Syracuse University

SURFACE

Syracuse University Honors Program Capstone Projects

Syracuse University Honors Program Capstone Projects

Spring 5-1-2011

Color Image Noise Reduction with the Total Variation Model and Proximity Operators

Aaron Katchen

Follow this and additional works at: https://surface.syr.edu/honors_capstone

 Part of the Applied Mathematics Commons

Recommended Citation

Katchen, Aaron, "Color Image Noise Reduction with the Total Variation Model and Proximity Operators" (2011). *Syracuse University Honors Program Capstone Projects*. 246.
https://surface.syr.edu/honors_capstone/246

This Honors Capstone Project is brought to you for free and open access by the Syracuse University Honors Program Capstone Projects at SURFACE. It has been accepted for inclusion in Syracuse University Honors Program Capstone Projects by an authorized administrator of SURFACE. For more information, please contact surface@syr.edu.

Color Image Noise Reduction with the Total Variation Model and Proximity Operators

A Capstone Project Submitted in Partial Fulfillment of the Requirements of the Renée Crown University Honors Program at Syracuse University

Aaron Katchen

Candidate for B.A. Degree
and Renée Crown University Honors

May 2011

Honors Capstone Project in Mathematics

Capstone Project Advisor:_____
Professor Lixin Shen

Honors Reader:_____
Professor Yuesheng Xu

Honors Director:_____
James Spencer, Interim Director

Date:_____

Abstract

The following paper discusses how efficient and effective color image noise reduction may be achieved through the use of mathematic numerical analysis. Digital image noise is a longstanding problem for which efficient and effective solutions are critical to the advancement of the field of digital imaging. Micchelli-Shen-Xu [3] used the Total Variation Model in conjunction with proximity operators to propose a set of algorithms to effectively and efficiently solve for noisy grayscale images. They proposed the use of the proximity operator in anisotropic and isotropic total variation in fixed point algorithms. The following paper will discuss their algorithms as well as expand and implement these algorithms to apply to color images as well. When reducing noise in color images we may either apply the fixed point algorithm proposed by Micchelli-Shen-Xu [3] to the luma channel of YCbCr colorspace or apply the algorithm in parallel to the R G B channels of RGB colorspace. The later algorithm will produce better results at the expense of efficiency.

Table of Contents

Acknowledgements

My journey to this capstone all began one afternoon in my middle-school hallway in between classes in 8th grade. After a semi-dismal year of school I had signed up for the normal incoming math class at Central High School for 9th grade, when I was approached by Mr. Giotas, my math teach at the time, who persuaded and recommended I take the honors math class the following year. With his support I began what would turn out to be a nine-year roller coaster of an experience.

From there I met many other teachers who pushed me to be my best, in not only math, but in academics and life. For four years Frau Calvano and Mr. Sterling scolded me when I didn't try hard enough and praised me when I did; Mrs. J introduced me to applied math (horse betting based on statistics) and Mrs. Fallu somehow made calculus fun.

My interest in computational mathematics and numerical analysis was first sparked my freshman year at Syracuse University as a young aerospace engineering major. I quickly excelled in Professor Dannenhoffer's ECS 104 class when I was first introduced to MATLAB and the concept of numerical analysis. I used those skills to acquire an internship at Zanaqua Technologies where Dave Dussault took me under his wing and taught me everything there was to know about FORTRAN 90 and helped me develop an advance sense of numerical analysis.

Shortly after, I changed out of aerospace engineering to pursue photography, but I kept my math with me. Professor Banerjee furthered my knowledge of numerical analysis and suggested I pursue my capstone in the field. He introduced me with Professor Shen, who generously donated three semesters of his time for one on one

lessons, paper reviews, tutorials, and ultimately proved to be my capstone advisor. I studied his work with Professor Xu who generously agreed to be my honors capstone reader.

Of course, none of this would be possible without my parents and family who supported me financially and emotionally the whole time and my wonderful girlfriend Allana who has listened to me complain about anything and everything since we first met. To everyone mentioned here and so many more who are not, I thank you for every bit of help you've given and every bit of knowledge you've bestowed upon me.

1. Overview

Efficient and effective noise reduction algorithms are essential to modern science in a multitude of ways. They are regularly used in the communications, medical, military, and scientific analysis fields. This paper will explore fixed point algorithms using the Total Variation Model; both anisotropic as well as isotropic total variation will be considered. We will first consider these algorithms on grayscale square images, progress to non-square grayscale images, and finally non-square multi-dimensional color images. This paper will attempt to provide an efficient and effective solution to non-square multi-dimensional color image noise reduction, using isotropic total variation in a fixed point algorithm.

2. Introduction

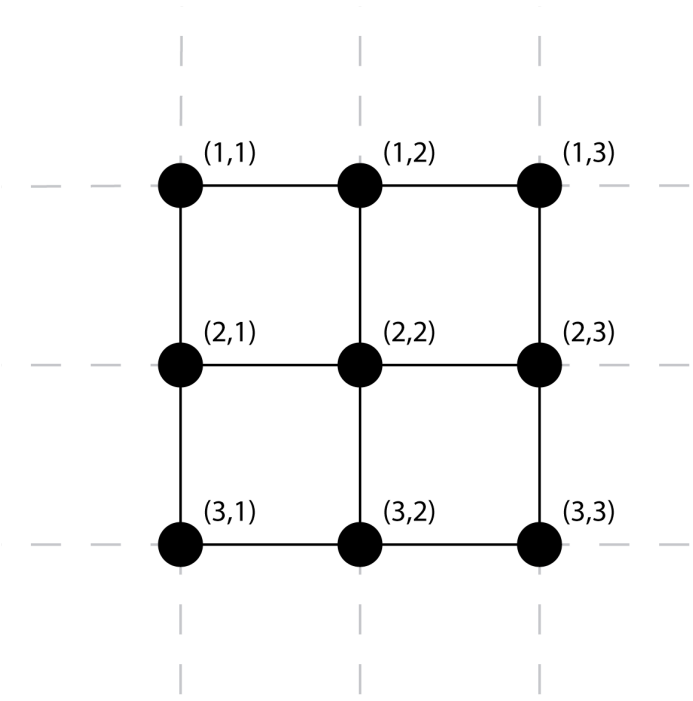
Throughout this paper, we will represent images as matrices, wherein each matrix element represents a pixel of the image. Pixel values, ($u_{i.j}$), will be converted to

$$\{u : u \in \mathbb{R}^d, 0 \le u_i \le 1, \text{ for } i = 1,2,...,d\} \quad (1)$$

where 0 is the darkest possible value and 1 is the brightest possible value. In the application of grayscale images, 0 would be black and 1 would be white. In the application of color images, 0 would be black and 1 would be the brightest possible value of that color dimension. All matrices will be

converted back to their original bit-depth and colorspace upon completion of the algorithm.

We will use the Total Variation Model to solve for the noise matrix in the images. The Total Variation Model will filter using nodes compared to other models which may filter using edges. This means that when calculating gradient information at a particular location, the Total Variation Model will produce smoother gradients and less “gridding” than a method which utilizes edges. “Gridding” is the phenomenon of visible grids forming at gradient areas rather than smooth tonal transitions. The figure below shows how node (2,2) is ‘connected’ and receiving gradient information from eight other surrounding nodes.



When using the Total Variation Model,

$$\min\{\tfrac{1}{2}\|u-x\|_2^2+\alpha\|u\|_{TV} : u\in\mathbb{R}^d\} \tag{2}$$

where u is the original noisy image, x is the noise for which we wish to solve, $\{\alpha : 0<\alpha,\alpha\in\mathbb{R}\}$, and $\|u\|_{TV}$ is the total variation of the u, $\|u\|_{TV}$ may be defined quite a few ways. The algorithms in this paper will use both anisotropic (ATV) and isotropic (ITV) total variation.

The Total Variation Model, as described in statement (2), is a composition of a convex function (ℓ^1 norm for anisotropic and ℓ^2 norm for isotropic) and a linear transformation (in this case the first order difference operator). The convex nature of the proposed Total Variation Model, when used in an iterative fixed point algorithm, should lead to convergence. The Total Variation Model uses gradient information of the image in an attempt to preserve hard edges during the noise reduction process. By using gradient information, the algorithm can then leave parts of the image alone where the gradient is high (hard edges) and smooth out parts of the image where the gradient is generally low (smooth areas). This method allows images to remain sharp after the algorithm has been applied.

3. Proximity Operator and Subdifferential

The proximity operator is a key element in the development of the iterative fixed point algorithm and the subdifferential is key to the execution of both ATV and ITV. These terms must be rigidly defined before continuing.

The proximity operator is defined as follows:

Definition 3.1 *Let ψ be a real-valued convex function on $\mathbb{R}^d$. The proximity operator of ψ is defined for $x \in \mathbb{R}^d$ by*

$$\operatorname{prox}_{\psi} x := \arg\min\left\{\tfrac{1}{2}\|u-x\|_2^2 + \psi(x) : u \in \mathbb{R}^d\right\} \tag{3}$$

The subdifferential is defined as:

Definition 3.2 *Let ψ be a real-valued convex function on $\mathbb{R}^d$. The subdifferential of ψ at $x \in \mathbb{R}^d$ is defined by*

$$\partial\psi(x) := \left\{y : y \in \mathbb{R}^d \text{ and } \psi(z) \geq \psi(x) + \langle y, z-x\rangle \forall z \in \mathbb{R}^d\right\} \tag{4}$$

An example demonstrating how the subdifferential and proximity operator apply to a simple two-dimensional absolute value function $\psi = \frac{4}{\lambda}|\cdot|$, where $\lambda > 0$ may be as follows:

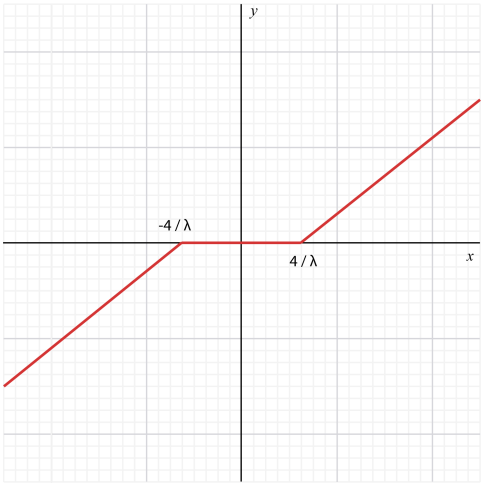
Example 3.3 *If $\lambda > 0$ and $x \in \mathbb{R}$ then*

$$\partial\left(\tfrac{4}{\lambda}|\cdot|\right)(x) = \begin{cases} \frac{4}{\lambda}\{\operatorname{sign}(x)\}, & x \neq 0 \\ \left[-\frac{4}{\lambda}, \frac{4}{\lambda}\right], & \textit{otherwise} \end{cases} \tag{5}$$

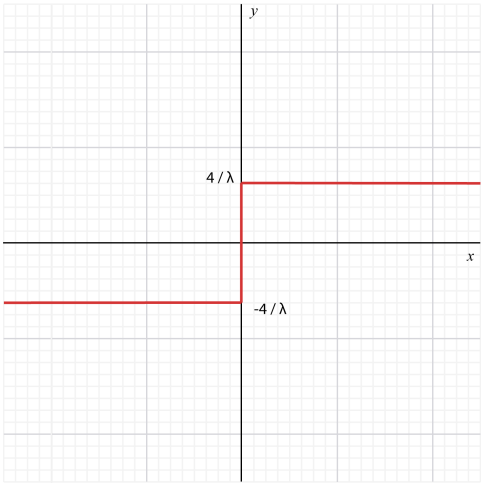
and

$$\text{prox}_{\frac{4}{\lambda}} x = \max\left(|x| - \tfrac{4}{\lambda}, 0\right)\text{sign}(x)$$

This example can be demonstrated visually by the figures below. The subdifferential, as stated in the example, is either $\frac{4}{\lambda}$ or $-\frac{4}{\lambda}$ unless $x = 0$, at which point it may be any value in between. This allows a derivative to be defined for non-continuous points on a function, such as the $x = 0$ value of $y = |x|$. The proximity operator acts as a threshold operator based upon a predetermined lambda value.



proximity operator



subdifferential

Although this example demonstrates the two-dimensional case, these principals may be applied to any m-dimensional case, where $m \in \mathbb{N}$.

4. Formulation of Iterative Fixed Point Problem

Now that the proximity operator and subdifferential have been defined, the formulation of the iterative fixed point algorithm that will be used may be explained. First, an $m \times d$ matrix *B,* which will represent the first order difference operator, as well as a convex function φ on $\mathbb{R}^m$ will be used to define a function $\psi : \forall x \in \mathbb{R}^d$ such that

$$\psi(x) \coloneqq (\varphi \circ B)(x) . \tag{6}$$

The new function ψ is convex, since it is the composition of a convex function φ and the first order difference operator *B*; this will guarantee convergence later. We now wish to develop an algorithm that will evaluate $\text{prox}_{\varphi \circ B}$. We will define $\hat{u} \in \mathbb{R}^m$ such that

$$\hat{u} \coloneqq \text{prox}_{\varphi \circ B}(x) \tag{7}$$

where $x \in \mathbb{R}^d$. From the optimality condition in convex analysis it may be concluded that

$$0 \in \hat{u} - x + \partial(\varphi \circ B)(\hat{u}) \tag{8}$$

which, by using the chain rule, may be rewritten as

$$0 \in \hat{u} - x + B^T \partial\varphi(B\hat{u}) . \tag{9}$$

We now say there exists some vector b where

$$\exists b \in \tfrac{1}{\lambda} \partial\varphi(B\hat{u}) \tag{10}$$

where $\lambda > 0$. We may then conclude

$$\begin{aligned} 0 &= \hat{u} - x + \lambda B^T b \\ -\hat{u} &= x - \lambda B^T b \end{aligned} . \tag{11}$$

Since (11) is true, we may combine this with **Definition 3.1** to conclude

$$\begin{aligned} b &\in \tfrac{1}{\lambda}\partial\varphi(B\hat{u}) \\ b + B\hat{u} &= \operatorname{prox}_{\frac{1}{\lambda}\varphi}(b + B\hat{u}) + b \\ b &= (b + B\hat{u}) - \operatorname{prox}_{\frac{1}{\lambda}\varphi}(b + B\hat{u}) \end{aligned} . \tag{12}$$

By simply rewriting (12) we obtain

$$b = \left(\mathrm{I} - \operatorname{prox}_{\frac{1}{\lambda}\varphi}\right)(b + B\hat{u}) \tag{13}$$

By combining with (11) we conclude

$$b = \left(\mathrm{I} - \operatorname{prox}_{\frac{1}{\lambda}\varphi}\right)\left(b + B\left(x - \lambda B^T b\right)\right) \tag{14}$$

This may be written as the fixed point problem which will be used in calculations

$$b = \left(\mathrm{I} - \operatorname{prox}_{\frac{1}{\lambda}\varphi}\right)\left(Bx + \left(\mathrm{I} - \lambda BB^T\right)b\right). \tag{15}$$

By using the Picard iterative process on equation (15), we may calculate the vector, b, of the noisy image, x, for some $\lambda > 0$. Once the vector, b, has been calculated, it can be used to correct for noise in a the noisy image, x, as follows:

$$x_{clean} = x_{noisy} - \lambda B^T b \tag{16}$$

5. Gradient, Adjoint, and Discrete Divergence Definitions for Algorithm Execution

This section will focus on methods of executing equation (15) using both ATV and ITV. Before going any further, ∇u must be defined, where u is a 2-dimensional $N \times N$ image. Later we will easily adapt this to include 3-dimensional $M \times N$ images. Our current definition of ∇u is based on [2] Chambolle's definition and is as follows.

Definition 5.1 *We denote X as the Euclidean space $\mathbb{R}^{NxN}$. If $u \in X$, the ∇u is a vector in $Y = X \times X$ given by*

$$(\nabla u)_{i,j} = \left((\nabla u)^1_{i.j}, (\nabla u)^2_{i.j} \right) \tag{17}$$

with

$$(\nabla u)^1_{i.j} = \begin{cases} u_{i+1,j} - u_{i,j} & \text{if } i < N \\ 0 & \text{if } i = N \end{cases}$$

$$(\nabla u)^2_{i.j} = \begin{cases} u_{i,j+1} - u_{i,j} & \text{if } j < N \\ 0 & \text{if } j = N \end{cases}$$

for $i, j = 1, ..., N$.

This definition of ∇u allows the computation of the gradient at all edges of the image, which allows flexibility during execution. We will now define an $N \times N$ matrix D_N as follows

$$D_N := \begin{bmatrix} 0 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix} \tag{18}$$

We may then define the $2N^2 \times N^2$ matrix B, based upon **Definition 5.1**, as

$$B := \begin{bmatrix} \mathrm{I}_N \otimes D_N \\ D_N \otimes \mathrm{I}_N \end{bmatrix} \tag{19}$$

where $F \otimes G$ is the Kronecker product of F and G. It should be noted, that the matrix B is an operator which determines the gradient of a $N^2 \times 1$ vector; this vector represents our image u. It should also be noted that BB^T, as described in the fixed point problem (15), calculates the the adjoint of a $N^2 \times 1$ vector.

6. Anisotropic Total Variation

Anisotropic total variation is a directionally independent version of the total variation. This means that the proximity operator will consider the i - directional gradient completely independent of the j - directional gradient. The anisotropic total variation term may be defined as

$$\|u\|_{TV} \coloneqq \|Bu\|_1 . \tag{20}$$

To implement this definition of total variation into the fixed point problem (15), we may define

$$\varphi(z) \coloneqq \mu\|z\|_1 \tag{21}$$

where $z \in \mathbb{R}^{2N^2}$ and $\mu > 0$. This redefines the proximity operator for anisotropic total variation as $\operatorname{prox}_{\frac{\mu}{\lambda}\|\cdot\|_1}$ and thus alters (15) to

$$b = \left(\mathrm{I} - \operatorname{prox}_{\frac{\mu}{\lambda}\|\cdot\|_1}\right)\left(Bx + \left(\mathrm{I} - \lambda BB^T\right)b\right). \tag{22}$$

By applying **Definition 3.1** to our new form of the proximity operator shown in (22), we will essentially just apply the same version of the proximity operator demonstrated in **Example 3.3** to each element of the operand except we will now have $\frac{\mu}{\lambda}$ instead of $\frac{1}{\lambda}$.

Equation (22) may now be implemented into a working algorithm for input into a computer program (Appendix, §12). The implementation is not a straightforward interpretation of the fixed point problem, but rather a

program which interprets the meaning of each individual step due to memory restrictions. For example, instead of calculating Bx, which would be the multiplication of a $20{,}000 \times 10{,}000$ matrix with a $10{,}000 \times 1$ matrix for a 100×100 pixel image, we can easily just calculate the gradient of the image and never have to create a matrix larger than 100×100.

7. Isotropic Total Variation

Isotropic total variation is directionally dependent. This forces the row-directional and column-directional gradients to be codependent; in reference to **Definition 5.1**, $(\nabla u)^1_{i,j}$ and $(\nabla u)^2_{i,j}$ are codependent. This version of the total variation is more complex, as it attempts to solve a non-linear PDE, but it tends to produce better results. In this case, better results refers to less blurring while attaining the same level of noise reduction when compared to anisotropic total variation. The isotropic total variation may be defined as

$$\|u\|_{TV} \coloneqq \sum_{i=1}^{N^2} \left\| \begin{bmatrix} (Bu)_i \\ (Bu)_{N^2+i} \end{bmatrix} \right\|_2 \tag{23}$$

thus defining our φ for the proximity operator as

$$\varphi(z) \coloneqq \mu \sum_{i=1}^{N^2} \left\| \begin{bmatrix} z_i \\ z_{N^2+i} \end{bmatrix} \right\|_2 \tag{24}$$

where $z \in \mathbb{R}^{2N^2}$ and $\mu > 0$. Thus the proximity operator is written as $\text{prox}_{\frac{1}{\lambda}\varphi}$

and the fixed point problem is written as

$$b = \left(\text{I} - \text{prox}_{\frac{1}{\lambda}\varphi}\right)\left(Bx + \left(\text{I} - \lambda BB^T\right)b\right). \tag{25}$$

8. Redefining Terms for Non-Square Images

All the theory thus far has been defined in relation to square $N \times N$ images. In this section we will redefine terms and equations for non-square $M \times N$ images. Everything up through **Section 4** will remain the same. We will first redefine the gradient, ∇u, **Definition 5.1**. This is a simple definition to redefine for non-square images and the definition will be as follows

Definition 5.1.A *We denote X as the Euclidean space $\mathbb{R}^{MxN}$. If $u \in X$, the ∇u is a vector in $Y = X \times X$ given by*

$$(\nabla u)_{i,j} = \left((\nabla u)^1_{i.j}, (\nabla u)^2_{i.j}\right)$$

with (26)

$$(\nabla u)^1_{i.j} = \begin{cases} u_{i+1,j} - u_{i,j} & \text{if } i < M \\ 0 & \text{if } i = M \end{cases}$$

$$(\nabla u)^2_{i.j} = \begin{cases} u_{i,j+1} - u_{i,j} & \text{if } j < N \\ 0 & \text{if } j = N \end{cases}$$

for $i = 1,...,M$ *and* $j = 1,...,N$.

Clearly, all we've done to alter this definition is introduce the M dimension and separate the i^{th} row dimension from the j^{th} column dimension. The rest of the definition still applies as before, but now the columns of the image may be a different size than the rows.

In addition to our new definition of the gradient, we must slightly alter our definitions of D and subsequently B, our representation of the image u, and our definitions of our total variation to fully implement non-square images into our fixed point problem (15).

Since we just redefined the gradient, ∇u, we should discuss how this will apply to our fixed point problem (15). In **Section 5**, matrices D and B were defined to implement our definition of the gradient ∇u. It should be noted that for square images, matrix B was of size $2N^2 \times N^2$. This gradient operator was then applied to an image b of size $N \times N$ in vector form; in vector form, image b was of size $N^2 \times 1$. After the application of matrix B, the gradient was of size $2N^2 \times 1$; the first N^2 terms represented the i-directional gradient, while the second N^2 terms represented the j-directional gradient. Now we wish to apply all this theory

to an $M \times N$ image. Since the image u will now be of size $M \times N$, its vector representation will be of size $MN \times 1$. Since it's vector representation is of size $MN \times 1$, it only follows that our matrix B should be of size $2MN \times MN$; thus, after the application of B to our image, the gradient will be of size $2MN \times 1$, where the first MN terms will represent the i-directional gradient and the second MN terms will represent the j-directional gradient.

Now we will redefine our matrices D and B. Our top $MN \times MN$ terms of our new matrix B must apply the i-directional gradient, while our bottom $MN \times MN$ terms must apply the j-directional gradient. This leads us to the conclusion that we must adjust our matrix D. For the application of non-square images we will redefine the matrix D in two ways, D_M and D_N. Both these matrices will hold the same form as described in (18), but D_M will be of size $M \times M$, while D_N will be of size $N \times N$. Our new definition of matrix B is very similar to the old definition (19), but it shall be changed to:

$$B := \begin{bmatrix} \mathrm{I}_N \otimes D_M \\ D_N \otimes \mathrm{I}_M \end{bmatrix}. \tag{27}$$

This new definition of B will successfully apply both the i-directional and j-directional gradients to any image of size $M \times N$.

Our definition of the ATV will only change very slightly. We will still define

$$\|u\|_{TV} \coloneqq \|Bu\|_1 \tag{28}$$

as we did in (20), but now the proximity operator will be defined as

$$\varphi(z) \coloneqq \mu \|z\|_1 \tag{29}$$

where $z \in \mathbb{R}^{2MN}$ and $\mu > 0$. Notice that the only difference in the definition of the ATV is that $z \in \mathbb{R}^{2MN}$ instead of $z \in \mathbb{R}^{2N^2}$. This is a very minor change that only affects the proper definition of the proximity operator, but does not affect its implementation.

The definition of the ITV will change only slightly as well. We will now define the total variation as

$$\|u\|_{TV} \coloneqq \sum_{i=1}^{MN} \left\| \begin{bmatrix} (Bu)_i \\ (Bu)_{MN+i} \end{bmatrix} \right\|_2 . \tag{30}$$

The subsequent changes to the proximity operator are

$$\varphi(z) \coloneqq \mu \sum_{i=1}^{MN} \left\| \begin{bmatrix} z_i \\ z_{MN+i} \end{bmatrix} \right\|_2 \tag{31}$$

where $z \in \mathbb{R}^{2MN}$ and $\mu > 0$.

Although the most drastic change is of matrix B, these other small changes generalize the fixed point problem (15) to apply to both square and non-square images. This is essential, as most images will not be square.

9. Multi-Dimensional Matrix Manipulation and Color Image Processing

The next challenge is to apply this theory to color, or multi-dimensional, images. This is essential, since most digital images used in the modern world are not grayscale, but usually operate in either the RGB or YCbCr colorspaces. Both these colorspaces are 3-dimensional, but other multi-dimensional colorspaces exist, such as CMYK. For the sake of simplicity, we will focus on the more common RGB colorspace in this section.

There are multiple ways to approach the issue of noise reduction in color images. One method a.) suggests that we convert the color image to YCbCr color space, apply the algorithms to the Y, or luminance, channel and then recompose the image. Another similar method b.) suggests applying the algorithms to the B channel of an RGB image, since most digital noise is found in the blue channel anyway. A third method c.) suggests that we apply the algorithm to all three channels independently and upon completion recompose the image. Method a.) is efficient, since it only applies the algorithm to one channel, and fairly thorough, since a lot

of noise is often found in the Y channel, but it only accounts for luminance noise and does not account for color noise, which is a common problem in low light image capture. Method b.) is also efficient, since it only applies the algorithm to one channel, and takes care of some luminance and color noise, but does not consider any noise in the red or green channels. Method c.) is the most thorough and will solve both luminance and color noise problems, but isn't efficient as it requires many extra iterations. A fourth method may then be proposed; this method will apply the algorithm to all three channels, to account for both luminance and color noise, but will allow all three layers to work dependently upon each other, rather than independently. This will ensure that each layer only have the minimal amount of noise reduction applied to it and maximize results. It is still unclear how to properly approach this method, but this is a future research point.

For all methods, we will first redefine our input image u as

$$u := \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \tag{32}$$

where $u_k \in \mathbb{R}^{MN}$ for $k = 1,2,3$ and (u_1, u_2, u_3) represent the RGB channels respectively. The algorithm will run in parallel on the three separate channels, but our total variation will be redefined as one term for simplicity.

This may apply to both ATV and ITV. Our new definition of the ATV will be

$$\|u\|_{CATV} \coloneqq \|Bu_1\|_1 + \|Bu_2\|_1 + \|Bu_3\|_1 \tag{33}$$

where $u \in \mathbb{R}^{3MN}$. This new definition of the ATV applies the 2-dimensional ATV to each channel of the color image. This method when applied to the ITV appears as

$$\|u\|_{CITV} \coloneqq \sum_{p=1}^{MN} \left\| \begin{bmatrix} (Bu_1)_p \\ (Bu_1)_{MN+p} \end{bmatrix} \right\|_2 + \sum_{p=1}^{MN} \left\| \begin{bmatrix} (Bu_2)_p \\ (Bu_2)_{MN+p} \end{bmatrix} \right\|_2 + \sum_{p=1}^{MN} \left\| \begin{bmatrix} (Bu_3)_p \\ (Bu_3)_{MN+p} \end{bmatrix} \right\|_2 \tag{34}$$

where $u \in \mathbb{R}^{3MN}$. The same theory applies to the ITV as it did the ATV, we just now consider a different type of total variation. From (33), the definition of the color ATV proximity operator is

$$\varphi(z) \coloneqq \mu\|z_1\|_1 + \mu\|z_2\|_1 + \mu\|z_3\|_1 \tag{35}$$

where $z \in \mathbb{R}^{3MN}$ and $\mu > 0$. The definition for the ITV proximity operator is then

$$\varphi(z) \coloneqq \mu\sum_{p=1}^{MN} \left\| \begin{bmatrix} (z_1)_p \\ (z_1)_{MN+p} \end{bmatrix} \right\|_2 + \mu\sum_{p=1}^{MN} \left\| \begin{bmatrix} (z_2)_p \\ (z_2)_{MN+p} \end{bmatrix} \right\|_2 + \mu\sum_{p=1}^{MN} \left\| \begin{bmatrix} (z_3)_p \\ (z_3)_{MN+p} \end{bmatrix} \right\|_2 \tag{36}$$

where $z \in \mathbb{R}^{3MN}$ and $\mu > 0$.

The next step is to apply this knowledge to an algorithm. One way to approach this is mentioned above as “Method C”. This would run the algorithm proposed by [3] Micchelli-Shen-Xu on all three channels in parallel, independently. The algorithm would then be

Given: Noisy Image x; $\lambda > 0$; $\mu > 0$

Initialization: $v^0 = 0$

For $n = 0,1,2,...$

$$\begin{aligned} v_1^{n+1} &\leftarrow \left(I - \operatorname{prox}_{\frac{\mu}{\lambda}\|\cdot\|_1} \right)\left(Bx_1 + \left(I - \lambda BB^t \right) v_1^n \right) \\ v_2^{n+1} &\leftarrow \left(I - \operatorname{prox}_{\frac{\mu}{\lambda}\|\cdot\|_1} \right)\left(Bx_2 + \left(I - \lambda BB^t \right) v_2^n \right) \\ v_3^{n+1} &\leftarrow \left(I - \operatorname{prox}_{\frac{\mu}{\lambda}\|\cdot\|_1} \right)\left(Bx_3 + \left(I - \lambda BB^t \right) v_3^n \right) \end{aligned} \tag{37}$$

End

Write the output of $v^n = \begin{pmatrix} v_1^n \\ v_2^n \\ v_3^n \end{pmatrix}$ from the above loop as

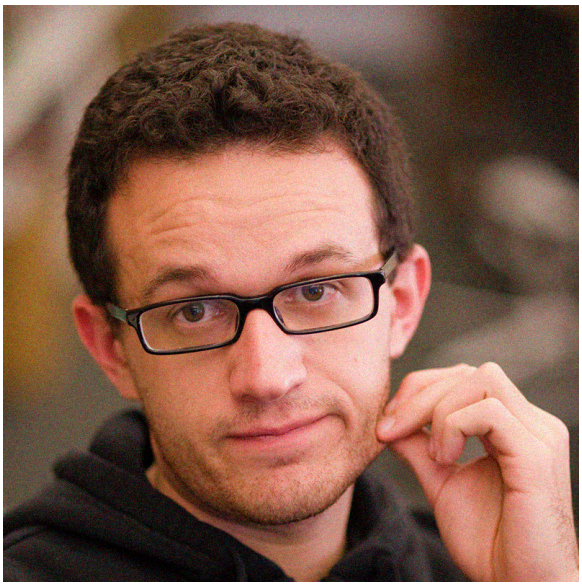
$v^\infty = \begin{pmatrix} v_1^\infty \\ v_2^\infty \\ v_3^\infty \end{pmatrix}$ and compute

$$\operatorname{prox}_{\varphi \circ B} x = \begin{pmatrix} x_1 - \lambda B^t v_1^\infty \\ x_2 - \lambda B^t v_2^\infty \\ x_3 - \lambda B^t v_3^\infty \end{pmatrix}$$

This algorithm could also be applied to the ITV. The ITV proximity operator would just have to be switched in for the current ATV proximity operator.

10. Experiments and Outcomes

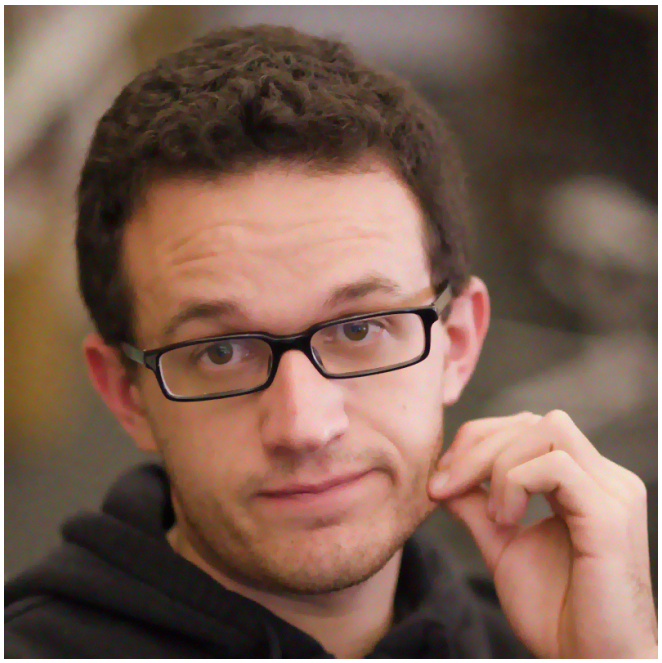
This section will show some experimental outcomes of different algorithms discussed in this paper. The algorithms will be applied to the following noisy image:



For every experimental outcome we will also show a detailed shot to examine the pixel structure and noise reduction more thoroughly.

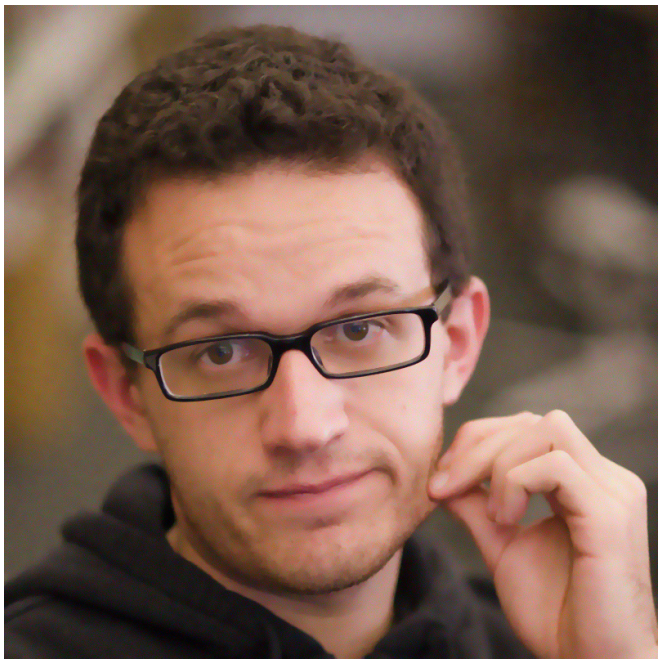
One-Dimensional Algorithm Applied to the Luminosity Channel in YCbCr with ATV

$\mu = 0.05$



One-Dimensional Algorithm Applied to the Luminosity Channel in YCbCr with ITV

$\mu = 0.05$



Three-Dimensional Algorithm Applied to un-linked RGB channels with ATV

$\mu = 0.05$

Three-Dimensional Algorithm Applied to un-linked RGB channels with ITV

$\mu = 0.05$

11. Conclusion and Further Research

After experimentation on color image processing, it appears that applying ATV and ITV algorithms introduced by Micchelli-Shen-Xu [3] in the one-dimensional case to the luminosity channel of YCbCr colorspace is a quick and efficient way to reduce noise in an image and may remain one of the best algorithms tested. Although this algorithm strictly focuses on luminosity noise and therefore does not clean up color noise, it converges quickly, leaves sharp edges, and generally produces desirable results.

Applying the algorithms to each color channel independently produces sharp edges and solves for both luminosity and color noise, but converges much more slowly than the one-dimensional algorithm applied to the luminosity layer. It seems that if time is no issue this is the approach to take, but for quick color image noise reduction results, one should use the one-dimensional luminosity channel algorithm.

For future research I would like to develop an algorithm which allows quick convergence three-dimensional color image noise reduction using a three-dimensional proximity operator.

12. Appendix

12.1 Standard Fixed Point Algorithm with ATV in MATLAB

```
function [ CLNImg ] = ImgDenoise_FPPO( Img, mu, beta )
%Denoising Program
%Theory by Lixin Shen, Yuesheng Xu, and Charles A. Micchelli
%Implementation by Aaron Katchen

%Fixed Point Algorithm Based on the Proximity Operator with ATV
(FP^20-ATV)

%-------------------------------------------------------

%Setup

%Initialize count
count = 1;

%initialize argmin variable
argmin = 1;

%Determine information about noisy image
%Read Image and convert to double
temp_X = imread(Img);

%Determine if image is color
color = isrgb(temp_X);

%if color image convert to YCbCr and strip Y channel
%otherwise just run as normal
if (color == 1);

    temp_X = rgb2ycbcr(temp_X);
    X = temp_X(:, :, 1);

else

    X = temp_X;

end

X = im2double(X);

%Determine size of image
[M, N] = size(X);

%create and initialize v matrix
V = zeros(2*M,N);

%determine lambda
lambda = 1/4*(sin((N-1)*pi/(2*N)))^(-2);

%determine alpha
alpha = mu / lambda;
```

```
%-----------------------------------------------------

%determine x and y differentials
BX = YXGradient(X);

%-----------------------------------------------------

%Initialize BtV
BtV = zeros(M,N);

%Initialize BBtV
BBtV = zeros(2*M, N);

%-----------------------------------------------------

%loop fixed point algorithm
while argmin>beta;

    %count
    count = count + 1

    %if there are more than 20 iterations, stop the code
    if count>20;
        break;
    end

    %Store old BtV for relative error calc
    BtV_Prev = BtV;

    %Calculate gamma
    gamma = BX + V - lambda * BBtV;

    %Calculate new V
    V = gamma - L1ProxOp(gamma, alpha);

    %Determine new B' of v
    BtV=CalcAdjoint(V);

    %Determine new BBtV
    BBtV=YXGradient(BtV);

    %Determine Relative Error
    argmin = norm((X - lambda * BtV) - (X - lambda * BtV_Prev)) /
norm((X - lambda * BtV));

end

%-----------------------------------------------------

%Set Output image and convert to uint8
CLNDoubleImg = X - lambda * BtV;


if (color == 1);

    temp_X(:, :, 1) = im2uint8(CLNDoubleImg);
    CLNImg = ycbcr2rgb(temp_X);
```

```
else

    CLNImg = mat2gray(CLNDoubleImg);

end

end
```

12.2 Standard Fixed Point Algorithm with ITV in MATLAB

```
function [ CLNImg ] = ImgDenoise_FPPO_ITV( Img, mu, beta )
%Denoising Program
%Theory by Lixin Shen, Yuesheng Xu, and Charles A. Micchelli
%Implementation by Aaron Katchen

%Fixed Point Algorithm Based on the Proximity Operator with ITV
(FP^20-ITV)

%-------------------------------------------------------

%Setup

%initialize count
count = 1;

%initialize argmin variable
argmin = 1;

%Determine information about noisy image
%Read Image and convert to double
temp_X = imread(Img);

%Determine if image is color
color = isrgb(temp_X);

%if color image convert to YCbCr and strip Y channel
%otherwise just run as normal
if (color == 1);

    temp_X = rgb2ycbcr(temp_X);
    X = temp_X(:, :, 1);

else

    X = temp_X;

end

X = im2double(X);

%Determine size of image
[M, N] = size(X);

%create and initialize v matrix
V = zeros(2*M,N);

%determine lambda
```

```
lambda = 1/4*(sin((N-1)*pi/(2*N)))^(-2);

%determine alpha
alpha = mu / lambda;

%-------------------------------------------------------

%determine x and y differentials
BX = YXGradient(X);

%-------------------------------------------------------

%Initialize BtV
BtV = zeros(M,N);

%Initialize BBtV
BBtV = zeros(2*M, N);

%-------------------------------------------------------

%loop fixed point algorithm
while argmin>beta;

    %count
    count = count + 1

    %if there are more than 20 iterations, stop the code
    if count>20;
        break;
    end

    %Store old BtV for relative error calc
    BtV_Prev = BtV;

    %Calculate gamma
    gamma = BX + V - lambda * BBtV;

    %Calculate new V
    V = gamma - L2ProxOp(gamma, alpha);

    %Determine new B' of v
    BtV=CalcAdjoint(V);

    %Determine new BBtV
    BBtV=YXGradient(BtV);

    %Determine Relative Error
    argmin = norm((X - lambda * BtV) - (X - lambda * BtV_Prev)) /
norm((X - lambda * BtV));

end

%-------------------------------------------------------

%Set Output image and convert to uint8
CLNDoubleImg = X - lambda * BtV;
```

```
if (color == 1);

    temp_X(:, :, 1) = im2uint8(CLNDoubleImg);
    CLNImg = ycbcr2rgb(temp_X);

else

    CLNImg = mat2gray(CLNDoubleImg);

end

end
```

12.3 Multi-Dimensional Fixed Point Algorithm with ATV in MATLAB

```
function [ CLNImg ] = MultiD_ImgDenoise_FPPO_ITV( Img, mu, beta )
%Denoising Program
%Theory by Lixin Shen, Yuesheng Xu, and Charles A. Micchelli
%Implementation by Aaron Katchen

%Fixed Point Algorithm Based on the Proximity Operator with ITV
(FP^20-ITV)
%For the multi-dimensional case

%-------------------------------------------------------

%Setup

%initialize count
count = 1;

%initialize argmin variable
argmin = 1;

%Determine information about noisy image
%Read Image and convert to double
temp_X = imread(Img);

%separate the R, G, and B channels and convert to double

r = temp_X(:, :, 1);
g = temp_X(:, :, 2);
b = temp_X(:, :, 3);

r = im2double(r);
g = im2double(g);
b = im2double(b);

%Determine size of image
[M, N] = size(r);

%create and initialize v matrices
Vr = zeros(2*M, N);
Vg = zeros(2*M, N);
Vb = zeros(2*M, N);

%determine lambda
lambda = 1/4*(sin((N-1)*pi/(2*N)))^(-2);
```

```
%determine alpha
alpha = mu / lambda;

%-----------------------------------------------------

%determine x and y differentials
Br = YXGradient(r);
Bg = YXGradient(g);
Bb = YXGradient(b);

%-----------------------------------------------------

%Initialize BtV
BtVr = zeros(M,N);
BtVg = zeros(M,N);
BtVb = zeros(M,N);

%Initialize BBtV
BBtVr = zeros(2*M, N);
BBtVg = zeros(2*M, N);
BBtVb = zeros(2*M, N);

%Initialize Terms for Error Calculation
Curr_Term = zeros(M,N,3);
Prev_Term = zeros(M,N,3);

%Initialize Clean Double Image Out
CLNDoubleImg = zeros(M,N,3);

%-----------------------------------------------------

%loop fixed point algorithm
while argmin>beta;

    %count
    count = count + 1

    %if there are more than 10 iterations, stop the code
    if count>10;
        break;
    end

    %Store old BtV for relative error calc
    BtVr_Prev = BtVr;
    BtVg_Prev = BtVg;
    BtVb_Prev = BtVb;

    %Calculate gamma
    gamma_r = Br + Vr - lambda * BBtVr;
    gamma_g = Bg + Vg - lambda * BBtVg;
    gamma_b = Bb + Vb - lambda * BBtVb;

    %Calculate individual proximity operators
    ProxOp_r = L1ProxOp(gamma_r, alpha);
    ProxOp_g = L1ProxOp(gamma_g, alpha);
    ProxOp_b = L1ProxOp(gamma_b, alpha);
```

```
    %Calculate new V
    Vr = gamma_r - ProxOp_r;
    Vg = gamma_g - ProxOp_g;
    Vb = gamma_b - ProxOp_b;

    %Determine new B' of v
    BtVr = CalcAdjoint(Vr);
    BtVg = CalcAdjoint(Vg);
    BtVb = CalcAdjoint(Vb);

    %Determine new BBtV
    BBtVr = YXGradient(BtVr);
    BBtVg = YXGradient(BtVg);
    BBtVb = YXGradient(BtVb);

    %Calc Error terms
    Curr_Term(:,:,1) = r - lambda * BtVr;
    Curr_Term(:,:,2) = g - lambda * BtVg;
    Curr_Term(:,:,3) = b - lambda * BtVb;

    Prev_Term(:,:,1) = r - lambda * BtVr_Prev;
    Prev_Term(:,:,2) = g - lambda * BtVg_Prev;
    Prev_Term(:,:,3) = b - lambda * BtVb_Prev;

    %Determine Relative Errors
    argmin_r = norm(Curr_Term(:, :, 1) - Prev_Term(:, :, 1)) /
norm(Curr_Term(:, :, 1));
    argmin_g = norm(Curr_Term(:, :, 2) - Prev_Term(:, :, 2)) /
norm(Curr_Term(:, :, 2));
    argmin_b = norm(Curr_Term(:, :, 3) - Prev_Term(:, :, 3)) /
norm(Curr_Term(:, :, 3));

    %take infinity norm of argmins
    argmin = norm([argmin_r; argmin_g; argmin_b], inf);

end

%-------------------------------------------------------

%Set Ouput image and convert to uint8
CLNDoubleImg(:,:,1) = r - lambda * BtVr;
CLNDoubleImg(:,:,2) = g - lambda * BtVg;
CLNDoubleImg(:,:,3) = b - lambda * BtVb;

CLNImg = im2uint8(CLNDoubleImg);

end
```

12.4 Multi-Dimensional Fixed Point Algorithm with ITV in MATLAB

```
function [ CLNImg ] = MultiD_ImgDenoise_FPPO_ITV( Img, mu, beta )
%Denoising Program
%Theory by Lixin Shen, Yuesheng Xu, and Charles A. Micchelli
%Implementation by Aaron Katchen

%Fixed Point Algorithm Based on the Proximity Operator with ITV
(FP^20-ITV)
%For the multi-dimensional case
```

```
%-----------------------------------------------------

%Setup

%initialize count
count = 1;

%initialize argmin variable
argmin = 1;

%Determine information about noisy image
%Read Image and convert to double
temp_X = imread(Img);

%separate the R, G, and B channels and convert to double

r = temp_X(:, :, 1);
g = temp_X(:, :, 2);
b = temp_X(:, :, 3);

r = im2double(r);
g = im2double(g);
b = im2double(b);

%Determine size of image
[M, N] = size(r);

%create and initialize v matrices
Vr = zeros(2*M, N);
Vg = zeros(2*M, N);
Vb = zeros(2*M, N);

%determine lambda
lambda = 1/4*(sin((N-1)*pi/(2*N)))^(-2);

%determine alpha
alpha = mu / lambda;

%-----------------------------------------------------

%determine x and y differentials
Br = YXGradient(r);
Bg = YXGradient(g);
Bb = YXGradient(b);

%-----------------------------------------------------

%Initialize BtV
BtVr = zeros(M,N);
BtVg = zeros(M,N);
BtVb = zeros(M,N);

%Initialize BBtV
BBtVr = zeros(2*M, N);
BBtVg = zeros(2*M, N);
BBtVb = zeros(2*M, N);
```

```
%Initialize Terms for Error Calculation
Curr_Term = zeros(M,N,3);
Prev_Term = zeros(M,N,3);

%Initialize Clean Double Image Out
CLNDoubleImg = zeros(M,N,3);

%-----------------------------------------------------

%loop fixed point algorithm
while argmin>beta;

    %count
    count = count + 1

    %if there are more than 10 iterations, stop the code
    if count>10;
        break;
    end

    %Store old BtV for relative error calc
    BtVr_Prev = BtVr;
    BtVg_Prev = BtVg;
    BtVb_Prev = BtVb;

    %Calculate gamma
    gamma_r = Br + Vr - lambda * BBtVr;
    gamma_g = Bg + Vg - lambda * BBtVg;
    gamma_b = Bb + Vb - lambda * BBtVb;

    %Calculate individual proximity operators
    ProxOp_r = L2ProxOp(gamma_r, alpha);
    ProxOp_g = L2ProxOp(gamma_g, alpha);
    ProxOp_b = L2ProxOp(gamma_b, alpha);

    %Calculate new V
    Vr = gamma_r - ProxOp_r;
    Vg = gamma_g - ProxOp_g;
    Vb = gamma_b - ProxOp_b;

    %Determine new B' of v
    BtVr = CalcAdjoint(Vr);
    BtVg = CalcAdjoint(Vg);
    BtVb = CalcAdjoint(Vb);

    %Determine new BBtV
    BBtVr = YXGradient(BtVr);
    BBtVg = YXGradient(BtVg);
    BBtVb = YXGradient(BtVb);

    %Calc Error terms
    Curr_Term(:,:,1) = r - lambda * BtVr;
    Curr_Term(:,:,2) = g - lambda * BtVg;
    Curr_Term(:,:,3) = b - lambda * BtVb;

    Prev_Term(:,:,1) = r - lambda * BtVr_Prev;
    Prev_Term(:,:,2) = g - lambda * BtVg_Prev;
    Prev_Term(:,:,3) = b - lambda * BtVb_Prev;
```

```
    %Determine Relative Errors
    argmin_r = norm(Curr_Term(:, :, 1) - Prev_Term(:, :, 1)) /
norm(Curr_Term(:, :, 1));
    argmin_g = norm(Curr_Term(:, :, 2) - Prev_Term(:, :, 2)) /
norm(Curr_Term(:, :, 2));
    argmin_b = norm(Curr_Term(:, :, 3) - Prev_Term(:, :, 3)) /
norm(Curr_Term(:, :, 3));

    %take infinity norm of argmins
    argmin = norm([argmin_r; argmin_g; argmin_b], inf);

end

%-----------------------------------------------------

%Set Output image and convert to uint8
CLNDoubleImg(:,:,1) = r - lambda * BtVr;
CLNDoubleImg(:,:,2) = g - lambda * BtVg;
CLNDoubleImg(:,:,3) = b - lambda * BtVb;

CLNImg = im2uint8(CLNDoubleImg);

end
```

12.5 ATV Proximity Operator

```
function [ Y ] = L1ProxOp( X, alpha)
%Calculate the L1 Proximity Operator of input matrix X
%   Calculate the Proximity Operater of Input Matrix X

%-----------------------------------------------------

Y=max(abs(X)-alpha,0).*sign(X);
```

12.6 ITV Proximity Operator

```
function [ Y ] = L2ProxOp( X, alpha)
%Calculate the L2 Proximity Operator of input matrix X
%   Calculate the Proximity Operator of Input Matrix X

%-----------------------------------------------------
%determine size of X
[F, N] = size(X);

%Set M equal to half the size of X's rows
M = F / 2;

%initialize X2
X2=zeros(F,N);

%take the norm of corresponding pixels
for i=1:M;
    for j=1:N;
        X2(i,j)=norm([X(i,j); X(i+M,j)]);
    end
end
```

```
%copy top part of X2 to bottom part
X2(M+1:F, :)=X2(1:M, :);

%create index of where zeros are in X2 to avoid division by zero
Index = (X2 == 0);

%create Denominator Matrix Term
Denom = (1 - Index) .* X2 + Index * (10e-10);

%Calculate ProxOp and set Output
Y = max(X2-alpha,0) .* (X ./ Denom);
```

12.7 Adjoint Calculator

```
function [ Y ] = CalcAdjoint( X )
%This function calculates the adjoint of input matrix X
%Input argument is of size 2M x N
%Output argument is of size M x N

%-------------------------------------------------------

%Determine amount of columns in X
[F, N] = size(X);

M = F/2;

%-------------------------------------------------------

%split X into two matrices
X1 = X(1:M, :);
X2 = X((M+1):2*M, :);

%-------------------------------------------------------

%Calculate Y1 and Y2

%initialize Y1 and Y2 matrices
Y1 = zeros(M, N);
Y2 = zeros(M, N);

%setup the rows of Y1 and the columns of Y2
Y1(M,:) = X1(M-1,:);
Y2(:,N) = X2(:,N-1);

Y1(1, :) = -X1(1,:);
Y2(:, 1) = -X2(:,1);

Y1(2:(M-1),:) = X1(1:(M-2),:) - X1(2:(M-1),:);
Y2(:,2:(N-1)) = X2(:,1:(N-2)) - X2(:,2:(N-1));

Y = Y1 + Y2;
```

12.8 Gradient Calculator

```
function [ Matrix_Out ] = YXGradient( Matrix_In )
```

```
%This function takes in an argument of Matrix_In Size N x N and
outputs
%a gradient of size 2N x N with the first N x N elements
representing the
%Y gradient and the second N x N elements representing the X
gradient

%-------------------------------------------------------

%Determine size of Matrix_In
[M, N] = size(Matrix_In);

%-------------------------------------------------------

%Determine X Differential
diffx=[diff(Matrix_In, 1, 2) zeros(M,1)];

%-------------------------------------------------------

%determine y differential
diffy=[diff(Matrix_In, 1, 1); zeros(1, N)];
%-------------------------------------------------------

%restructure into output matrix
Matrix_Out = [diffy; diffx];

end
```

13. References

[1] P. Blomgren and T. Chan, “Color TV: Total Variation methods for restoration of vector valued images,” *IEEE Transactions on Image Processing*, Vol. 7, No. 3, pp. 304-309, Mar. 1998.

[2] A. Chambolle, “An Algorithm for Total Variation Minimization and Applications,” *Journal of Mathematical Imaging and Vision 20,* pp. 89-97, 2004.

[3] C. A. Micchelli, L. Shen, and Y. Xu, “Proximity Algorithms for Image Models: Denoising,” 27 (2011) 045009 (30 pp).

[4] S. G. Nash and A. Sofer, *Linear and Nonlinear Programming*, McGraw-Hill, 1996.

[5] M. A. Pinsky, *Partial Differential Equations and Boundary-Value Problems with Applications*, Waveland Press, Inc., 1998.

Summary

Efficient noise reduction algorithms for digital image processing have been sought after for as long as digital imaging has existed. The problem lies in digital image sensors, whether that be a point and shoot digital camera, an MRI machine, or satellite imaging equipment, all digital image capture devices must approximate any information which it receives. When a lot of information is available, say a bright sunny day when a lot of light would flood the image sensor, the capture device is much less likely to create noise, however if information is scarce, say a photo outdoors at night, the device must make its best approximation as to what the missing information should look like; this is comparable to grainy photos in the days of film. Although digital image sensors have improved drastically over the years, there still remains a need for noise reduction algorithms to clean up any remaining noise in certain applications; for instance, it may be crucial to reduce the noise in an MRI image to obtain a clear image for patient diagnosis.

This paper discusses how efficient and effective noise reduction may be achieved through the use of mathematic numerical analysis. Numerical analysis is a branch of applied mathematics which uses computers and algorithms to solve problems. Often, real world problems are not explicitly defined functions, meaning there is usually not only one independent variable, and this can create problems when trying to solve

these problems outright. Mathematicians may then employ the power of numerical analysis.

Numerical analysis, first developed in ancient Babylonia, is a form of mathematics used to find approximations of solutions as often the exact solution may be impossible to solve for. Instead of attempting to solve directly for a problem, mathematicians instead create algorithms which approximate the solution within a given tolerance. These algorithms tend to work in an educated ‘guess and check’ methodology. Often, the algorithms will ‘check’ to see if they are within a given tolerance to the answer, if they’re not, they will make a calculated ‘guess’ as to what the answer may be based on the previous ‘guess’. Once this new ‘guess’ is checked the process either stops if it is within the tolerance or continues until it is. Occasionally, the algorithm may not ‘converge’. This means that the algorithm is stuck in an infinite loop and will never find the approximated answer. It is thus important for mathematicians to create algorithms which guarantee convergence as well as converge in an efficient and timely manner. Luckily, due to the modern age and super computers, efficiency is less of an issue than it used to be, as computers can run much more quickly than they used to, but nonetheless it is still a top priority of mathematicians to create efficient algorithms.

The algorithms discussed in the paper are either algorithms created by [3] Micchelli-Shen-Xu or extensions thereof. These algorithms are concerned with reducing the noise in both grayscale (black and white) as

well as color (red, blue, green) images. The basic grayscale algorithm works by first analyzing the image. When first analyzing the image, the algorithm calculates the luminosity (brightness) differences between each pixel. The algorithm does this to find what may be noisy pixels. For instance, if the image only sees very small changes in luminosity between pixels, we can assume this to be a smooth area of the image, with no sharp edges or noise; if the image sees a large change in luminosity between pixels then we have either discovered a sharp edge or a noisy pixel. By comparing the results with pixels around it, the algorithm can determine if that is likely an edge or a noisy pixel; if it's an edge, then all the other pixels along that edge would likely experience a similar change, if it's a noisy pixel then it's likely that that large change would only be experienced by the sole pixel under analysis. If the pixel is indeed considered to be noisy, the algorithm attempts to select a new, less noisy, value for the pixel based on the surrounding pixel values. This algorithm should ensure optimal noise reduction while retaining sharp edges.

The original algorithm, proposed by Micchelli-Shen-Xu [3], is then slightly modified to apply to color images. The paper discusses a few different ways this may be achieved. First, it should be understood that all color images can be broken apart into different colorspaces, or primary components. We are often taught that red, yellow, and blue are the primary colors, but that is just in one specific color model. Color may also be broken into cyan, magenta, yellow, and black (CMYK); red, green, and

blue (RGB) which is how the human eye and computer monitors interpret color; luminosity, red-difference chroma, and blue-difference chroma (YCbCr) often used in video; and so forth. The paper deals primarily with RGB and YCbCr colorspaces as these are usually the only colorspaces used in digital imaging. One algorithm applies the basic grayscale algorithm to the luminosity channel of a YCbCr image while another algorithm applies the basic grayscale algorithm to each channel of the RGB colorspace individually. Both algorithms provide acceptable, though slightly different, results.

The algorithms were implemented using MATLAB script and experiments were ran to test their effectiveness. The algorithms and experimental outcomes may be found toward the end of paper in sections 12 and 10 respectively.