Question 4, Assignment 3: CS 754, Spring 2024-25

Amitesh Shekhar
IIT Bombay
22b0014@iitb.ac.in

Anupam Rawat
IIT Bombay
22b3982@iitb.ac.in

Toshan Achintya Golla
IIT Bombay
22b2234@iitb.ac.in

March 21, 2025

Declaration: The work submitted is our own, and we have adhered to the principles of academic honesty while completing and submitting this work. We have not referred to any unauthorized sources, and we have not used generative AI tools for the work submitted here.

1. In class, we studied a video compressive sensing architecture from the paper 'Video from a single exposure coded snapshot' published in ICCV 2011 (See `http://www.cs.columbia.edu/CAVE/projects/single_shot_video/`). Such a video camera acquires a 'coded snapshot' E_u in a single exposure time interval u. This coded snapshot is the superposition of the form $E_u = \sum_{t=1}^{T} C_t \cdot F_t$ where F_t is the image of the scene at instant t within the interval u and C_t is a randomly generated binary code at that time instant, which modulates F_t. Note that E_u, F_t and C_t are all 2D arrays. Also, the binary code generation as well as the final summation all occur within the hardware of the camera. Your task here is as follows:

   (a) Read the 'cars' video in the homework folder in MATLAB using the 'mmread' function which has been provided in the homework folder and convert it to grayscale. Extract the first $T = 3$ frames of the video. You may use the following code snippet:
   ```
   A = mmread('cars.avi'); T = 3; for i=1:T, X(:,:,i) = double(rgb2gray(A.frames(i).cdata));  end;
   [H,W,T] = size(X);
   ```

   (b) Generate a $H \times W \times T$ random code pattern whose elements lie in $\{0, 1\}$. Compute a coded snapshot using the formula mentioned and add zero mean Gaussian random noise of standard deviation 2 to it. Display the coded snapshot in your report.

   (c) Given the coded snapshot and assuming full knowledge of C_t for all t from 1 to T, your task is to estimate the original video sequence F_t. For this you should rewrite the aforementioned equation in the form $\boldsymbol{Ax} = \boldsymbol{b}$ where $\boldsymbol{x}$ is an unknown vector (vectorized form of the video sequence). Mention clearly what $\boldsymbol{A}$ and $\boldsymbol{b}$ are, in your report.

   (d) You should perform the reconstruction using either the ISTA algorithm or the OMP algorithm (the original paper used OMP). You can re-use your own code from a previous assignment. For computational efficiency, we will do this reconstruction patchwise. Write an equation of the form $\boldsymbol{Ax} = \boldsymbol{b}$ where $\boldsymbol{x}$ represents the ith patch from the video and having size (say) $8 \times 8 \times T$ and mention in your report what $\boldsymbol{A}$ and $\boldsymbol{b}$ stand for. For perform the reconstruction, assume that each 8×8 slice in the patch is sparse or compressible in the 2D-DCT basis.

   (e) Repeat the reconstruction for all overlapping patches and average across the overlapping pixels to yield the final reconstruction. Display the reconstruction and mention the relative mean squared error between reconstructed and original data, in your report as well as in the code.

   (f) Repeat this exercise for $T = 5, T = 7$ and mention the mention the relative mean squared error between reconstructed and original data again.

   (g) **Note: To save time, extract a portion of about 120×240 around the lowermost car in the cars video and work entirely with it. In fact, you can show all your results just on this part. Some sample results are included in the homework folder.**

   (h) Repeat the experiment with any consecutive 5 frames of the 'flame' video from the homework folder. [20 points = 12 points for correct implementation + 4 points for correct expressions for A,b; 4 points for display results correctly.]

Soln:

1 Theory

We know that the final 2-D image $I(x, y)$ is given by

$$I(x,y) = \sum_{t=1}^{T} C_t(x,y) \cdot F_t(x,y) \;\forall\; x, y \tag{1}$$

We can rewrite the expression in terms of matrix-vector product where each pixel signifies an element of the vector as below

$$\underbrace{\begin{bmatrix}\text{diag}(\text{vec}(C_1)) & \text{diag}(\text{vec}(C_2)) & \cdots & \text{diag}(\text{vec}(C_T))\end{bmatrix}}_{\boldsymbol{A}\in\mathbb{R}^{HW\times HWT}} \underbrace{\begin{bmatrix}\text{vec}(F_1)\\ \text{vec}(F_2)\\ \vdots \\ \text{vec}(F_T)\end{bmatrix}}_{\boldsymbol{x}\in\mathbb{R}^{HWT}} = \underbrace{\begin{bmatrix}\text{vec}(I)\end{bmatrix}}_{\boldsymbol{b}\in\mathbb{R}^{HW}} \tag{2}$$

Here, $\text{vec}(\boldsymbol{A})$ is conversion of an $m \times n$ matrix into a $mn-$sized vector.

Moreover, for each patch i, we can write $\boldsymbol{A} = \boldsymbol{\Phi}\boldsymbol{\Psi}$, where $\boldsymbol{\Phi}$ is the measurement matrix and $\boldsymbol{\Psi}$ is the sparsifying basis matrix (common for all patches). Now, for each time frame F_i, we can write $\boldsymbol{H} \in \mathbb{R}^{H_pW_p\times H_pW_p}$ such that $\text{vec}(F_t) = \boldsymbol{H}\,\text{vec}(\hat{F}_t)$ where $\boldsymbol{H}$ is taken as the 2-D DCT matrix.

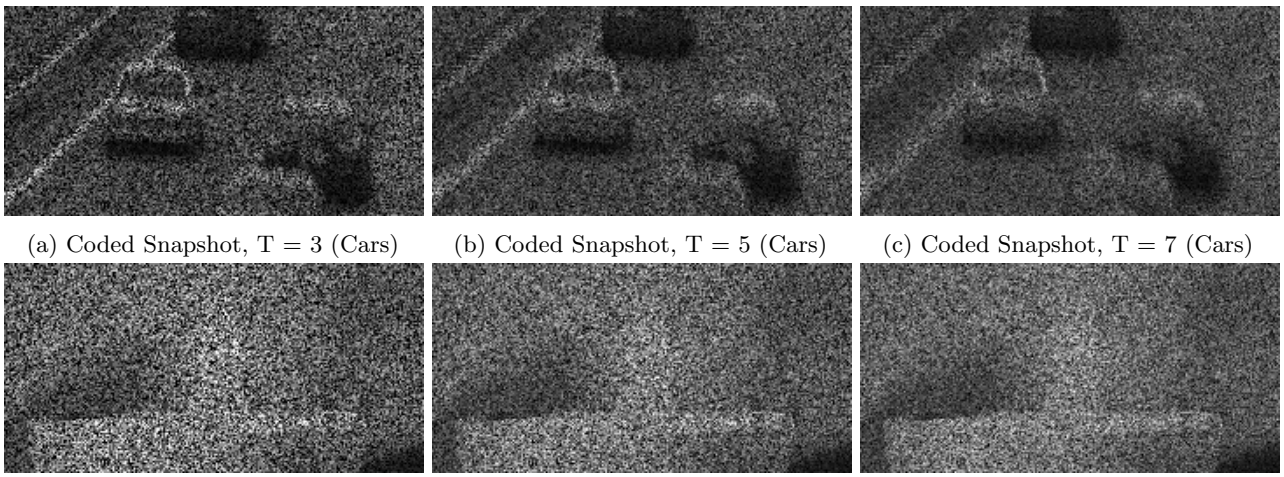
$$\underbrace{\overbrace{\begin{bmatrix}\text{diag}(\text{vec}(C_1)) & \text{diag}(\text{vec}(C_2)) & \cdots & \text{diag}(\text{vec}(C_T))\end{bmatrix}}^{\boldsymbol{\Phi}\in\mathbb{R}^{H_pW_p\times H_pW_pT}} \overbrace{\begin{bmatrix}\boldsymbol{H} & \boldsymbol{0} & \cdots & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{H} & \cdots & \boldsymbol{0}\\ \vdots & \vdots & \ddots & \vdots\\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{H}\end{bmatrix}}^{\boldsymbol{\Psi}\in\mathbb{R}^{H_pW_pT\times H_pW_pT}}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix}\text{vec}(\hat{F}_1)\\ \text{vec}(\hat{F}_2)\\ \vdots \\ \text{vec}(\hat{F}_T)\end{bmatrix}}_{\boldsymbol{\theta}\in\mathbb{R}^{H_pW_pT}} = \underbrace{\begin{bmatrix}\text{vec}(I)\end{bmatrix}}_{\boldsymbol{b}\in\mathbb{R}^{H_pW_p}} \tag{3}$$

Here, the dimensions of each patch is $(H_p, W_p) = (8, 8)$.

Above part contained the theory required for the question. ISTA algorithm was used for reconstruction purposes. The full code for running the required algorithm(s) is present in the code file.
Attaching the results obtained from running the code:

2 Results

2.1 Coded Snapshots



(a) Coded Snapshot, T = 3 (Cars) (b) Coded Snapshot, T = 5 (Cars) (c) Coded Snapshot, T = 7 (Cars)

(d) Coded Snapshot, T = 3 (Flame) (e) Coded Snapshot, T = 5 (Flame) (f) Coded Snapshot, T = 7 (Flame)

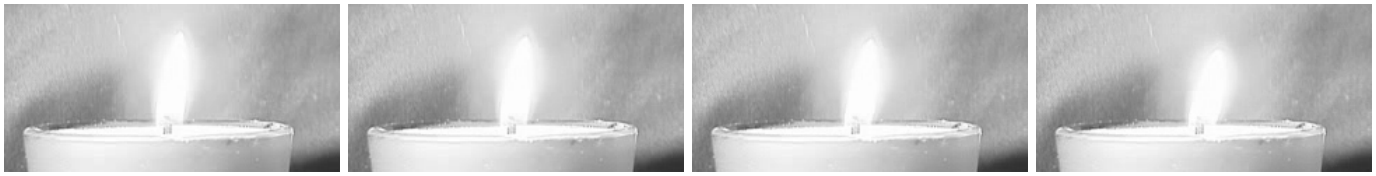
Figure 1: Coded Snapshots for Cars and Flame Videos at Different T Values

2.2 Frame Sequences

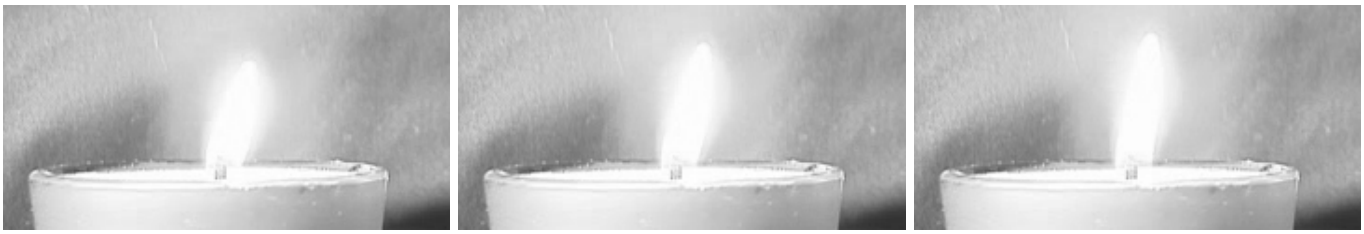
(a) Frame 1 (b) Frame 2 (c) Frame 3 (d) Frame 4

(e) Frame 5 (f) Frame 6 (g) Frame 7

Figure 2: Extracted frames from the Cars video



(a) Frame 1 (b) Frame 2 (c) Frame 3 (d) Frame 4



(e) Frame 5 (f) Frame 6 (g) Frame 7

Figure 3: Extracted frames from the Flame video

2.3 Reconstructed Frames

(a) Reconstructed, T = 3, Frame = 1 (Cars) (b) Reconstructed, T = 5, Frame = 1 (Cars) (c) Reconstructed, T = 7, Frame = 1 (Cars)

(d) Reconstructed, T = 3, Frame = 1 (Flame) (e) Reconstructed, T = 5, Frame = 1 (Flame) (f) Reconstructed, T = 7, Frame = 1 (Flame)

Figure 4: Reconstructed Frames for Different T Values

2.4 Reconstruction Losses

Table 1: Relative Mean Squared Error (RMSE) for Reconstructed Videos

Video	T = 3	T = 5	T = 7
Cars	0.12107	0.14484	0.16047
Flame	0.025309	0.028312	0.032038

3 Excerpt from the Code

Listing 1: Reading and Preprocessing the Video

```
% Read and preprocess video
video_input = mmread(char("../images/"+ image_name +".avi"));
start_frame = start_frames(image_name);
for i=1:T
    X(:,:,i) = double(rgb2gray(video_input.frames(i+start_frame-1).cdata));
end
% Cropping region of interest
H = dimensions(1);
W = dimensions(2);
X = X(start_heights(image_name):start_heights(image_name)+H-1, ...
      start_widths(image_name):start_widths(image_name)+W-1, :);
```

Listing 2: Generating Coded Snapshot

```
% Generate coded snapshot
code = randi([0 1], size(X));  % Random binary code
coded_snapshot = X .* code;
coded_snapshot = sum(coded_snapshot, 3) + generate_gaussian_noise(dimensions, 0, 4);
```

Listing 3: ISTA Algorithm for Reconstruction

```
function theta = ista(y, A, threshold, lambda)
    theta = randn([size(A,2) 1]); % Random initialization
    alpha = eigs(A'*A,1); % Compute step size
    while true
        theta_next = wthresh(theta + 1/alpha*A'*(y-A*theta), 's', lambda / (2*alpha));
        if norm(theta_next-theta) < threshold
            break;
        end
        theta = theta_next;
    end
end
```

Listing 4: Calculating RMSE

```
function RMSE = calculate_RMSE(image_original, image_reconstructed)
    RMSE = norm(image_reconstructed(:) - image_original(:)) / norm(image_original(:));
end
```