Image Inpainting Using Ridge vs. Lasso Algorithm

Azza Iqbal
Tingcen Dong

INTRODUCTION

Image Inpainting is the process of reconstructing missing regions in an image. It is an important problem in computer vision and an essential functionality in many imaging and graphics applications, e.g. object removal, image/video restoration, manipulation, re-targeting, compositing, and image-based rendering. Further applications include text/object removal, texture synthesis, and transmission error concealment[8]. In this process, we essentially start with a corrupted or masked input image and define which pixels are missing or invalid that need to be filled, and also the remaining pixels as the truth or valid pixels that would be used to help fill in the missing pixels.

This form of representation is defined as *sparse representation* which assumes that signals can be described as a linear combination of a few atoms from a predefined *dictionary*. In this application, it assumes that what was described earlier: missing pixels in an image can be defined by the pre-existing ones for image inpainting and reconstruction. This assumption is a suitable one to make because natural images existing all around us have a lot of redundancy and can be represented by a smaller set of basic vectors. Since in sparse representation, most entries are simply zero, we only have to store the non-zero elements, which leads to more memory and computation efficiency. [2]

The core idea behind tackling this problem is to first search for the most similar image 'patches' from the existing pixels from the image itself (or other images within the dataset) and directly paste the patches onto the missing parts. This kind of primitive search algorithm can be very inefficient and time consuming, so several deep learning approaches have been developed to improve on this basic idea.[3]

The approaches are based on convolutional neural deep learning networks which use different loss functions to guide the algorithm towards the optimal solution. The loss functions can include L1 based, L2 based, Adversarial loss, texture loss, as well as combinations of several loss functions. The paper which we explore uses L1 regularization in images with sparse representations. L1 was used in this case because this regularization method penalizes high signals in a way that they will be set to zero, and that entire feature is removed from the model. That leads to the resulting signal having a large number of 0 entries. This does not happen with L2 regularization as it penalizes $weight^2$, and the derivative is 2 * weight, while in L1, it penalizes $|weight|$ and the derivative is a constant whose value is independent of weight. Thus, in L2, it removes a percentage of the weight every time, but never actually gets to 0 [7]. Therefore, it is understandable why L1 was used in sparse representations of image inpainting to encourage 0 values. However, this is a very computationally expensive operation, so other regularization methods should be explored, such as the L2 regularization, or Ridge algorithm.

LITERATURE REVIEW

Most approaches, like the one used with sparse representation in the research paper we are referencing, use the *Lasso* algorithm which uses the L1 norm. The Lasso algorithm stands for Least Absolute Shrinkage and Selection operator and it adds a penalty to the cost function in order to reduce loss. We can use it as a way to efficiently solve L1 minimization to reconstruct the corrupted image pixels. However, there are some limitations of this algorithm. Firstly, it does not work as well if the number of predictors (p) is greater than the number of observations (n), since it will only pick at most 'n' number of predictors and set the rest to zero even if they were all relevant. Moreover, if there are two or more highly collinear variables then Lasso regression selects one of them randomly, which is not good for image imprinting.

One famously established L1 norm regularization approach is the Total Variation (TV) approach, which performs edge-preserving image restoration, but at a high computational cost. TV regularization requires linearization of a highly nonlinear penalty term, which increases the restoration time considerably for large scale images. [6]

Our proposal is to implement a different algorithm in place of Lasso: the *Ridge* algorithm which adds a penalty to the L2 norm rather than L1. The L1 regularization follows the basic formulation of:

$$\min \ ||Ax - d||_2^2 \ + \lambda ||Cx||_1$$

while the L2 regularization follows:

$$\min ||Ax - d||_2^2 \ + \lambda ||Cx||_2$$

Both formulations will lead to a convex optimization problem which would allow us to converge to a globally optimal solution. Therefore, both are options that can be used in image inpainting to increase robustness against collinearity of ordinary least squares regression. Both algorithms attempt to minimize a cost function.

However, there are several differences between the two. The main intuitive difference between the L1 and L2 regularization is that L1 regularization tries to estimate the median of the data while the L2 regularization tries to estimate the mean of the data to avoid overfitting [4]. This leads to a tradeoff of higher bias for lower variance. Moreover, L1 regularization helps in feature selection by eliminating the features that it declares as not important, and dropping variables associated with coefficients that go to zero. But this may cause us to lose some important features that were incorrectly identified as unimportant. L2, on the other hand, is useful when you have collinear/codependent features. In the case of image reconstruction, we do not wish to risk misclassifying the few pixels we have available as unimportant and lose them through feature extraction.

The most important difference is that L1 regularization is computationally more expensive, because it cannot be solved in terms of matrix math and does not lead to a closed-form or linear solution. To solve the L1 formulation, we typically need to start iterating using methods like steepest descent, conjugate gradients, or other descent-based methods and see where we end up[6]. L1 is much more computationally cheaper as the Thus, we will explore replacing the Lasso algorithm with the Ridge algorithm for image inpainting, and observe if the Ridge algorithm will help retrieve similar results which lower computational cost.

PROPOSED SOLUTION

In the paper we are referencing, image inpainting was performed using the Lasso algorithm in the following formulation:

$$\hat{x} = argmin_x \left||y - Dx|\right|_2^2 + \beta \left||x|\right|_1$$

The L1 norm next to the regularization coefficient β encourages sparsity while β itself controls the tradeoff between this sparsity and the reconstruction error as we try to minimize it.
The letter D represents the *dictionary*. There are many ways to carry out dictionary learning, which is what will define how the empty pixels are filled. In fact, the whole success of inpainting lies on how well it infers the missing pixels from the observed pixels which make up the dictionary. In our case, we will use this source image minus the target region to construct the dictionary.

In our proposed solution, we will change the algorithm to work with the Ridge algorithm which uses the L2 norm, which has much less sparsity but has lower computation cost:

$$\hat{x} = argmin_x \left||y - Dx|\right|_2^2 + \beta \left||x|\right|_2$$

For the comparative study, we will carry out both algorithms for image inpainting on 3 different images. We will remove a target section of each image, which will leave our image with a collection of invalid/empty pixels which need to be filled ie, impainted using the algorithm.

All the pixels in our image have an RGB color value; which means that there will be a value for the red, green, and blue hues depending on the final color of that pixel. The input to the algorithm will be our source image, as well as an additional image where the target is colored red (R:256, G:0, B:0) and the background is black (R:0, G:0, B:0).

Our goal is to first remove the target region from the source image and be left with only the source region of the inputted image minus the target region. The target region will leave a hole in the image which will have our corrupted components which we have predefined. If we denote those pixels by I, where $I = \{i \mid e_i \neq 0\}$ then y\I denotes the signal we obtain after removing the corrupted pixels I.
Moreover, D\I will be the reduced dictionary matrix after removing the corrupted pixels. Hence, we can rewrite the equation as:

$$\hat{x} = argmin_x \left||y\backslash I - D\backslash_I x|\right|_2^2 + \beta \left||x|\right|_2$$

Next, the image inpainting algorithm will fill in those empty pixels with new ‘reshaped’ RGB values which would give us the overall output image as a smooth realistic image with the target image removed. Two different algorithms will be tried in this process, first with the Lasso Algorithm which uses L1, and then with the Ridge Algorithm which uses L2.

The *sparsity* of the image will need to be adjusted differently for the two algorithms, as mentioned earlier since the L1 algorithm encourages sparsity on its own, but the L2 algorithm does not. Therefore, we will set the sparsity higher for the Ridge algorithm than in the Lasso to have a meaningful comparative analysis.

Thus, the corrupted image (after removing the target from the inputted image) can be recovered with these two simple rules:

$$\hat{y} = \begin{cases} y_i, & if\ i\ \notin I \\ (D\hat{x})_i, & if\ i\ \in I \end{cases}$$

The equation above tells us that if a pixel is not within the corrupted pixels then it can be displayed with its existing RGB values. If it is within the corrupted pixels, it will have to be translated using the dictionary to an updated RGB value, and ultimately into a reconstructed pixel.

An overview of the algorithm is shown below:

Algorithm 1: Image Inpainting via Lasso/Ridge

Input: source image, target region
Dictionary Construction: formed by sampling random patches from the source region of the image (source image minus target region)
Recovery of corrupted pixels in target region:

1. Iterate through each pixel index
2. Slide the window for each pixel with the data dictionary and reshape the R, G, and B of each pixel through the LASSO or Ridge algorithm using CVX computation.
3. Reconstruct the RGB values for each patch with the Alpha RGB computed by LASSO/Ridge.

Plot the reconstructed image

EXPERIMENTAL RESULTS

To test our algorithm, we have constructed two codes on Matlab to implement the Lasso and Ridge Functions through the CVX Tool. Code 1 implements the Lasso Algorithm and Code 2 implements the Ridge Algorithm as shown in the Appendix. Both codes will be carrying out image inpainting on 3 different inputted images. Due to the limitation on our device's computation speed, we used fairly small image sizes in order to finish running the program in an acceptable amount of time. We expect that the comparative result will be the same if recreated on a larger scale.

The first case involves an 80x64 picture of a foggy landscape with a person standing in the background as shown in figure 1. This picture has very minimal details, but very blurry features without sharp edges. The person will be our target that needs to be removed from the picture. To do so, we will input an image of the target region with the shape of the person colored in red as shown in figure 2. Both codes are run, and the result of image imprinting with the Lasso Algorithm is shown in figure 3, while the Ridge Algorithm results are shown in figure 4.

Figure 1. Case 1 Input Image

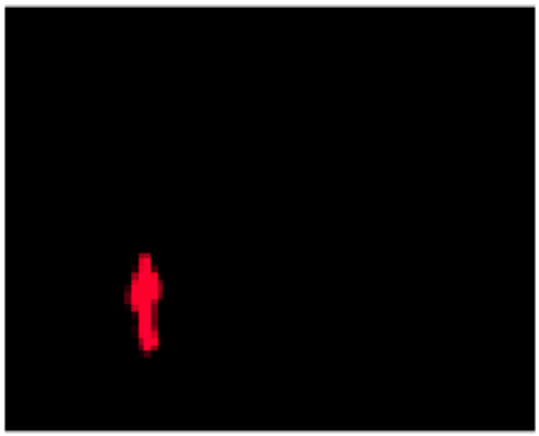

Figure 2. Case 1: Target Region

Figure 3. Result with L1 (sparsity of 10)

Figure 4. Result with L2 (sparsity of 30)

As we can see from figures 3 and 4, both results with L1 and L2 achieved were very similar, without any noticeable depreciation of quality from using the L2 algorithm.

We will now repeat the analysis on a larger picture, of size 128 x 160 with a decent amount of detail in it. It is a natural photo taken of a cat on the floor, where the cat is the target to be removed from the image, and then the image is to be restored.

Figure 5. Case 2 Input Image

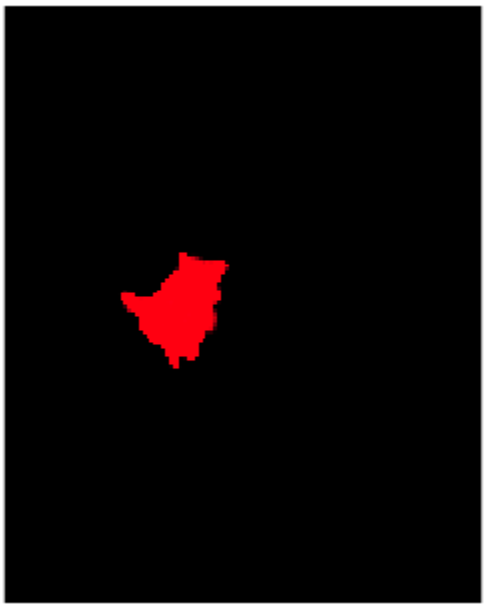

Figure 6. Case 2 Target Region

Figure 7. Result with L1 (sparsity of 50)

Figure 8. Result with L2 (sparsity of 100)

As before, the input includes the original image as shown in figure 5 and the target region displayed in figure 6. The results from the L1 Lasso Algorithm are shown in figure 7 and the L2 Ridge Algorithm results are shown in figure 8. Once again, we do not see a significant difference between the L1 and L2 results.

Our third and final case is an extremely detailed picture of a bunch of flowers, of size 160x80 as shown in figure 9. We choose the target to be two specific flowers from the bunch which are shown in figure 10.

Figure 9. Case 3 Input Image

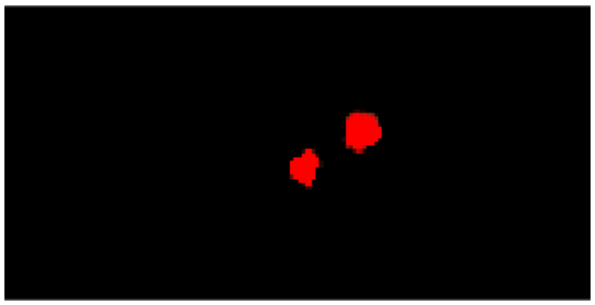

Figure 10. Case 2 Target Region

Figure 11. Result with L1 (sparsity of 50)

Figure 12. Result with L2 (sparsity of 100)

As we can see from the L1 results in figure L1 and the L2 results in figure 12, there is not much difference between the impainting results. A miniscule improvement can actually be seen in the impainting of L2 where the result looks more natural than in the L1 result.

Note that the sparsity in all three images was set higher for the L2 algorithm than the L1 because L1 naturally comes pre-existing with a lot of sparsity.

We also took note of the computational time required with different values of the sparsity and recorded them below in Table 1, 2 and 3 for each image. The parameters values which gave the best inpainting results for whose results we displayed above are in bold font.

Algorithm	Time (s)	Sparsity
L1	64.05	5
	66.72	**10**
	58.85	15
	58.85	30
L2	55.41	1
	56.29	10
	60.98	20
	59.62	**30**

Table 1. Computation Time L1 vs L2 for Case Image 1

Algorithm	Time (s)	Sparsity
L1	556.70	10
	553.56	20
	556.35	30
	547.83	40
	577.44	**50**
L2	513.35	30
	515.86	50
	521.51	**100**

Table 2. Computation Time L1 vs L2 for Case Image 1

Algorithm	Time (s)	Sparsity
L1	320.07	20
	323.99	30
	329.60	**50**
L2	308.81	30
	310.02	50
	312.99	80
	319.23	**100**

Table 3. Computation Time L1 vs L2 for Case Image 1

In the case of larger pictures, there is an obvious increase in time as we increase sparsity, which in turn increases the quality of the inpating. In all cases, the time taken in the L2 algorithm is less than L1.

CONCLUSIONS

The results from implementing the L1 Lasso and L2 Ridge algorithm on three different images with varying levels of intricacy show us there was no visually significant difference in the quality of the image restoration. Our experiment was to check if the L2 algorithm can successfully reproduce results as well as the more commonly used L1 algorithm in the application of Image Inpainting, and we were successful in proving so.

More importantly, the reason we wanted to prove this is because L2 is known to be computationally less expensive than the L1 algorithm. We saw from Table 1, 2 and 3 that indeed the L2 algorithm took less time than the L1 algorithm did. When looking at the best inpainting conditions, we saw a decrease of 7 seconds in case 1, 56 seconds in case 2 and 10 seconds in case 3. As we were only testing on small images for this report, the change may not seem so significant. However, when reproducing this result on larger scale images, we expect to see a larger decrease in computation time with the L2 algorithm when compared to L1. We do not anticipate any loss of quality when doing so, as our images did not show any evidence of diminishing image restoration. As a further exploration of this study, further images that are even more complicated and larger can be tested to make sure the same results are achieved.

The significance of this solution is that image inpainting is a common procedure in many applications in image processing around us. Computational time and expense to be saved is very vital in implementing meaningful and usable solutions as optimization is all about getting the solution in the most efficient way possible.

REFERENCES

1. "Image Inpainting," PaperswithCode. [Online]https://paperswithcode.com/task/image-inpainting. (Accessed April 17, 2022)
2. A. Kumar, S. Kataria, "Dictionary Learning Based Applications in Image Processing using Convex Optimisation," Dept. Electrical Engineering, Indian Institute of Technology, Kanpur, 2016. [Online]https://www.semanticscholar.org/paper/Dictionary-Learning-Based-Applications-in-Image-Kumar-Kataria/39c94a0a4f21fdc7f4855b2c5cc24342c29940a2
3. C. Li. "10 Papers You Must Read for Deep Image Inpainting." Towards Data Science. [Online]https://towardsdatascience.com/10-papers-you-must-read-for-deep-image-inpainting-2e41c589ced0. (Accessed April 17, 2022)
4. D. Taunk. "L1 vs L2 Regularization: The Intuitive Difference." Analytics Vidhya. [Online]https://medium.com/analytics-vidhya/l1-vs-l2-regularization-which-is-better-d01068e6658c. (Accessed April 17, 2022)
5. K. Pykes. "Fighting Overfitting with L1 or L2 Regularization: Which One Is Better?" Neptune Blog. [Online]https://neptune.ai/blog/fighting-overfitting-with-l1-or-l2-regularization#:~:text=The%20differences%20between%20L1%20and,regularization%20solution%20is%20non%2Dsparse. (Accessed April 17, 2022)
6. "Image Reconstruction as Optimization: L1 Formulations and Compressed Sensing." Univ. of Wisconsin-Madison. [Online]https://qiml.radiology.wisc.edu/wp-content/uploads/sites/760/2021/03/lecture_B1_05_l1recon.pdf
7. "Regularization for Sparsity: L1 Regularization." Machine Learning Crash Course. [Online]https://developers.google.com/machine-learning/crash-course/regularization-for-sparsity/l1-regularization
8. D. Ding. "Image Inpainting Based on Exemplars and Sparse Representation." Univ. of Arizona. 2017. [Online]https://repository.arizona.edu/handle/10150/625897
9. Adapted from https://new.qq.com/omn/20210825/20210825A0DBG000.html
10. By Tingcen Dong, 2022
11. By Caisheng Dong, 2022