8-23-2019

Numerical Algorithms for Solving Nonsmooth Optimization Problems and Applications to Image Reconstructions

Karina Rodriguez
Portland State University

Numerical Algorithms for Solving Nonsmooth Optimization Problems and Applications to Image Reconstructions

Nguyen Mau Nam[1], Lewis Hicks[2], Karina Rodriguez[3], Mike Wells[4],

Abstract. In this project, we apply nonconvex optimization techniques to study the problems of image recovery and dictionary learning. The main focus is on reconstructing a digital image in which several pixels are lost and/or corrupted by Gaussian noise. We solve the problem using an optimization model involving a sparsity-inducing regularization represented as a difference of two convex functions. Then we apply different optimization techniques for minimizing differences of convex functions to tackle the research problem.

1 Introduction

Convex optimization has been strongly developed since the 1960s, providing minimization techniques to solve many real-world problems. However, a challenge in modern optimization is to go from convexity to nonconvexity as nonconvex optimization problems appear frequently in many applications. This is the motivation for the search for new optimization methods to deal with broader classes of functions and sets where convexity is not assumed. One of the most successful approaches to go beyond convexity is to consider the class of DC (difference of convex) functions. Given a linear space X and two convex functions $g, h : X \to \mathbb{R}$, a DC optimization program minimizes $f = g - h$. It was recognized early by P. Hartman [7] that the class of DC functions exhibits many convenient algebraic properties. This class of functions is closed under many operations usually considered in optimization. In particular, it is closed with respect to taking linear combinations, maxima, and finite products of DC functions. Another nice feature of DC programming is that it possesses a very nice duality theory; see [16] and the references therein. Generalized differential properties of DC functions were investigated by Hirriart Urruty in [8] with some recent generalizations in [13].

Although the role of DC functions has been known earlier in optimization theory, the first algorithmic approach was developed by Pham Dinh Tao in 1985. The algorithm introduced by Pham Dinh Tao for minimizing $f = g - h$, called the DCA, is based on subgradients of the function h and subgradients of the Fenchel conjugate of the function g. This algorithm is summarized as follows: with given $x_1 \in \mathbb{R}^n$, define $y_k \in \partial h(x_k)$ and $x_{k+1} \in \partial g^*(y_k)$. Under suitable conditions on the DC decomposition of the function f, the two sequences $\{x_k\}$ and $\{y_k\}$ in the DCA satisfy the monotonicity conditions in the sense that $\{g(x_k) - h(x_k)\}$ and

[1]Fariborz Maseeh Department of Mathematics and Statistics, Portland State University, Portland, OR 97207, USA (mnn3@pdx.edu). Research of this author was partly supported by the USA National Science Foundation under grant DMS-1716057.

[2]Fariborz Maseeh Department of Mathematics and Statistics, Portland State University, Portland, OR 97207, USA

[3]Fariborz Maseeh Department of Mathematics and Statistics, Portland State University, Portland, OR 97207, USA

[4]Fariborz Maseeh Department of Mathematics and Statistics, Portland State University, Portland, OR 97207, USA

$\{h^*(y_k) - g^*(y_k)\}$ are both decreasing. In addition, the sequences $\{x_k\}$ and $\{y_k\}$ converge to *critical points* of the primal function $g - h$ and the dual function $h^* - g^*$, respectively; see [2, 16, 17] and the references therein. The DCA is an effective algorithm for solving many nonconvex optimization problems without requiring the differentiability of the data. However, to deal with optimization problems of large scale, it is necessary to develop new optimization techniques to accelerate the convergence rate of this algorithm.

In this project, we focus on applications of nonconvex optimization techniques to the problems of image reconstructions and dictionary learning. In particular, we develop new acceleration techniques for the DCA and apply them to the image reconstruction problem. A digital (black and white) image M is represented by an $N_1 \times N_2$ matrix in which each entry contains a numerical value (of bit depth 8) of each pixel of the image. The main focus is on reconstructing a digital image in which several pixels are lost and/or corrupted by Gaussian noise. After the image is corrupted by a linear sampling operator A and distorted by some noise ξ, we observe only the image $b = A(M) + \xi$, and seek to recover the true image M.

Sampled image (SR=50%)

Recovered Image

A vector is referred to as sparse when many of its entries are zeros. An image $x \in \mathbb{R}^n$ (in vectorized form) is said to have a sparse representation y under D if there is some $n \times K$ matrix D, known as a dictionary, and a vector $y \in \mathbb{R}^K$ such that $x = Dy$. In this case, the dictionary D maps a sparse vector to a full image. The columns of D are called *atoms*, and given a suitable dictionary in this model, theoretically any image can be built from a linear combination of the columns (atoms) of the dictionary. Using a clever choice of dictionary allows us to work with sparse vectors, thereby reducing the amount of computer memory needed to store an image. Further, sparse representations tend to capture the true image without extraneous noise.

2 Problem Formulation and Accomplished Goals

In this section, we formulate image reconstruction as an optimization problem and present our accomplished goals within the first month of the project.

Consider a dictionary D and an observed image b which has been corrupted by a linear operator A and distorted by some noise ξ. A vectorized image $x \in \mathbb{R}^d$ is a "good" image if

it has a sparse representation y under the dictionary D, i.e.,

$$x = Dy, \text{ where } y \text{ is sparse.}$$

We require that $A(x) = A(Dy)$ be as close to the corrupted image b as possible by minimizing $\|A(Dy) - b\|^2$, while making sure that y is sparse. We thus add an additional regularization term to $\|A(Dy) - b\|^2$ to induce sparsity. The classical approach involves using the ℓ^1-norm regularization:

$$\text{minimize } \frac{1}{2}\|A(Dy) - b\|^2 + \lambda\|y\|_1, \tag{2.1}$$

where $\lambda > 0$ is a parameter

Another approach for sparsity-inducing uses a regularization term with differences of convex functions known as $(\ell_1 - \ell_2)$ regularization (see [14, 19, 20]):

$$\text{minimize } \frac{1}{2}\|A(Dy) - b\|^2 + \lambda(\|y\|_1 - \|y\|_2), \tag{2.2}$$

where $\lambda > 0$ is a parameter.

The optimization problem in (2.2) can be solved using the DCA with smoothing techniques; see [14]. However, we observe the slow convergence rate due to the high dimensionality of the data and the use of smoothing parameters. Note that if M is a standard 512×512 image, then the vectorized image belongs to $\mathbb{R}^{(512)^2} = \mathbb{R}^{262,144}$. In this project, we use different accelerated versions of the DCA in combination with the patching approach, which is used to divide the large image into small patches, to study (2.2) and compare our numerical results with the state-of-the-art methods for image reconstructions applied to (2.1). We also use the accelerated DCA to build a dictionary D instead of using an available one obtained from the DCT (Discrete Cosine Transform).

3 Patching

Through dividing the image into smaller pieces before beginning image reconstruction, improved results and execution speed are achieved. Patching is the process of dividing an $N_1 \times N_2$ image into smaller rectangular subdivisions. The patches will be indexed by row $(1 \le i \le t_1)$ and column $(1 \le j \le t_2)$, where t_1 and t_2 are the number of patches per row and number of patches per column of the original image, respectively.

First, the original image $M \in \mathbb{R}^{N_1\times N_2}$ is *vectorized* by adjoining the columns of M end-to-end. In particular, if $m_1, m_2, ..., m_{N_2} \in \mathbb{R}^{N_1}$ are the columns M, then $M = [m_1 m_2 ... m_{N_2}]$ and its vectorized form is $[m_1^\top m_2^\top ... m_{N_2}^\top]^\top$. We denote this form by $v(M)$.

For the patch in the ith row and the jth column, a *patch extraction matrix* $R_{ij} \in \mathbb{R}^{n_1 n_2 \times N_1 N_2}$ is defined through the indices of its upper-left corner (s, t), its number of rows n_1 and its number of columns n_2. In order to build R_{ij}, an indexing matrix $J \in \mathbb{R}^{n_1\times n_2}$ is first defined by

$$J_{rq} = N_1((t-1) + (q-1)) + s + (r-1)$$

for $1 \leq q \leq n_2$ and $1 \leq r \leq n_1$. Next, the matrix J is vectorized by v and used to define each row $r_k \in \mathbb{R}^{N_1 N_2}$ $(1 \leq k \leq n_1 n_2)$ of R_{ij}:

$$r_k = e_{v(J)_k}^\top,$$

where $\{e_k : k \in \{1, ...N_1 N_2\}\}$ is the set of standard basis vectors of $\mathbb{R}^{N_1 N_2}$. Thus, the patch extraction matrix can be framed as an identity matrix with missing rows. Note that the patch extraction matrices do not depend on the contents of the original image, only its size. Therefore, a set of patching matrices can be generated once, saved to a file, and re-used for all image reconstruction methods. The vectorized patch of the original image at index (i, j) is given by $P_{ij} = R_{ij} v(M) \in \mathbb{R}^{n_1 n_2}$.

4 Sampling and Noise

In order to distort the original image, a fraction of pixels are removed and Gaussian noise is added. Given a sample rate $S \in [0, 1]$, a set $\Omega \subseteq \{1, 2, ..., N_1 N_2\}$ represents which pixels of the image are sampled. For $1 \leq k \leq N_1 N_2$, a real number $\omega_k \in [0, 1]$ is chosen at random. If $\omega_k \leq S$, then $k \in \Omega$.

Next, each row of a *sampling operator* $A \in \mathbb{R}^{|\Omega| \times N_1 N_2}$ is defined by

$$A_{k:} = e_k^\top \tag{4.1}$$

for all $k \in \Omega$, where $\{e_k : k \in \{1, ...N_1 N_2\}\}$ is the set of standard basis vectors of $\mathbb{R}^{N_1 N_2}$. Given a vectorized image $v(M) \in \mathbb{R}^{N_1 N_2}$, $Av(M) \in R^{|\Omega|}$ therefore represents the original image with $N_1 N_2 - |\Omega|$ pixels deleted. Next, random noise $\xi \in \mathbb{R}^{|\Omega|}$ is generated and added to create the blurred vectorized image $B = Av(M) + \xi$.

5 Reconstructions of Small Images

In this section, we show how to apply techniques for general image restoration to a small blurred image b. The restored patch of size $n_1 \times n_2$ (usually 8×8) can be considered as a part of a larger image.

To create the reconstructed image, a dictionary matrix $D \in \mathbb{R}^{n_1 n_2 \times K}$ is used. The K columns of D are called the *atoms* of the dictionary. The number of atoms is usually chosen to be much larger than $n_1 n_2$. Dictionaries are created from two sources: the DCT (discrete cosine transform) or through a DCA-based dictionary learning process. The DCT dictionary used is defined as

$$D_{ij} = \begin{cases} \sqrt{\frac{1}{n_1 n_2}}, & j = 1 \\ \sqrt{\frac{2}{n_1 n_2}} \cos(\frac{\pi}{n_1 n_2}(j-1)(i+\frac{1}{2})), & j = 2, ..., n_1 n_2. \end{cases}$$

Since the sample operator for the entire image is large, computing products with it is inefficient. Furthermore, it does not need to be explicitly calculated. For each patch extraction

operator R_ij, we define $\mathcal{A} = A(R_{ij}^\top D)$. The value of $\mathcal{A}$ does not need to be found explicitly, so in practice functions $y \mapsto \mathcal{A}y$ and $z \mapsto \mathcal{A}^\top z$ are computed for each patch.

The goal of our optimization for each patch is to find a vector $y \in \mathbb{R}^K$ such that $x = Dy$ is close to the blurry patch b under the sample operator $\mathcal{A}$ and y is very sparse. Here, y is called the *sparse representation of x under D*. In essence, finding the value of y amounts to simultaneously minimizing two terms: an error term $\frac{1}{2}\|\mathcal{A}y - b\|^2$ and a sparsity penalty term $\|y\|_0$. However, the 0-norm cannot be used because it returns a discrete value (the integer number of non-zero entries in y). Therefore, we use the $\ell_1 - \ell_2$ regularization; $\|y\|_0 \approx \|y\|_1 - \|y\|_2$. Combining the two terms yields the overall function $f : \mathbb{R}^k \to \mathbb{R}$ defined by

$$f(y) = \frac{1}{2}\|\mathcal{A}y - b\|^2 + \lambda(\|y\|_1 - \|y\|_2), \tag{5.1}$$

where $\lambda > 0$ is a weight parameter which determines how sensitive the optimization is to the sparsity of y. By finding y for each patch of the image and recombining all patches, the restored image is generated.

6 The Boosted DCA Algorithm

In this section, we discuss the Boosted DCA algorithm. The Boosted DCA is an algorithm which outperforms the traditional DCA both in computation time and number of iterations for convergence. Below is the traditional DCA algorithm.

DCA Algorithm
INPUT: x_1, $N \in \mathbb{N}$
for $k = 1, \ldots, N$ **do**
    Find $y_k \in \partial h(x_k)$
    Find $x_{k+1} \in \partial g^*(y_k)$
end for
OUTPUT: x_{N+1}

The Boosted DCA is similar, except there is a line search which improves performance. We outline the steps below.

Boosted DCA Algorithm
INPUT: x_0, $N \in \mathbb{N}$,
$\alpha > 0, \bar{\lambda} > 0, 0 < \beta < 1$.
for $k = 0, \ldots, N$ **do**
Find $z_k \in \partial h(x_k)$.
Solve $y_k = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}}\{g(x) - \langle z_k, x\rangle\}$.
Set $d_k = y_k - x_k$.
if $d_k = 0$, **stop, return** x_k. **else** continue.
Set $\lambda_k = \bar{\lambda}$.
while $f(y_k + \lambda_k d_k) > f(y_k) - \alpha\lambda_k \|d_k\|^2$
Set $\lambda_k = \beta\lambda_k$
Set $x_{k+1} = y_k + \lambda_k d_k$.
if $x_{k+1} = x_k$, **stop, return** x_k.
end for
OUTPUT: x_{N+1}

Note that $x_{k+1} \in \partial g^*(y_k)$ is equivalent to $y_k \in \partial g(x_{k+1})$ by a property of the Fenchel conjugate. This in turn is equivalent to

$$x_{k+1} = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}}\{g(x) - \langle y_k, x\rangle\}.$$

This is because

$$\partial(g(x) - \langle y_k, x\rangle) = \partial g(x) - y_k$$

and 0 is in the subdifferential of a function at a local minimum. Thus, the first several steps of the two algorithms are indeed equivalent. If $\lambda_k = 0$ then the steps of the Boosted DCA and DCA are the same for that iteration. The term $d_k = y_k - x - k$ is a descent direction and the while loop initiates a line search which will give us a better x_{k+1} than the DCA.

7 DCA with Smoothing Algorithm

The DCA Algorithm is a useful tool for minimizing functions of the form $f = g - h$ where g and h are convex. In our case, $f(x) = \frac{1}{2}\|\mathcal{A}x - b\|^2 + \lambda(\|x\|_1 - \|x\|_2)$. Since $\|x\|_1$ is non-smooth, we wish to find a smooth approximation which will enable a faster computation of the DCA. To do so, we use Nesterov's Smoothing Technique. Given a function of the form

$$q(x) = \max_{u \in Q}\{\langle \mathcal{A}x, u\rangle - \phi(x)\},$$

we may find a smooth approximation for a parameter $\mu > 0$ by the function

$$q_\mu(x) = \max_{u \in Q}\{\langle \mathcal{A}x, u\rangle - \phi(x) - \frac{\mu}{2}\|u\|^2\}.$$

If $Q = \{x \in \mathbb{R}^n \mid |x_i| \leq 1\}$, the unit box, we see that the function $p(x) = \|x\|_1$ can be written

$$p(x) = \max_{u \in Q}\{\langle x, u\rangle\},$$

and hence a smooth approximation corresponding to $\mu > 0$ is

$$p_\mu(x) = \max_{u \in Q}\{\langle x, u\rangle - \frac{\mu}{2}\|u\|^2\}.$$

Note that

$$\begin{aligned}
p_\mu(x) &= \max_{u \in Q}\{\langle x, u\rangle - \frac{\mu}{2}\|u\|^2\} \\
&= -\frac{\mu}{2}\min_{u \in Q}\{\langle -\frac{2x}{\mu}, u\rangle + \|u\|^2\} \\
&= -\frac{\mu}{2}\min_{u \in Q}\{-\frac{1}{\mu^2}\|x\|^2 + \frac{1}{\mu^2}\|x\|^2 - \langle \frac{2x}{\mu}, u\rangle + \|u\|^2\} \\
&= \frac{1}{2\mu}\|x\|^2 - \frac{\mu}{2}\min_{u \in Q}\{\|u - \frac{x}{\mu}\|\} \\
&= \frac{1}{2\mu}\|x\|^2 - \frac{\mu}{2} d\left(\frac{x}{\mu}, Q\right)^2.
\end{aligned}$$

This function has gradient

$$\nabla p_\mu(x) = \Pi_Q(x)$$

where $\Pi_Q(x)$ is the projection onto Q. We approximate $f(x) = \frac{1}{2}\|Ax - b\|^2 + \lambda\|x\|_1 - \lambda\|x\|$ by

$$\begin{aligned}
f_\mu(x) &= \frac{1}{2}\|\mathcal{A}x - b\|^2 + \frac{\lambda}{2\mu}\|x\|^2 - \frac{\lambda\mu}{2} d\left(\frac{x}{\mu}, Q\right)^2 - \lambda\|x\| \\
&= \frac{\lambda}{2\mu}\|x\|^2 + \frac{\gamma}{2}\|x\|^2 - \left(\frac{\lambda\mu}{2} d\left(\frac{x}{\mu}, Q\right)^2 + \lambda\|x\| - \frac{1}{2}\|\mathcal{A}x - b\|^2 + \frac{\gamma}{2}\|x\|^2\right).
\end{aligned}$$

We set $g(x) = \left(\frac{\lambda + \mu\gamma}{2\mu}\right)\|x\|^2$ and $h(x) = \frac{\lambda\mu}{2} d(\frac{x}{\mu}, Q)^2 + \lambda\|x\| - \frac{1}{2}\|\mathcal{A}x - b\|^2 + \frac{\gamma}{2}\|x\|^2$. The constant $\gamma > 0$ is chosen so that the function $\frac{\gamma}{2}\|x\|^2 - \|\mathcal{A}x - b\|^2$ is convex and hence h is convex. In our work, we set $\gamma = 50/\lambda$. Recall that we wish to find $y_k \in \partial h(x_k)$. We compute

$$\begin{aligned}
\partial h(x) &= \lambda\mu(\mu^{-1}x - \Pi_Q(\mu^{-1}x)\mu^{-1} - \mathcal{A}^T(\mathcal{A}x - b) + \gamma x + \lambda\partial\|x\| \\
&= \left(\frac{\lambda + \gamma\mu}{\mu}\right) x - \lambda\Pi_Q(\mu^{-1}x) - \mathcal{A}^T(\mathcal{A}x - b) + \lambda\partial\|x\|.
\end{aligned}$$

Thus, we must compute $\partial\|x\|$. We know that $p(x) = \|x\|$ is differentiable when $x \neq 0$ and $\nabla p(x) = \frac{x}{\|x\|}$ in this case. When $x = 0, \partial p(x) = \mathbb{B}$, the closed unit ball. Thus, we use the function

$$\omega(x) = \begin{cases} \frac{x}{\|x\|} & x \neq 0, \\ 0 & x = 0 \end{cases}$$

to compute an element of $\partial\|x\|$. We note that for $y = \Pi_Q(x)$,

$$y_i = \begin{cases} 1 & x_i \geq 1 \\ x_i & |x_i| \leq 1 \\ -1 & x_i \leq -1, \end{cases}$$

and thus we have a simple formula for computing $\Pi_Q(x)$. After computing $y_k \in \partial h(x_k)$, we must find $x_{k+1} \in \partial g^*(y_k)$ which is equivalent to finding x_{k+1} such that $y_k \in \partial g(x_{k+1})$. This is easily achieved since g is differentiable with gradient

$$\nabla g(x) = \left(\frac{\lambda + \mu\gamma}{\mu}\right) x$$

and thus

$$y_k = \left(\frac{\lambda + \mu\gamma}{\mu}\right) x_{k+1}$$

implies

$$x_{k+1} = \left(\frac{\mu}{\lambda + \mu\gamma}\right) y_k.$$

The algorithm thus works as follows.

DCA with Smoothing Algorithm
INPUT: x_1, $N \in \mathbb{N}$
for $k = 1, \ldots, N$ **do**
Compute $y_k = \left(\frac{\lambda+\gamma\mu}{\mu}\right) x - \lambda\Pi_Q(\mu^{-1}x_k) - \mathcal{A}^T(\mathcal{A}x_k - b) + \lambda\omega(x_k)$.
Compute $x_{k+1} = \left(\frac{\mu}{\lambda+\mu\gamma}\right) y_k$.
end for
OUTPUT: x_{N+1}

8 Boosted DCA with Smoothing Algorithm

The algorithm we implemented combines the methods of the Boosted DCA and the DCA with smoothing algorithms. First, we compute $z_k \in \partial h(x_k)$ and then find $y_k \in \partial g^*(z_k)$ in the same manner as in the DCA with smoothing algorithm. Then we execute the line search. The steps are as follows.

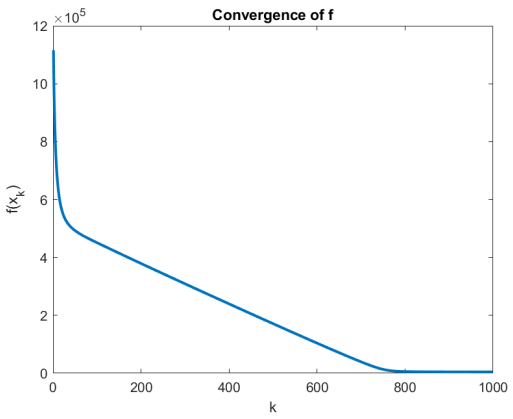
Boosted DCA with Smoothing Algorithm
INPUT: x_0, $N \in \mathbb{N}$,
$\alpha > 0, \bar{\lambda} > 0, 0 < \beta < 1$.
for $k = 0, \ldots, N$ **do**
Compute $z_k = \left(\frac{\lambda+\gamma\mu}{\mu}\right) x - \lambda\Pi_Q(\mu^{-1}x_k) - \mathcal{A}^T(\mathcal{A}x_k - b) + \lambda\omega(x_k)$.
Compute $y_k = \left(\frac{\mu}{\lambda+\mu\gamma}\right) z_k$.
Set $d_k = y_k - x_k$.
if $d_k = 0$, **stop, return** x_k. **else** continue.
Set $\lambda_k = \bar{\lambda}$.
while $f_\mu(y_k + \lambda_k d_k) > f_\mu(y_k) - \alpha\lambda_k\|d_k\|^2$
Set $\lambda_k = \beta\lambda_k$
Set $x_{k+1} = y_k + \lambda_k d_k$.
if $x_{k+1} = x_k$, **stop, return** x_k.
end for
OUTPUT: x_{N+1}

9 Results and Discussion

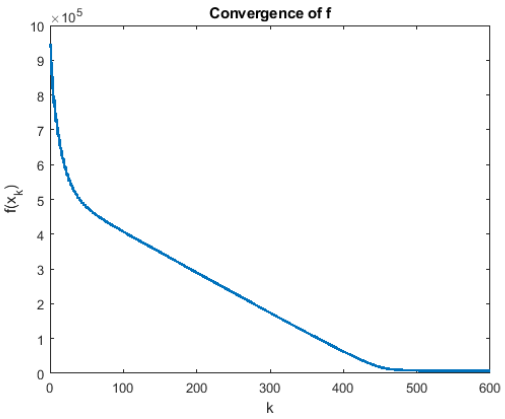
Sampled image

DCA, DCT Dictionary

Boosted DCA, DCT Dictionary



DCA, DCT Dictionary



Boosted DCA, DCT Dictionary

	Rel. Error (%)	PSNR	time(s)
DCA	7.04	82.6772	53.6187
Boosted DCA	6.22	83.7584	601.9423

Figure 1: Results for denoising and inpainting problems using the DCA and Boosted DCA. The DCT dictionary used for both algorithms. The PSNR, RE, and time are averaged.

To evaluate the quality of our reconstructed image, we tested with two measurements; the relative error (RE), measuring the difference between our original image and our reconstructed image given as $RE = \frac{\|M-\hat{M}\|}{\|M\|}$ and the peak signal to noise ratio (PSNR) which calculates the max possible value of a signal, represented roughly by the number of pixels, and the value of the distorting noise that affects the quality of our image, measured as $PSNR = 20\log_{10}\frac{\sqrt{N_1 N_2}}{\|M-\hat{M}\|_F}$, where M is our original image, $\hat{M}$ is our reconstructed image, N_1 and N_2 is the image size. For the RE, the lower the percent, the better, while for our PSNR, the higher the measurement, the better.

In terms of convergence rate for the inpainting test, the Boosted DCA with the line search converged in fewer iterations, with approximately 600 iterations, as opposed to the DCA convergence of approximately 800 iterations. For the relative error and peak signal to noise ratio, the Boosted DCA had the best RE of 6.22% and a PSNR of 83.76 as compared to the DCA which resulted in a RE of 7.04% and a PSNR of 82.68.When it came to time, the DCA was faster at approximately 54 seconds as compared to the Boosted DCA time of approximately 602 seconds.

These results play an important role when it comes to the real world, where computer algorithms are used to enhance videos or images. This is especially useful in the police force, where noisy images can prevent the identification and apprehension of criminals. These applications may be further expanded by enhancing the algorithms in future work through the exploration of dictionary learning to improve our image quality. This would improve the image by creating a dictionary fit for the input data and greatly increase the sparsity, rather than if we had used a predefined dictionary which may not be ideal for the feature space of our images.

References

[1] Aharon M, Elad M, Bruckstein A. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. IEEE Trans. Signal Process. **54** (2006), 4311–4322.

[2] An LTH, Tao PD. DC programming and DCA: thirty years of developments. Mathematical Programming. Special Issue: DC Programming - Theory, Algorithms and Applications, 169(1):564, 2018.

[3] Beck A, Teboulle M. Smoothing and first order methods: A unified framework. SIAM J. Optim. **22**(2012), 557–580.

[4] Beck A, Teboulle M. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM J. Imaging Sci. **2** (2009), 183–202.

[5] Clarke FH. Nonsmooth Analysis and Optimization. John Wiley & Sons, Inc., New York, 1983.

[6] Giles JR. A Survey of Clarkes Subdifferential and the Differentiability of Locally Lipschitz Functions. In: Progress in Optimization. Applied Optimization, vol 30. Springer, Boston, MA.

[7] P. Hartman, On functions representable as a difference of convex functions. *Pacific J. Math.* **9**, (1959), 707–713.

[8] J. B. Hiriart-Urruty, Generalized differentiability, duality and optimization for problems dealing with differences of convex functions. *Lecture Note in Economics and Math. Systems.* **256** (1985), 37–70.

[9] Mairal J, Bach F, Ponce J, Sapiro G. Online dictionary learning for sparse coding. Proc. 26th Int'l Conf. Machine Learning. Montreal, Canada, 2009.

[10] Martin D, Fowlkes C, Tal D, Malik J. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. Proc. 8th Int'l Conf. Computer Vision. **2** (2001), 416–423.

[11] Mordukhovich BS. Variational Analysis and Generalized Differentiation, I: Basic Theory, II: Applications. Grundlehren Series (Fundamental Principles ofMathematical Sciences), Vols. 330 and 331, Springer, Berlin, 2006.

[12] Mordukhovich BS, Nam NM. An Easy Path to Convex Analysis and Applications. Morgan & Claypool, 2014.

[13] B. S. Mordukhovich, N.M. Nam, and N. D. Yen, Fréchet subdifferential calculus and optimality conditions in nondifferentiable programming. *Optimization.* **55** (2006), 685–708.

[14] N.M. Nam, L.T.H. An, N.T. An, D.Giles, Smoothing techniques and difference of convex functions algorithms for image reconstructions, Optimization (2019), accepted.

[15] Nesterov Y. Smooth minimization of non-smooth functions. Math.Program., Ser. A, **103** (2005), 127–152.

[16] Pham Dinh T, Le Thi HA. Convex analysis approach to D.C. programming: Theory, algorithms and applications. Acta Math. Vietnam. **22** (1997), 289–355.

[17] Pham Dinh T, Le Thi HA. A d.c. optimization algorithm for solving the trust-region subproblem. SIAM J. Optim., **8** (1998), 476–505.

[18] Vandenberghe L. Optimization methods for large-scale systems, EE236C lecture notes, UCLA.

[19] Xin J, Osher S, Lou Y. Computational aspects of L1-L2 minimization for compressive sensing. Advances in Intelligent Systems and Computing, **359** (2015), 169–180.

[20] Yin P, Lou Y, He Q, Xin J. Minimization of L1-L2 for compressed sensing. SIAM J. of Sci. Comput. **37** (2015), A536–A563.

[21] Xu Y, Yin W. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. SIAM J. Imaging Sci. **6**(2013), 1758–1789.

[22] Xu Y, Yin W. A fast patch dictionary method for whole image recovery. Inverse Problems and Imaging, **10** (2016), 563–583.

[23] Zălinescu C. Convex Analysis in General Vector Spaces, World Scienctific, Singapore, 2002.