AMATH 515: HOMEWORK 7 REPORT

SID MEKA

Applied Mathematics Department, University of Washington, Seattle, WA
gameka2k@uw.edu

1. Introduction

Here, we use image inpainting through methods of Scientific Computing through `scipy.fft`, Optimization through `cvxpy`, Numerical Computing through `numpy`, and Computational Vision and Image Processing through `scikit-image`. Furthermore, we have the goal of developing code through this image inpainting technique in order to computationally recover the a damaged version of the painting *Son of Man* by Rene Magritte. It is worth noting that the corruption arises from missing pixel values represented by a binary masking matrix, M. The key idea behind this code or algorithm is that natural images are often sparse in an appropriate transform basis, such as the discrete cosine transform, or in simpler abbreviation, DCT. By formulating this problem as a convex optimization problem with an ℓ_1 regularization term, we aim to leverage this sparsity to recover the missing pixels. Ultimately, we utilize a masked least squares problem employing the DCT and iDCT using proximal gradient descent; we further verify the solution using `cvxpy` and evaluate performance using the relative mean squared error.

2. Theory

(1) Derive an expression for $\text{prox}_{\lambda\psi}(x)$ when $\psi(x) = \|x\|_1$. Show your work.

Problem Setup: We are tasked with computing:

$$\text{prox}_{\lambda\psi}(x) = \arg\min_z \frac{1}{2}\|x - z\|_2^2 + \lambda\|z\|_1$$

where:

- $x \in \mathbb{R}^n$ is a given vector
- We have $\psi(z)$ as so:
  - $\psi(z) = \|z\|_1$
  - $\|z\|_1 = \sum_{i=1}^n |z_i|$
  - We have that $\psi(z)$ is the ℓ_1 norm of the vector z

$\lambda > 0$ is a regularization parameter

Now, we solve with our steps:

(a) We can write

$$\text{prox}_{\lambda\psi}(x) = \arg\min_z \frac{1}{2}\|x - z\|_2^2 + \lambda\|z\|_1$$

as

$$\text{prox}_{\lambda\psi}(x) = \arg\min_z \left(\frac{1}{2}\sum_{i=1}^n (x_i - z_i)^2 + \sum_{i=1}^n \lambda|z_i|\right)$$

Date: March 19, 2025.

Note that here ψ can also be seen as $\|\cdot\|_1$, which we can use and think about when solving this problem.

(b) For a fixed component i, the problem becomes:

$$\min_{z_i} \frac{1}{2}(x_i - z_i)^2 + \lambda |z_i|$$

We now solve this by considering the subdifferential:

(i) If $z_i > 0$, the derivative is: $z_i - x_i + \lambda = 0$, which means $z_i = x_i - \lambda$.
(ii) If $z_i < 0$, the derivative is: $z_i - x_i - \lambda = 0$, which means $z_i = x_i + \lambda$.
(iii) If $z_i = 0$, the subdifferential contains zero when we have $-\lambda \le x_i \le \lambda$.

Therefore, the solution is:

$$z_i = \begin{cases} x_i - \lambda & \text{when } x_i > \lambda \\ x_i + \lambda & \text{when } x_i < -\lambda \\ 0 & \text{when } -\lambda \le x_i \le \lambda \end{cases}$$

(c) Expressing $\text{prox}_{\lambda\psi}(x)$:
This means:

$$\text{prox}_{\lambda\psi}(x) = \begin{cases} x - \lambda & \text{when } x > \lambda \\ x + \lambda & \text{when } x < -\lambda \\ 0 & \text{when } -\lambda \le x \le \lambda \end{cases}$$

(2) Let $X, Y \in \mathbb{R}^{N_y \times N_x}$. We think of X as the DCT of an image which we want to find, and Y as the true/uncorrupted image. Define the objective

$$f(x) = \frac{1}{2}\|M \odot (\text{iDCT}(X) - Y)\|_F^2$$

Let $x = \text{vec}(X)$ and $y = \text{vec}(Y)$, where $\text{vec}(\cdot)$ stacks the columns of the matrix into a single vector. Define a diagonal matrix D_M from the mask: $D_M = \text{diag}(\text{vec}(M))$. Now, the masked objective becomes: $f(x) = \frac{1}{2}\|D_M(A^{-1}x - y)\|_2^2$. Now, that the Hadamard product is not present anymore we have that:

$$\nabla f(x) = A^{-T} D_M (A^{-1}x - y)$$
$$\nabla f(x) = A D_M (A^{-1}x - y)$$

Or in other words:

$$\nabla f(X) = \text{DCT}(M \odot (\text{iDCT}(X) - Y))$$

Note that A is unitary as the DCT function is defined using cosine basis functions, which are orthogonal, and A is constructed from these orthogonal basis functions normalized to have unit length leading giving A to be unitary. Because A is unitary, we have: $A^{-1} = A^T$.

(3) Derive an upper bound for the Lipschitz constant of ∇f.

$$L = \|A^T D_M A^{-1}\|_2$$

(By Cauchy-Schwarz Inequality) $\quad L \le \|A^T\|_2 \cdot \|D_M\|_2 \cdot \|A^{-1}\|_2$

(As $\|A^T\|_2 = 1$ and $\|A^{-1}\|_2 = 1$ as A is unitary) $\quad L \le \|D_M\|_2$

(As D_M is a diagonal matrix with entries 0 or 1) $\quad \|D_M\|_2 \le 1$

(From $L \le \|D_M\|_2$ and $\|D_M\|_2 \le 1$) $\quad L \le 1$

Thus, because $L \le 1$, we bound our Lipschitz Constant or L at 1.

3. Methods

3.1. **Problem Setup.** We aim to solve the following convex optimization problem to reconstruct a corrupted image:

$$\min_X \quad \frac{1}{2}\left\|M \odot (\text{iDCT}(X) - Y)\right\|_2^2 \; + \; \lambda \left\|\text{vec}(X)\right\|_1$$

We can consider X to be the DCT of an image which we want to find and Y to be the true or uncorrupted image. Furthermore, we have M as a binary masking matrix with entries in $\{0,1\}$, where 0 represents missing pixels. We have our regularization parameter as well, which is λ.

3.2. **Gradient Calculation.** To compute the gradient, we first vectorize the following problem:

$$f(x) = \frac{1}{2}\|D_M(A^{-1}x - y)\|_2^2$$

where $x = \text{vec}(X)$ and $y = \text{vec}(Y)$ and this allows for stacking by flattening our matrices X and Y, $D_M = \text{diag}(\text{vec}(M))$ is a diagonal matrix constructed from the mask, and A and A^{-1} represent the DCT and iDCT matrices, respectively. We have that from the theory, $\nabla f(x) = AD_M(A^{-1}x - y)$ and $\nabla f(X) = \text{DCT}(M \odot (\text{iDCT}(X) - Y))$, and we implement this in the code.

3.3. **Proximal Gradient Descent Implementation.** To solve the problem, we implemented a proximal gradient descent algorithm. The update step is defined as: $x_{k+1} = \text{prox}_{t\lambda\|\cdot\|_1}(x_k - t\nabla f(x_k))$, where the step size is $t = \frac{1}{L}$, where we have our Lipschitz constant, L as $L = 1$ and we have derived $\text{prox}_{\lambda\psi}(x)$ in our Theory.

3.4. **Using CVXPY.** We also implemented `CVXPY` by importing `cvxpy` as `cvx`. The objective function was defined as

```
my_objective_function = cvx.Minimize(0.5 *
cvx.sum_squares(Dmask @ (inv_dctMat @ xxxtentacion - img.flatten())) +
regularization_parameter * cvx.norm(xxxtentacion, 1))
```

It is worth noting that these new lines are not there in the Jupyter Notebook file, but are here in the Report, in order to properly show the code. We have `Dmask` as the diagonal matrix from the mask and `inv_dctMat` as the matrix representing iDCT. The variable `xxxtentacion` is defined as a `cvx.Variable` object of size `Ny*Nx`. Initially, it holds a `None` value as it has not been computed. After defining the optimization problem using `cvx.Problem`, the solver computes the optimal values when `problem_DrewMcIntyre`, which is defined by `problem_DrewMcIntyre = cvx.Problem(my_objective_function)`, is solved with the `solve` command. Once the solver finishes, `xxxtentacion.value` contains the optimal solution with the shape `(999,)`.

3.5. **Relative Mean Squared Error.** We solve for Relative Mean Squared Error by having

```
relative_mse =
np.mean(((1-mask)*(reconstruction - img))**2)/np.var(((1-mask)*img)**2)
```

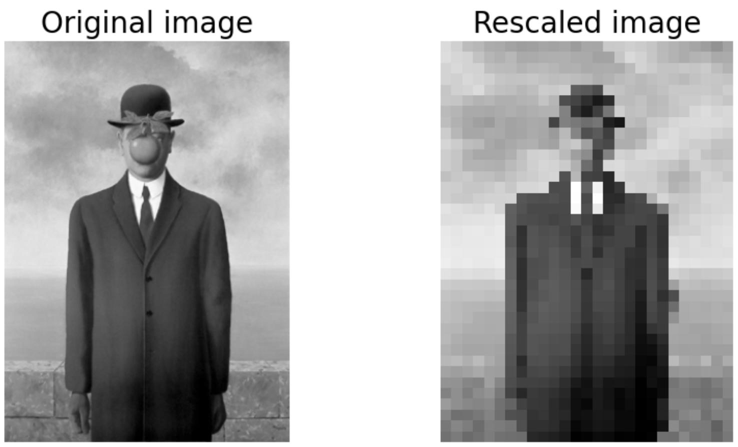
where

```
reconstruction = idctn(sol_pgd, norm="ortho")
```

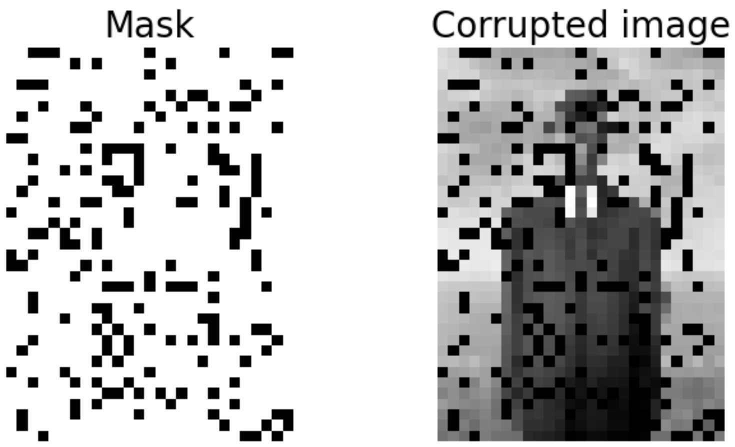
and `sol_pgd` utilizes our Proximal Gradient Descent Solver. Once again, it is worth noting that these new lines are not there in the Jupyter Notebook file, but are here in the Report, in order to properly show the code.

4. Results

We first loaded in the image and downsampled it so that `CVXPY` can solve the problem in a reasonable amount of time. We see that the original size of the image is $(374, 266)$ and the rescaled size is $(37, 27)$. This is the resulting image we get:



We then generate a mask and use that to show our corrupted image:

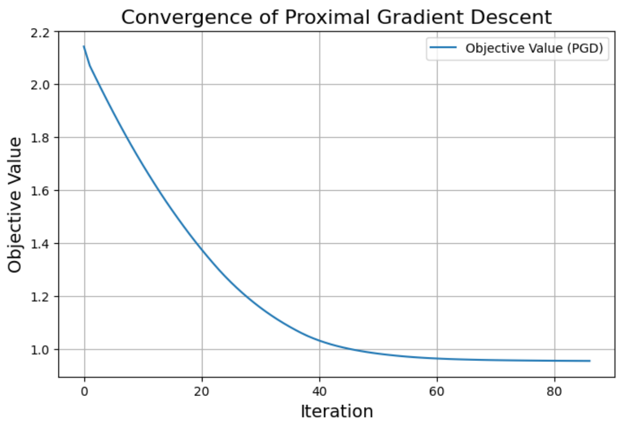


After defining the optimization problem using `CVXPY`, the solver computed the optimal solution by solving the convex objective function. The solution was stored in the variable `xxxtentacion`, which initially held a `None` value until the solver was executed. After solving the associated problem, the computed solution had a final shape of `(999,)`. This solution was then reshaped to the original image dimensions and processed using the inverse discrete cosine transform by using iDCT to reconstruct the image.

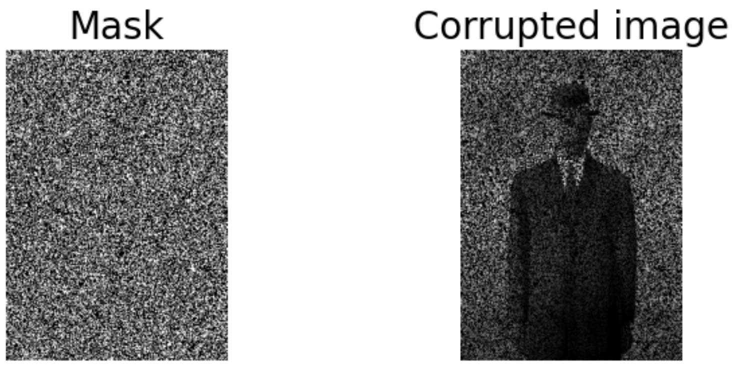
We then move on with our reconstruction:

```
reconstruction = idctn(sol.reshape(Ny,Nx),norm = 'ortho')
relative_mse =
np.mean(((1-mask)*(reconstruction - img))**2)/np.var(((1-mask)*img)**2)
```

Note that some the equation for `relative_mse` is split into two lines in order to show it in this report. We get `relative_mse` as 0.04792328392931286. We then get after applying our Proximal Gradient Descent solve that we converge at iteration 85 with objective 0.955262 with a Mean Squared Error of 0.0502605849843389. Although the Mean Squared Error is slightly increased after running and applying our Proximal Gradient Descent solve, the consistent decrease in the objective value through iterations indicates that the optimization algorithm is effectively minimizing the reconstruction error and converging toward an optimal solution. This is buttressed with the following plot:



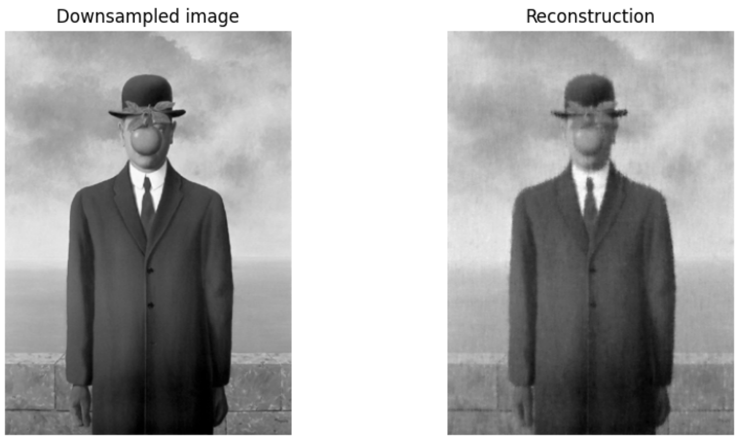
After rerunning on the full size image with more masking, we get that the rescaled size is $(374, 266)$. We are provided with the mask and corrupted image:



We finally get to where we are essentially solving the convex optimization problem and performing the following steps:

(1) Initial Guess with DCT: We create an initial guess for the solution by computing the DCT of the corrupted image `masked_image` using `dctn`. This provides a starting point for the optimization process because natural images are often sparse in the DCT domain.
(2) Build Objective Components
(3) Solve with Proximal Gradient Descent
(4) Reshape Solution Back to Matrix: After solving, we reshape the flattened solution `sol_pgd` back into the matrix form matching the dimensions of the original image.
(5) Apply iDCT: We apply the iDCT to convert the solution from the frequency domain back to the spatial domain. This effectively gives us the reconstructed image.
(6) Compute Mean Squared Error
(7) Plot Results: We plot results by comparing the downsampled image with the Reconstruction.

This causes us to converge at iteration 203 with objective 21.228047 with Mean Squared Error: 0.005637095073441073. Here is our image showcasing the downsampled image and Reconstruction:



5. Summary and Conclusions

The goal of this Computational Report was to apply image inpainting using a convex optimization approach with proximal gradient descent to reconstruct a corrupted image. The problem was formulated with a masked least squares term and an ℓ_1 regularization term in the DCT domain to promote sparsity. We observe good decrease of our convergence of proximal descent with respect to objective value by iteration meaning that the optimization algorithm is effectively minimizing the reconstruction error and steadily progressing toward an optimal solution. Eventually, towards the end, we get a Mean Squared Error of 0.005637095073441073 showcasing effective image reconstruction indicating effective recovery of the missing pixels. These results demonstrate the effectiveness and accuracy of using proximal gradient descent with ℓ_1 regularization in the DCT domain for image inpainting.

6. Further Work

Further work can be done on such a process. One such way is by using the Mellin Transform, denoted by $\mathcal{M}$. The Mellin Transform has a scale invariant feature which allows it to effectively capture and process patterns that are invariant under scaling, potentially improving the robustness and accuracy of the reconstruction, specially for images with geometric transformations or varying resolutions. Another such way would be by using `opencv`. This library provides a wide range of tools and functions for image processing and computational vision tasks, which could enhance the reconstruction process by introducing more sophisticated preprocessing and postprocessing tasks. For example, `opencv` could be used to implement edge detection, reduce visual noise, and improve sharpness after reconstruction. Furthermore, `opencv` could be used alongside our convex optimization approach by using `cv2.inpaint()` in order to fill in missing pixel values based on local pixel information. It would be cool to see how the tasks we did would look if we did it on the colored image instead.