{w}^T\phi(\mathbf{x}_i) - b \leq \epsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \geq 0, i = 1, \ldots, n. \end{aligned}$$

Where $\phi(\mathbf{x}_i)$ is a function that maps from the space of $\mathbf{x}_i$ to a higher dimensional feature space, and $\xi_i, \xi_i^*, i = 1, \ldots, n$ are positive and negative slack variables respectively that allow for points in the data set to lie outside of the "ϵ-tube", the volume within a distance of ϵ of the approximating function, formed in the feature space.

The dual is

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \quad & \frac{1}{2}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*)^T Q(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*) + \epsilon\sum_{i=1}^{n}(\alpha_i+\alpha_i^*) + \sum_{i=1}^{n} y_i(\alpha_i-\alpha_i^*) \\ \text{such that} \quad & \mathbf{e}^T(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*) = 0 \\ & 0 \le \alpha_i, \alpha_i^* \le C, i = 1,\ldots,n. \\ \text{where} \quad & \boldsymbol{\alpha} = [\alpha_1,\ldots,\alpha_n], \boldsymbol{\alpha}^* = [\alpha_1^*,\ldots,\alpha_n^*] \end{aligned}$$

where $Q_{ij} = K(\mathbf{x}_i,\mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ and $\mathbf{e} = [1\,1\cdots1]^T$, the vector of all ones. By solving the dual (using KKT conditions, as usual), we obtain our approximate function

$$\sum_{i=1}^{n}(-\alpha_i+\alpha_i^*)K(\mathbf{x}_i,\mathbf{x}) + b.$$

We can call C the cost, as it represents the magnitude of the penalty we apply for having a support vector outside the "ϵ-tube". ϵ is also a fitting variable name as it defines the error within which the resulting approximating function estimates the data. Our project makes use of a relatively new support-vector algorithm called ν-SVR, which similar to ϵ-SVR except it has the advantage that it automatically minimizes ϵ. Instead of specifying an epsilon, we choose a parameter $\nu \in [0,1]$ which controls the number of support vectors. Schölkopf, Smola et al. [10] prove that ν is an upper bound on the fraction of errors and a lower bound on the fraction of support vectors. Accordingly, ν is called a "hyperparameter" Therefore, with C and ν as parameters, ν-SVR is defined by

$$\begin{aligned} \min_{\mathbf{w},\xi_i,\xi_i^*,\rho,b} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C(\nu\epsilon + \frac{1}{n}\sum_{i=1}^{n}(\xi_i^*+\xi)) \quad \text{(eq. 1)} \\ \text{such that} \quad & \mathbf{w}^T\phi(\mathbf{x}_i) + b - y_i \le \epsilon + \xi_i, \\ & y_i - \mathbf{w}^T\phi(\mathbf{x}_i) - b \le \epsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \ge 0, i = 1,\ldots,n,\ \epsilon \ge 0. \end{aligned}$$

The dual is

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \quad & \frac{1}{2}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*)^T Q(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*) + \mathbf{y}^T(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*) \\ \text{such that} \quad & \mathbf{e}^T(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*) = 0,\ \mathbf{e}^T(\boldsymbol{\alpha}+\boldsymbol{\alpha}^*) \le C\nu, \\ & 0 \le \alpha_i, \alpha_i^* \le C/n, i = 1,\ldots,n. \end{aligned}$$

By solving the dual, we obtain our approximate function

$$\sum_{i=1}^{n}(-\alpha_i+\alpha_i^*)K(\mathbf{x}_i,\mathbf{x}) + b.$$

Note that in ν-SVR, α and α^* are bounded by C/n and not just C. For the rest of this paper, we will mean C/n whenever we use C, since we will only be using ν-SVR. For our purposes, we trained several ν-SVM's to find multiple such approximation functions that allowed us to estimate high-resolution images based on blurry, low-resolution ones.

4 Methodology

4.1 Overview

Our algorithm for deblurring and super resolving an image is based on the one outlined in [4]. Given a blurred, low-resolution image of text, our goal is to create a higher-resolution image where the text has defined edges and is more comprehensible to both humans and OCR software. More precisely, we have as input an image $\mathbf{x} \in \mathbb{R}^n$, and we wish to find an estimate of a higher-resolution image, $\mathbf{y} \in \mathbb{R}^m$ where $m = 4^r n,\ r \in \mathbb{N}$, that contains clearly rendered text and whose blurred and downsampled self could, with high probability, generate $\mathbf{x}$.

In this project, we only deal with the case of creating an estimate of an image with 2x the resolution of a low-resolution input image. We do this by using a $N \times N$ block of a low-resolution image to generate an estimate of 4 pixels of a high-resolution image corresponding to the center of the $N \times N$ low-resolution patch, i.e. $m = 4n$ (Figure 1).

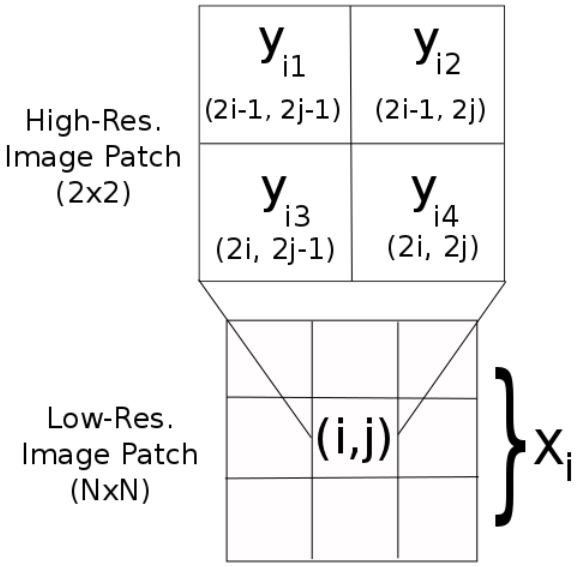

Figure 1: Data and labels for each of the SVM's. Note the indices in the top patch refer to the high-resolution image and the central index in the bottom patch refers to the low-resolution image.

Note that there is a lot of overlapping of the x_i's, and no overlap of the y_{ip}'s.

Figure 3: Low-resolution images used for training the dataset. These images all have had a Gaussian blur with $\sigma^2 = 20$ with filter size 61 x 61 applied before downsampling.

4.2 Training Data Generation

A training set consists of I image pairs, $(I_{lk}, I_{hk})_{k=1}^{n}$, where I_{lk} is the k^{th} low-resolution image and I_{hk} is the k^{th} high-resolution image. These must be segmented into smaller low-resolution input blocks, $x_i \in \mathbb{R}^{N \times N}$ which have labels $y_{ip},\ p \in \{1, \ldots, 4\}$ (Figure 1). Each of the labels y_{ip}, $i \in 1, \ldots, n$ correspond to the labels SVM p trains on.

To produce the high resolution elements of the training set, 26 letters were first generated in vector format and then rendered so that each letter was approximately 200 pixels along its largest axis. The letters were all generated in the font face Helvetica, which is not a fixed width font, so the images had to be padded to make a square 200 x 200 pixel image with a letter at its center. These images were stored as grayscale tiff images to retain all the data from the image that otherwise would be lost in lossy image formats. A representative sample of high resolution images are shown in figure 2.

Figure 2: High-resolution images used as labels. Each of these images is 200 x 200 pixels and contains a single lowercase helvetica letter.

To produce the 100 x 100 pixel low-resolution images, a 2 dimensional Gaussian filter was applied to the the original 200 x 200 pixel label data set. The σ^2 of the Gaussian for the filter was varied between 20 and 45 in 5 pixel increments so that the bounds of our method could be adequately tested across a varied field of blurriness. After being blurred, the images were downsampled by averaging non-overlapping blocks of 2 x 2 pixels to form a single pixel value. The result of this process was a set of 100 x 100 images that varied considerably in readability and blurriness. A sample of the downsampled images is show in figure 3.

4.3 Prediction via Support Vector Regression

The goal of support vector regression (SVR) is to approximate a function which, in our case, is an interpolating function f that will take a N x N block of a low-resolution image and produce an estimate for a pixel in the center of a high-resolution version of this block (Figure

1). With a set of high- and low-resolution image pairs, a training set was constructed by taking N x N patches (our x_i's) from a low-resolution images, vectorizing them, and assigning the vector four values (our y_{ip}'s) corresponding to the pixel intesity of the four center pixels in the high-resolution image.

We then pass this training data to a SVM software package, LIBSVM, via MATLAB, and it performs the optimization necessary to obtain the required values of $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$.

5 Experimentation

The error metrics we used to determine the quality of a regression are MSE and PSNR defined thus

$$MSE = \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}[I_{predicted}(i,j) - I_{actual}(i,j)]^2$$

$$PSNR = 20 \cdot log_{10}(\frac{MAX_I}{\sqrt{MSE}})$$

As put forth in [8], PSNR is a valid and good metric for comparing optimisation, denoising, etc. when the content is "fixed." We meet the "fixed" criteria as all the data generated is formed, centered, and blurred in the same manner, yet determining the quality of edges was still done visually. In fact, the PSNR was best-suited for comparison of super-resolved images, and did not provide a good absolute measure of image quality.

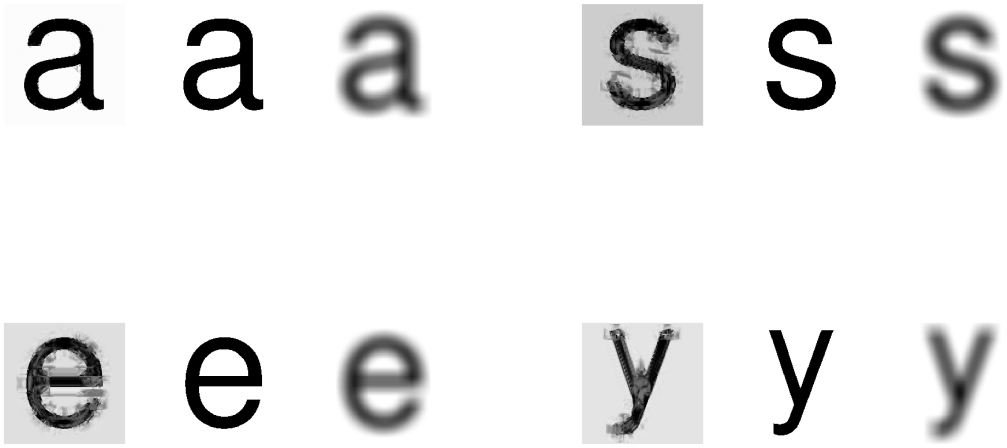

Figure 4: Examples of, from left to right, the reconstructed image, the target image and the starting image. The four SVM's used to reconstruct these images were trained on the letters g, i, v, and a.

The example predicted images (figure 4) above were generated with SVM's trained on images of the letters g, i, v, and a. We chose these four letters because we felt they capture a lot of the characteristics of the font Helvetica. All the letters were blurred with a Gaussian kernel of variance 30px. A few trials revealed that when training and predicting letters with very little blur (blurred with Gaussian kernels of variance of 5px), we consistantly got worse image enhancement than with a larger blur like with Gaussians of variance 30px. For this regression, we trained our four SVM's using the Gaussian kernel because of our goal of verifying its ability to preserve sharp edges and to deblur. We defined the parameters of our SVR objective function (eq. 1) as C = 150, $\nu = .5$, and the parameter of the Gaussian kernel, $\sigma^2 = \frac{25}{2}$. We now define γ as $\frac{1}{2\sigma^2}$, as we use this same definition of γ below.

Based on figure 4 the reconstruction of 'a' is nearly perfect; this is logical since it was part of our training set (in fact our training error is very low for all of 'g', 'i', 'v', and 'a'). As for the other letters, we have reasonable reconstructions; specifically, the edges were mostly recreated very well. In previous work it was noted how difficult it was to preserve edges with the Gaussian kernel, and diagonal edges most of all [4]. We seem to have partly overcome that problem as clearly shown in our recontsructed 'y'.

However, there are a few shortcomings of the reconstruction. Specifically, we see that there are strange grey artifacts surrounding the letters, which we suspect are derived from overfitting. After trying side lengths 3, 5, 7, 9, 11, we found that we acheived the best PSNR and had the best-looking edges when we used 3×3 patches. This result was counter-intuitive to our group as we expected having more information about a local region would provide a better fit. Instead, we saw more gray artifacts as a result of using patches larger than 3. Furthermore, there seems to be a few outliers in the predictions as normalization of the output of the regression results in a grey background. This implies that one output of a regression resulted in a value much greater than the others and has made the normally white background be grey after normalization. Additionally, the 'v' portion of the y is poorly reconstructed even though v was one of the letters we trained on and it has a similar structure. This provides additional evidence that our method is susceptible to overfitting. Using so few letter images means our model remained relatively small, and thus the algorithm's performance is impressive. Next we performed an analysis on the effect of the three parameters, C, γ, and ν on the algorithm's ability to enhance the blurry low-resolution images.

5.1 Parameter Search

A variety of parameters were used to train the SVR's in an attempt to quantify their effect on the resultant super-resolved images; each combination of parameters is represented in figure 5. In this section all our testing consisted of training on a single low- and high-resolution image pair representing a 'c' whose low-resolution image had been blurred with a Gaussian filter of variance 30 pixels applied to it. We then applied the four trained SVM's to predict super-resolved versions of low-resolution images of 'c', 'o' and 'w', all of which had the same level of blurring as the 'c' we trained on. Tesing on 'c' logically results in very good reconstruction. We chose 'o' as it is similar to 'c' and therfore we expected reasonably good reconstruction, on par with what 5 shows. Finally, we chose 'w' because it does not resemble 'c' as it is all diagonal edges with no curves. As expected, predicting on 'w' produced the worst results.

Through experimentation, we found a few fairly obvious things to be true; training on more letters improved the performance of our approximating function, and changing ν, which, recall, is an upper bound on the percentage of support vectors, does little to affect the resulting PSNR. The exception being when ν drops below .1 and limits the number of support vectors severely enough that performance worsens considerably. Also, training time goes down very quickly with decreasing ν, since we optimize over fewer support vectors. Changing γ had very little effect on the PSNR, and althogh a smaller γ sometimes produced a slightly improved PSNR, the image itself had less defined edges.

The parameter search we conducted was rather inconclusive as the kind of useful results we were looking for were only acheived when using a parameter $C \geq 100$, and thus we were only able to confirm that a high cost is necessary to ensure good-quality enhancement of text images. Conducting a useful parameter search, however, would take a very long time as when C exceeds 100, training time increases considerably. Additionally, PSNR has proven to be so ineffective at predicting image quality that it would be easier at this point to simply examine each image visually and exaluate the quality of its reconstruction. A better metric for judging image quality would be invaluable to our project.

5.2 Conclusion

The above results show that given a synthetic training set consisting of raster images of letters of a single font, SVR is a reasonable algorithm for removing even considerable blurring. It is clear from Figure 4 that given a minimal amount of post-processing, e.g. a median or mode filter and some further normalization, we could acheive very accurate super-resolution.

Thus, we have demonstrated that SVR using a Gaussian kernel can reconstruct sharp edges and effectively deblur images. A future direction that would be interesting to investigate would be to develop an algorithm capable of accounting for and rejecting interference from adjacent letters, for example, if there were letters close to one another in a word. Additionally, there is a potentially large increase in performance to be had if estimation between the four SVM's used in this paper could be made jointly. For joint estimations, some methods being considered currently by the super-resolution community are vector-valued or operator-valued kernel functions that could produce the multiple outputs that we would need for reconstruction with a single SVM.

	$\nu = 0.1$					
	$\gamma = 0.04$			$\gamma = 0.2$		
	C	O	W	C	O	W
$C = 1$	34.7708	34.6718	30.0509	36.4388	36.5512	30.2991
$C = 5$	34.9724	34.8854	30.0778	36.1413	35.1897	30.4198
$C = 10$	35.1707	35.0904	30.1056	35.8534	34.8587	30.4565
$C = 100$	35.8784	35.8385	30.2165	35.7282	34.6249	30.2231

(a) Table of PSNR data for several values of cost on two letters, with $\nu = 0.1$

	$\nu = 0.5$					
	$\gamma = 0.04$			$\gamma = 0.2$		
	C	O	W	C	O	W
$C = 1$	33.6653	33.4864	29.9225	35.1901	34.6718	30.1106
$C = 5$	33.7692	33.5868	29.9328	35.0016	34.1782	30.0884
$C = 10$	33.9006	33.704	29.9412	35.012	34.1712	30.0914
$C = 100$	34.9715	34.6559	30.0726	36.1848	36.1885	30.2231

(b) Table of PSNR data for several values of cost on two letters, with $\nu = 0.5$

	$\nu = 0.9$					
	$\gamma = 0.04$			$\gamma = 0.2$		
	C	O	W	C	O	W
$C = 1$	33.6498	33.4708	29.9225	35.1644	34.5977	30.1068
$C = 5$	33.6766	33.4917	29.924	35.0043	34.2803	30.0884
$C = 10$	33.7597	33.5639	29.9297	35.0048	34.1635	30.0906
$C = 100$	34.9699	34.6565	30.0716	36.1794	36.1823	30.2224

(c) Table of PSNR data for several values of cost on two letters, with $\nu = 0.9$

Figure 5: Tables show the PSNR data across all runs. For comparison the original PSNR between the blurry images and original high resolution ones are as follows: C 13.5795, O 15.1732, and W 11.4824.

6 Individual Contributions

Sean wrote code to break down image pairs into a format suitable for training our SVM's in MATLAB, and he generated the data to compare the PSNR of predicted images given different SVR parameters. Roy mainly developed and debugged the code for SVR in MATLAB. Ross created the training set for the SVR. Roy and Ross wrote up the final report.

7 References

[1] Dalong Li and Steven Simske. Example Based Single-frame Image Super-resolution by Support Vector Regression. JPRR 2010

[2] Daniel Glasner, Shai Bagon, and Michal Irani. Super-Resolution from a Single Image. ICCV 2009

[3] David Capel and Andrew Zisserman. Super-resolution Enhancement of Text Image Sequences.

[4] Karl S. Ni, Sanjeev Kumar, Nuno Vasconcelos, Truong Q. Nguyen. Single Image Superresolution based on Support Vector Regression. In ICASSP 2006

[5] Chang, Chih-Chung, Lin, Chih-Jen. LIBSVM: A library for support vector machines. In ACM Transactions on Intelligent Systems and Technology 2011

[6] Harris Drucker, Chris J.C. Burges, Linda Kaufman, Alex Smola and Vladimir Vapnik (1997). "Support Vector Regression Machines". Advances in Neural Information Processing Systems 9, NIPS 1996, 155161, MIT Press.

[7] Karl S. Ni, Truong Q. Nguyen. Image Superresolution Using Support Vector Regression. In IEEE Transactions on Image Processing 2007

[8] Q. Huynh-Thu and M. Ghanbari. Scope of Validity of PSNR in Image/Video Quality Assessment. Electronic Letters 2008

[9] William T. Freeman, Thouis R. Jones, and Egon C. Paszto. Example-Based Super-Resolution. 2009

[10] Bernhard Schölkopf and Alex J. Smola et al. New Support Vector Algorithms. 2007