Comparing 2D Total Variation Denoising and Image Denoising using Deep Image Priors

Philip Hopen
Tanmay Agarwal
Pooja Shah
Abhijit Mahishi
Andrew Xie

1. Introduction:

For this project, we will use ADMM to develop a solution to the 2D total variation denoising problem introduced in class. In general, denoising algorithms are applied to "noisy" signals (where signal elements deviate from their true values with some random error) in an attempt to approximate the ground-truth, "clean" signal from which the noisy input is derived. In the case of 2D images, denoising methods can be useful for improving image quality by reducing errors caused by lighting and camera conditions.

We first establish the 1D version of the total variation denoising problem:

$$\underset{\mathbf{x}}{\text{minimize}} \qquad \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|_2^2 + \lambda|\nabla\mathbf{x}|,$$

$$where\ |\nabla\mathbf{x}| = \textstyle\sum_{i=1}^{N-1} |x_{i+1}-x_i|, for\ \mathbf{x},\mathbf{y} \in \mathbb{R}^N\ \ (\mathbf{y}\ is\ the\ noisy\ signal)$$

This can be rewritten by replacing ∇x with the value Dx:

$$\min_{x} \quad \frac{1}{2}\|x-y\|_2^2 + \lambda\|Dx\|_1$$

where matrix D is the transformation on x which returns the disparity vector ∇x .This equation can then be rewritten as a conditional optimization problem by assigning the new variable $z = Dx$:

$$\begin{aligned} \min_{x} \quad & \frac{1}{2}\|x-y\|_2^2 + \lambda\|z\|_1 \\ \text{s.t.} \quad & z = Dx \end{aligned}$$

With the above equation, we can then derive the dual problem:

$$\underset{\boldsymbol{\gamma}\in\mathbb{R}^{N-1}}{\text{minimize}} \qquad \frac{1}{2}\|\mathbf{y}-\mathbf{D}^T\boldsymbol{\gamma}\|_2^2$$

$$\text{subject to} \qquad \|\boldsymbol{\gamma}\|_\infty \le \lambda.$$

$$where \nabla\mathbf{x} = \mathbf{D}\mathbf{x}$$

We are able to solve this dual problem using proximal gradient descent., plugging the resulting ɣ value into the equation:

$$\mathbf{x}_* = \mathbf{y} - \mathbf{D}^T\boldsymbol{\gamma}_*$$

in order to produce our primal solution. We have already explored this implementation of 1D denoising in a previous course assignment. With this baseline understanding, we now look to apply this framework to the 2D denoising setting.

In order to demonstrate the efficacy of the ADMM denoising implementation, we use other well-known denoising techniques to generate baseline results on noisy images. Performance is measured by taking the sum of squared differences between the clean images and the different denoising method outputs.

2. Total Variation Denoising with ADMM:

Unlike the 1D version of total variation denoising, 2D total variation denoising requires a different approach due to the total variation penalty having two terms instead of one:

$$\underset{\mathbf{x}}{\text{minimize}} \qquad \frac{1}{2}\|\mathbf{Y}-\mathbf{X}\|_F^2 + \tau|\nabla\mathbf{X}_{i,j}|$$
$$where|\nabla\mathbf{X}_{i,j}| = |\mathbf{X}_{i+1,j} - \mathbf{X}_{i,j}| + |\mathbf{X}_{i,j+1} - \mathbf{X}_{i,j}|$$

The Alternating Direction Method of Multipliers is a method that is commonly used to split an optimization problem into separate sub-problems that are individually easier to solve than the entire problem as a whole. In other words, ADMM is a divide-and-conquer approach to optimization, where we minimize multiple objectives one after another. The associated variables are updated in an alternating or sequential fashion, which accounts for the term alternating direction. We minimize one of the variables while keeping the others constant, with the condition that variables converge to a single solution, as shown below:

$$\begin{aligned} \min_{x,y} \quad & f(x) + g(y) \\ \text{s.t.} \quad & x = y \end{aligned}$$

ADMM problems can also be written in the general form:

$$\begin{aligned} \min_{x,y} \quad & f(x) + g(y) \\ \text{s.t.} \quad & Ax + By + c = 0 \end{aligned}$$

The ADMM method assigns one of the total variation penalty terms to a new variable Z and splits up the overall problem into two subproblems for X and Z. We can rewrite the problem with the augmented Lagrangian as:

$$L(X,Z,\Gamma) = \frac{1}{2}\|Y-X\|_F^2 + \frac{t}{2}\|X-Z+\Gamma\|_F^2 + \tau\sum_{i=1}^{M-1}\sum_{j=1}^{N}|X_{i+1,j}-X_{i,j}| + \tau\sum_{i=1}^{M}\sum_{j=1}^{N-1}|Z_{i,j+1}-Z_{i,j}|$$

The first step towards solving our total variation denoising problem is solving the X sub-problem:

$$\frac{1}{2}\|X-Y\|_F^2 + \frac{t}{2}\|X-Z+\Gamma\|_F^2 + \tau\sum_{i=1}^{M-1}\sum_{j=1}^{N}|X_{i+1,j}-X_{i,j}|$$

Expanding and simplifying, we get:

$$= \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{N}((X_{i,j}-Y_{i,j})^2 + t(X_{i,j}-Z_{i,j}+\Gamma_{i,j})^2) + \tau\sum_{i=1}^{M-1}\sum_{j=1}^{N}|X_{i+1,j}-X_{i,j}|$$

$$= \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{N}(X_{i,j}^2 - 2X_{i,j}Y_{i,j} + Y_{i,j}^2 + t(X_{i,j}^2 - 2X_{i,j}(Z_{i,j}+\Gamma_{i,j}) + (Z_{i,j}+\Gamma_{i,j})^2)) + \tau\sum_{i=1}^{M-1}\sum_{j=1}^{N}|X_{i+1,j}-X_{i,j}|$$

$$= \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{N}(X_{i,j}^2 - \frac{2}{t+1}X_{i,j}(Y_{i,j}-t(Z_{i,j}+\Gamma_{i,j})) + \frac{1}{t+1}(Y_{i,j}^2 + t(Z_{i,j}+\Gamma_{i,j})^2)) + \frac{\tau}{t+1}\sum_{i=1}^{M-1}\sum_{j=1}^{N}|X_{i+1,j}-X_{i,j}|$$

$$= \frac{1}{2}\|X - \frac{1}{t+1}(Y-t(Z+\Gamma))\|_F^2 + \frac{\tau}{t+1}\sum_{i=1}^{M-1}\sum_{j=1}^{N}|X_{i+1,j}-X_{i,j}|$$

$$= \sum_{j=1}^{N}(\frac{1}{2}\|X_j - \frac{1}{t+1}(Y_j - t(Z_j+\Gamma_j))\|_2^2 + \frac{\tau}{t+1}|\nabla X_j|)$$

Note that this is just the 1D total variation denoising problem summed over the columns of the input matrix $\frac{1}{t+1}(Y-t(Z+\Gamma))$. For each column, we can simply solve the dual problem:

$$\begin{aligned} \min_{\gamma} \quad & \frac{1}{2}\|\frac{1}{t+1}(Y_j - t(Z_j+\Gamma_j)) - D^T\gamma\|_2^2 \\ \text{s.t.} \quad & \|\gamma\|_\infty < \frac{\lambda}{t+1} \end{aligned}$$

Now for the Z sub-problem**:**

$$\frac{t}{2}\|X - Z + \Gamma\|_F^2 + +\tau \sum_{i=1}^{M}\sum_{j=1}^{N-1} |Z_{i,j+1} - Z_{i,j}|$$

We simplify the problem in the same manner as in the X sub-problem and see that it is the 1D denoising problem applied over the rows of input matrix $Z + \Gamma$:

$$\sum_{i=1}^{M}(\frac{1}{2}\|X_i - (Z_i + \Gamma_i)\|_2^2 + \frac{\tau}{t}|\nabla X_i|)$$

Which we solve with the dual solution:

$$\begin{aligned} \min_{\gamma} \quad & \frac{1}{2}\|(Z_i + \Gamma_i) - D^T\gamma\|_2^2 \\ \text{s.t.} \quad & \|\gamma\|_\infty < \frac{\lambda}{t} \end{aligned}$$

for each row.

Having shown that each of these problems can be reduced to a summation of the 1D denoising over rows/columns, we are able to apply our existing solution to 1D Total Variation Denoising via proximal gradient descent accordingly. We incorporate these sub-problem solutions into the larger ADMM solution over many iterations via the consensus term Γ.

3. Deep Image Priors:

Deep image priors use a type of neural network that is trained to reconstruct an image from its latent representation, rather than being trained to recognize specific image classes or patterns. The idea behind deep image priors is that the structure of natural images is highly structured and can be effectively captured by a convolutional neural network (CNN) through training on a large dataset of images.

Given a noisy image x_0 we can find a denoised version x^* through the following minimization:

$$x^* = \min_x \left[E(x, x_0) + P(x)\right]$$

where $E(x, x_0)$ is a data term that measures how close x is to x_0 and $P(x)$ is the prior term.
Since there's no good way to come up with a prior, we use representations from random CNNs as priors, hence the name Deep Image Priors. Concretely, let the network be $f_\theta(\eta)$ where $f_\theta(\eta)$ are parameters of the network and η is an input. The input is sampled from a standard Gaussian distribution and kept fixed. Now, we can denoise x_0 as follows:

$$\theta^* = \min_\theta \left[E(f_\theta(\eta), x_0)\right]; x^* = f_{\theta^*}(\eta)$$

To use a deep image prior, an input image is passed through the network and the resulting latent representation is used to reconstruct the image. The reconstruction process is iterative, with the network adjusting the latent representation at each iteration until the reconstructed image matches the input image as closely as possible. The network is then able to use the learned structure of natural images to fill in missing or corrupted parts of the input image, or to enhance the image in some way.

The optimization objective here closely resembles the approximation of the task which is minimizing the reconstruction error. We use the L1 objective function (mean squared error) from the PyTorch package and minimize it using the Adam optimizer.

We construct the deep image priors using an Encoder-Decoder neural network with 3 encoder blocks and 3 encoder blocks to first do a lossy compression on the image and then try to reconstruct it using the decoder blocks. We also apply LeakyRELU functions to induce non-linearity and Batch Normalization to normalize the inputs in each batch.

4. Model Architecture:

```
class EncDec(nn.Module):
    def __init__(self):
        super(EncDec, self).__init__()
        self.enc1 = nn.Sequential(
            nn.Conv2d(in_channels=1,
                      out_channels=16,
                      kernel_size=3,
                      stride=2, padding=1),
            nn.LeakyReLU(),
            nn.BatchNorm2d(num_features=16),
        )

# Two more encoder blocks –
        self.dec1 = nn.Sequential(
            nn.Upsample(scale_factor=2, mode='bilinear'),
            nn.Conv2d(in_channels=64,
                      out_channels=32,
                      kernel_size=3,
                      stride=1, padding=1),
            nn.LeakyReLU(),
            nn.BatchNorm2d(num_features=32),
        )
# Two more decoder blocks –
        self.up = nn.Upsample(scale_factor=2, mode='bilinear')
```

5. **Training Loop and optimization:**

```
for i in tqdm(range(n_epochs)):
    adam.zero_grad()

    # Forward pass through the network
    out_eta = model(eta)

    # Compute the average image at 100-epoch intervals
    if i in average:
      avg_img += out_eta.clone()[0, 0, :,
:].transpose(0,1).detach().numpy()

    # Compute the training loss
    loss_train = mse(out_eta, noisy_img)
    train_error.append(np.log(loss_train.item()))

    # Compute the test loss
    loss_test =
mean_squared_error(out_eta.clone().detach().cpu().numpy()[0, 0, :, :],
clean_img.detach().cpu().numpy()[0, 0, :, :])
    test_error.append(np.log(loss_test))

    # Backpropagation and optimization step
    loss_train.backward()
    adam.step()
```

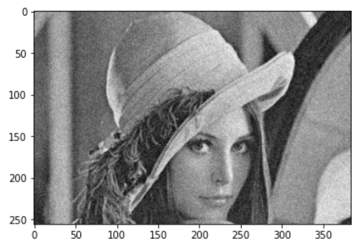
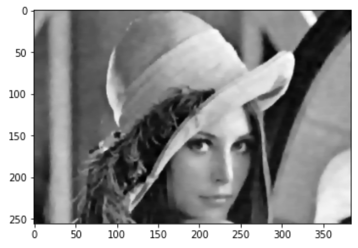
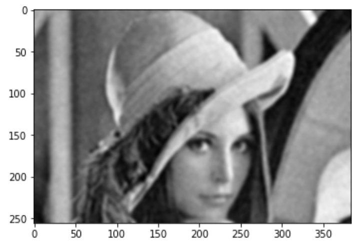
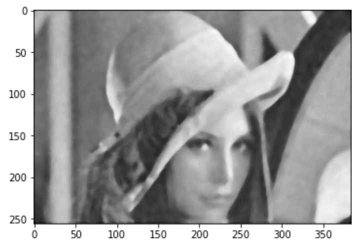
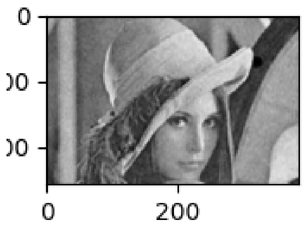
6. **Other Methods:**

We are also using Standard Gaussian Filtering and Median Filtering to compare our 2D TV Denoising and Deep Image Prior Denoising with.

Gaussian filtering involves performing a weighted average of the surrounding pixels based on the Gaussian distribution. It is used to remove Gaussian noise and is a realistic model of defocused lens.

Median filtering involves replacing the center pixel of a M × M neighborhood by the median value of the corresponding window.

7. Results:

	Standard Error	Result
Noisy Image	338.54	
TV using MC ADMM	1.3	
Gaussian Filtering	477.14	
Median Filtering	517.99	
Deep Image Priors	138.74	

8. Conclusion:

Based on the above standard error results for each 2D total variation denoising method, we can conclude that total variation using MC ADMM algorithm has the best result with the lowest standard error. On the contrary, Median Filtering gave the worst results.

Deep Image Priors is a very non-deterministic method that doesn't guarantee convergence because we only run it for an arbitrary number of iterations with an arbitrary model architecture. We could probably squeeze out better results if we experiment with hyperparameters, but that is outside the scope of this project. According to the literature, the advantage of this method over the others is that it can likely generalize well on other tasks but isn't the best approach when we have a clear optimization objective which can be better approximated by handcrafted objectives designed to solve a specific task like image denoising.

Naïve methods like Gaussian filtering are more effective at smoothing images in general but due to the randomness in noise it will not always produce a good result on the standard error metric that we used. Similarly, Median Filtering is effective at removing noise while preserving edges, expectedly it's not optimizing for a metric like the error term we have constructed.

These make it pretty clear why using good optimization routines or gradient based approaches work better than naïve smoothing algorithms.

As a future scope to our problem statement, we can try out different methods of image denoising. Recently, total variation denoising has lost popularity in favor of neural networks, which are able to process large amounts of data. This method involves using deep learning methods to train a denoiser, which will ideally be able to detect and remove noise from any given picture by itself. Future projects can test whether these denoisers can be trained accurately enough such that their standard error is close to or lower than that of our 2D ADMM total variation.

Other possible methods for future related projects include:

- Non-local means denoising
- GANs
- Variational Autoencoders

We can stratify these methods as ones that require large amounts of training data but also generalize well vs. methods that work out of the box but need a lot of handcrafting and manual input.

Link to Code on Google Colab:
https://colab.research.google.com/drive/1wBA_WBfs2qM_y-PR7tVeijJ55Nop1OYD

Link to DIP Code:
https://github.com/tanmay7270/image_denoising_590OP