Introduction to Machine Learning: Mini-Project 1

Compressed Sensing Image Recovery

ECE 580

Spring 2022

Stacy Tantum, Ph.D.

Objective

Recover a full image from a small number of sampled pixels (compressed sensing)

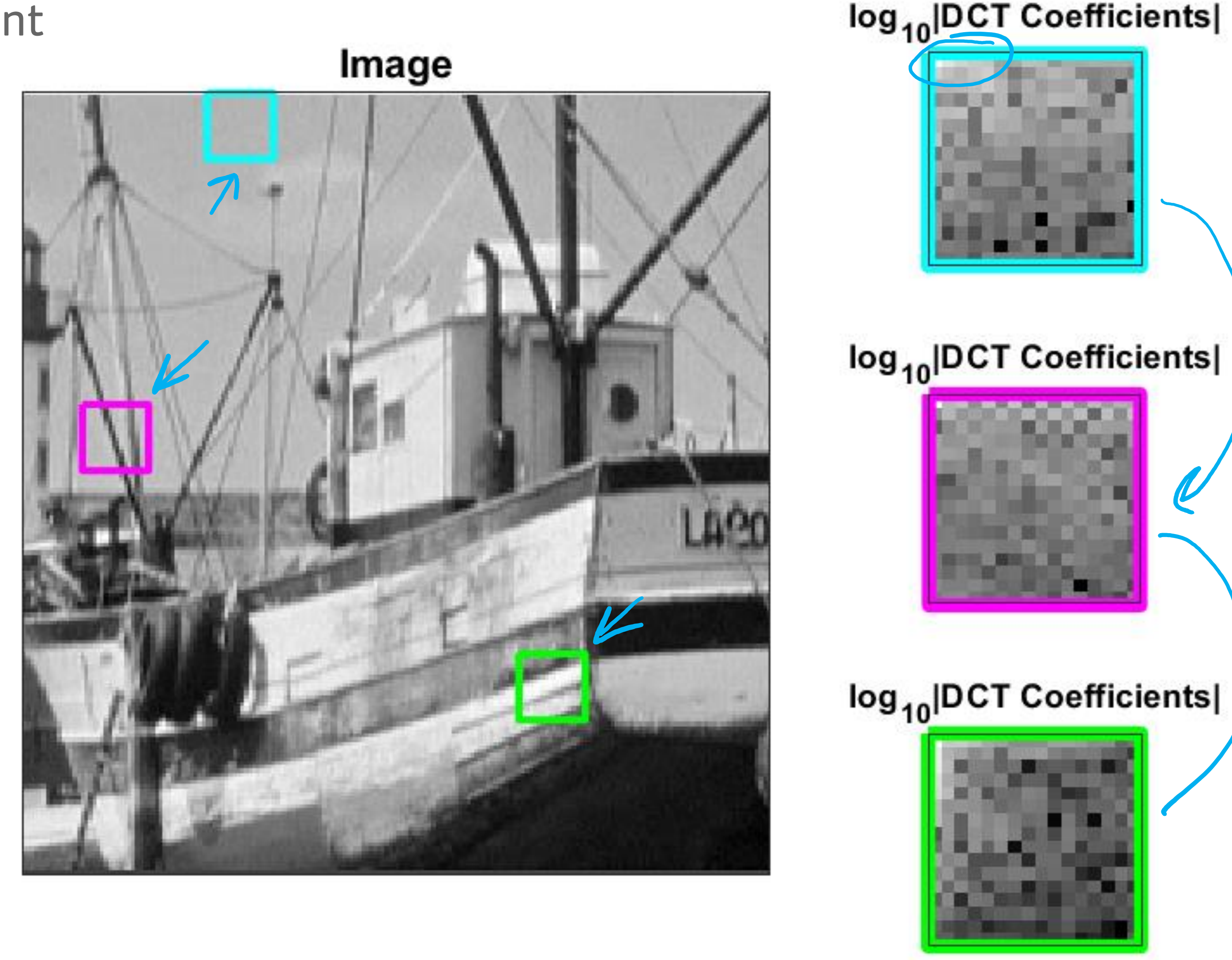
Original image

Sampled or
Corrupted image

Recovered image

2D Discrete Cosine Transform (DCT)

Provides spatial frequency content in **both** the horizontal (x) and vertical (y) directions



1-D Signal (i.e., audio signal)

A 1-dimensional signal **C** with N samples can be represented (approximated) as a weighted sum of D 1-dimensional basis functions, each of which also has N samples

- **C** is a column vector with N elements
- Each $\mathbf{T}_d$ is a column vector with N elements

$$C = \gamma_1 T_1 + \gamma_2 T_2 + \gamma_3 T_3 + \cdots + \gamma_D T_D$$

In vector-matrix notation, $\mathbf{C} = \mathbf{T}\alpha$

- Each element of **C** is a sample of the 1-D signal
- Each column of **T** is a basis function
  - the elements of each basis function (column) correspond to the samples of **C**
- Each element of γ is the weight for the corresponding basis function

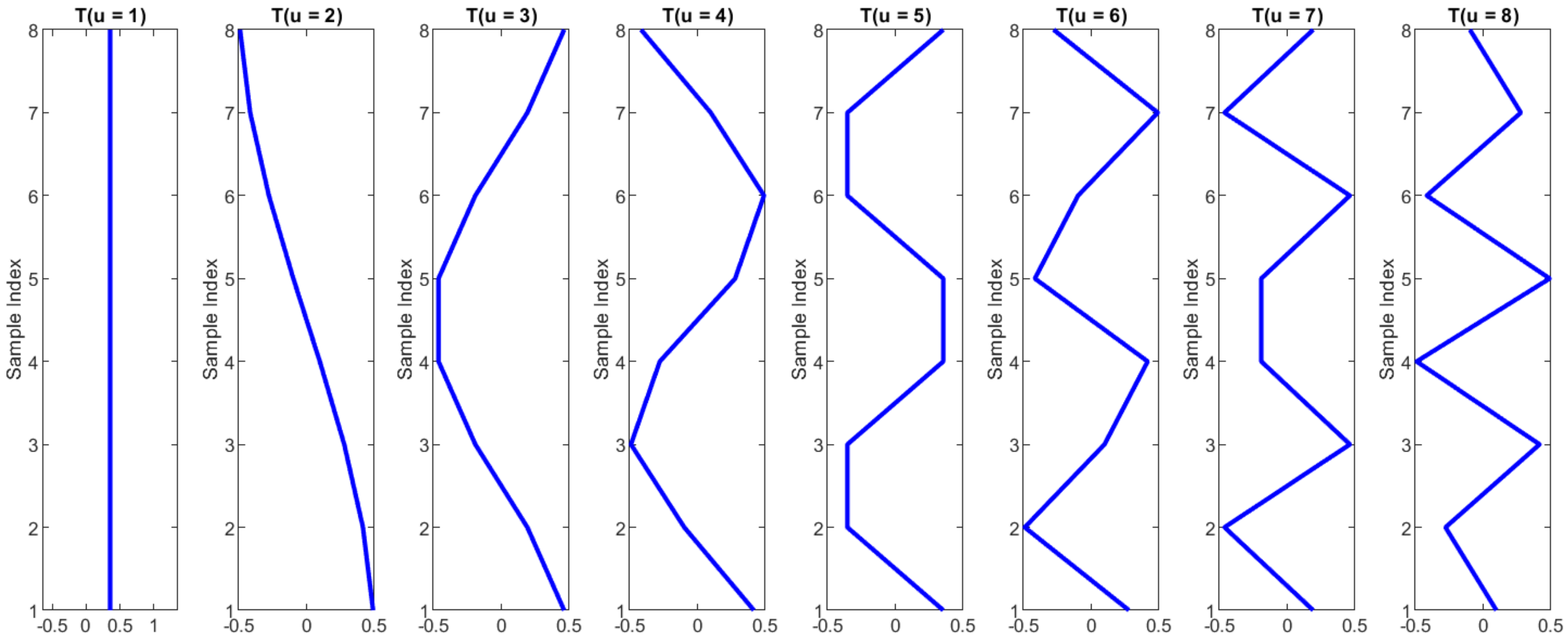
$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_N \end{bmatrix} = \begin{bmatrix} | & | & | & & | \\ T_1 & T_2 & T_3 & \cdots & T_D \\ | & | & | & & | \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \vdots \\ \gamma_D \end{bmatrix}$$

1-D Signal (i.e., audio signal)

For a 1-D DCT, the columns of **T** look something like this

- each vector corresponds to a fixed value for u (which corresponds to frequency)

u=1 u=8



2-D Signal (i.e., image)

A 2-dimensional signal **C** with NxM samples can also be represented (approximated) as a weighted sum of D 2-dimensional basis functions, each of which also has NxM samples

- **C** is a matrix with NxM elements
- Each $\mathbf{T}_d$ is a matrix with NxM elements

$$\mathcal{C} = \gamma_1 \mathbf{T}_1 + \gamma_2 \mathbf{T}_2 + \cdots + \gamma_D \mathbf{T}_D$$

The matrices **C** and $\mathbf{T}_d$ can be reshaped into column vectors with N*M elements

(this process is termed “rasterization”; the matrix is “rasterized”)

With **C** and $\mathbf{T}_d$ reshaped into column vectors, in vector-matrix notation, **C** = **T**γ

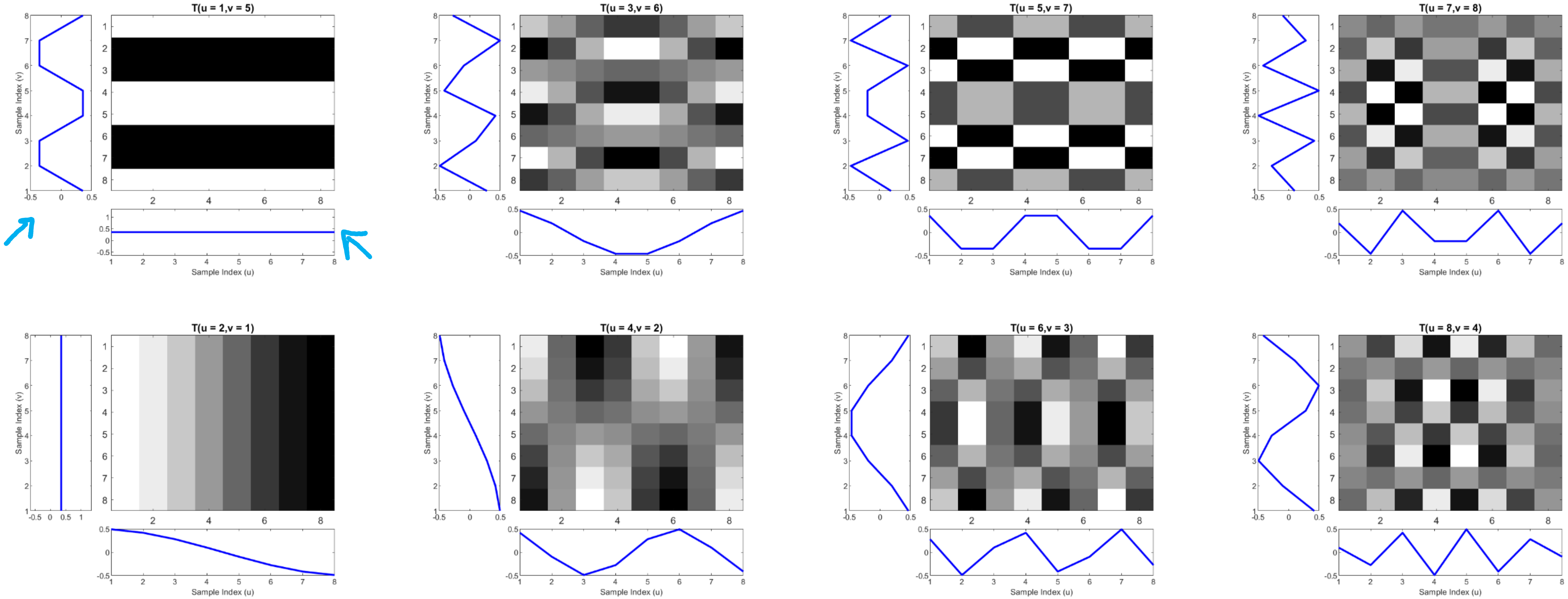
- Each element of **C** is a sample of the 2-D signal
- Each column of **T** is a basis function
  - the elements of each basis function (column) correspond to the samples of **C**
- Each element of γ is the weight for the corresponding basis function

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{N*M} \end{bmatrix} = \begin{bmatrix} | & | & | & & | \\ \mathbf{T}_1 & \mathbf{T}_2 & \mathbf{T}_3 & \cdots & \mathbf{T}_D \\ | & | & | & & | \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \vdots \\ \gamma_D \end{bmatrix}$$

2-D Signal (i.e., image)

For a 2-D DCT, the matrices $\mathbf{T}_d$ look something like this
(a subset of all possible pairwise combinations of u and v is shown)

- each matrix corresponds to fixed values for u and v
  (which correspond to horizontal spatial frequency and vertical spatial frequency)



Generating the Transformation Matrix T for a 2-D DCT

$$g(x,y)=\sum_{u=1}^{P}\sum_{v=1}^{Q}\alpha_u\cdot\beta_v\cdot\cos\frac{\pi(2x-1)(u-1)}{2\cdot P}\cdot\cos\frac{\pi(2y-1)(v-1)}{2\cdot Q}\cdot\boxed{G(u,v)}$$

This term is the DCT coefficient for the spatial frequency pair (u,v)

This term is the basis function for the spatial frequency pair (u,v).
It is a function of
1) spatial location (x,y) and
2) spatial frequency pair (u,v)
It populates the element of the matrix T_d that corresponds to the location (x,y) in the image for the frequency pair (u,v)

This equation says that the pixel value at location (x,y) is found by
1) Summing over all possible spatial frequency pairs (u,v)
2) The basis function for each spatial frequency pair (u,v) [the term outlined in orange]
3) Weighted by the coefficient for that spatial frequency pair, G(u,v) [the term outlined in teal]

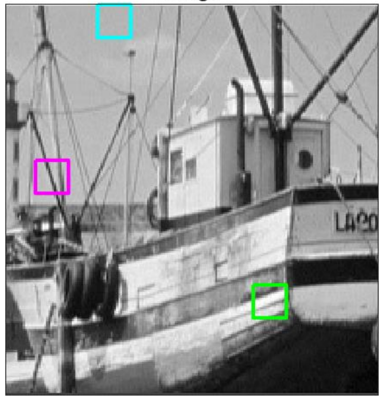
Create the matrix $\mathbf{T}_d$ for all locations in the image [(x,y) pairs] and a fixed combination of horizontal and vertical spatial frequencies [(u,v) pair]

Reshape this matrix to a column vector

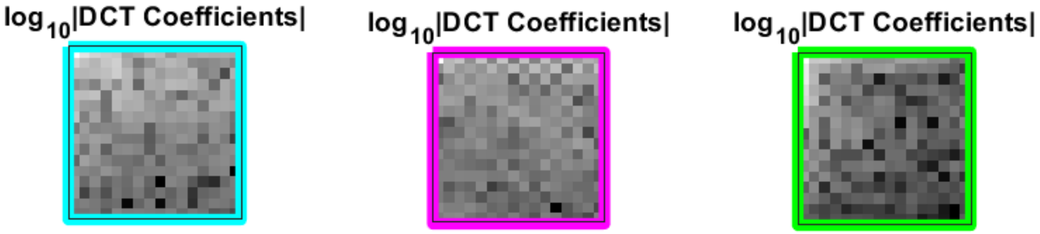
This column vector is a basis function (column vector) of **T** for the relationship **C** = **T**γ

2D Discrete Cosine Transform (DCT)

Image



DCT



$$g(x,y)=\sum_{u=1}^{P}\sum_{v=1}^{Q}\alpha_u\cdot\beta_v\cdot\cos\frac{\pi(2x-1)(u-1)}{2\cdot P}\cdot\cos\frac{\pi(2y-1)(v-1)}{2\cdot Q}\cdot G(u,v)$$

Image pixel **Transformation** **DCT coefficient**

block size = 8×8

$$x,u\in\{1,2,\cdots,P\} \quad P=8$$

$$y,v\in\{1,2,\cdots,Q\} \quad Q=8$$

$$\underbrace{\begin{bmatrix}\times\\\times\\\times\\\times\\\times\end{bmatrix}}_{\mathbf{C}}=\underbrace{\begin{bmatrix}\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\end{bmatrix}}_{\mathbf{T}}\cdot\underbrace{\begin{bmatrix}\times\\\times\\\times\\\times\\\times\end{bmatrix}}_{\boldsymbol{\gamma}}$$

$$\boxed{\alpha_u=\begin{cases}\sqrt{1/P} & (u=1)\\ \sqrt{2/P} & (2\le u\le P)\end{cases}}$$

$$\boxed{\beta_v=\begin{cases}\sqrt{1/Q} & (v=1)\\ \sqrt{2/Q} & (2\le v\le Q)\end{cases}}$$

Computing the DCT Coefficients

The transformation matrix T is known

When the complete image C is available:

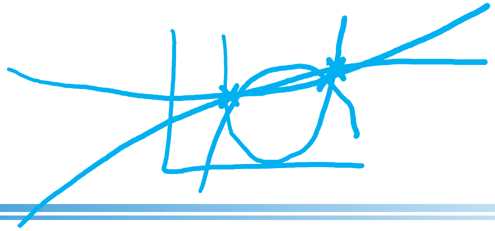
$$\gamma = T^{-1} \cdot C$$

$$\begin{array}{ccccc} P\times 1 & & P\times B & & B\times 1 \\ \begin{bmatrix}\times\\\times\\\times\\\times\\\times\end{bmatrix} & = & \begin{bmatrix}\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\end{bmatrix} & \cdot & \begin{bmatrix}\times\\\times\\\times\\\times\\\times\end{bmatrix} \\ \mathbf{C} & & \mathbf{T} & & \gamma \end{array}$$

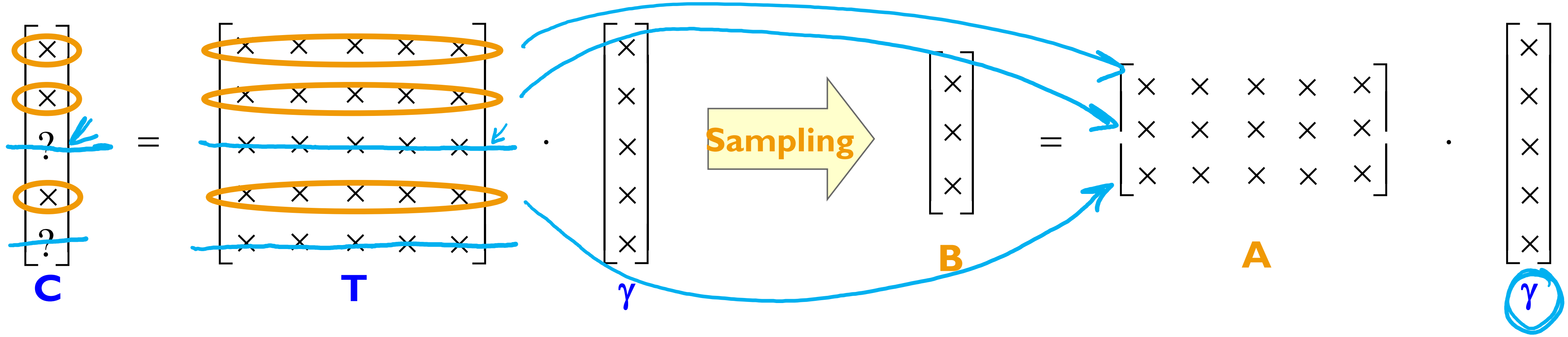
When only samples of C are available (sample vector B), can we approximate γ?

$$\begin{array}{cccccc} \begin{bmatrix}\times\\\times\\\times\end{bmatrix} & \leftarrow & \begin{bmatrix}\times\\\times\\\cancel{?}\\\times\\\cancel{?}\end{bmatrix} & = & \begin{bmatrix}\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\end{bmatrix} & \cdot \begin{bmatrix}\times\\\times\\\times\\\times\\\times\end{bmatrix} \\ \mathbf{B} & & \mathbf{C} & & \mathbf{T} & \gamma \end{array}$$

Image Recovery (Estimation)



Sampled image leads to an underdetermined linear system: $B = A \cdot \gamma$



Estimate DCT coefficients α by solving the underdetermined linear system $B = A \cdot \gamma$

Underdetermined systems generally have an infinite number of solutions

→ need to impose additional constraint(s)

Common constraint is smoothness; our constraint is ***sparsity***

Once DCT coefficients are estimated, recover the missing pixels by $\hat{C} = T \cdot \hat{\gamma}$ *(recover/estimate* ***only*** *the missing pixels)*

Why does this work? Natural images tend to be ***sparse*** *in the DCT domain*

(small number of non-zero coefficients)

Sparsity Constraint (L_1–norm Regularization – LASSO)

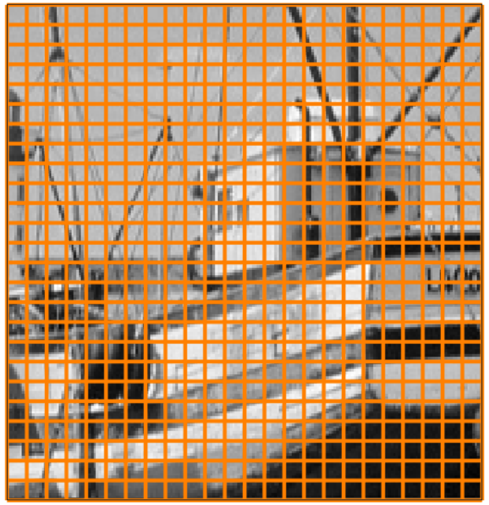
$$\underset{\mathbf{B}}{\begin{bmatrix} \times \\ \times \\ \times \end{bmatrix}} = \underset{\mathbf{A}}{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}} \cdot \underset{\boldsymbol{\gamma}}{\begin{bmatrix} \mathbf{0} \\ \times \\ \times \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}}$$

Impose sparsity on γ by L_1-norm regularization (LASSO)

$$\left.\begin{array}{l} \min\limits_{\gamma} \|A\gamma - B\|_2^2 \\ \text{such that} \quad \|\gamma\|_1 \le \lambda \end{array}\right\} \quad \begin{array}{l} \text{equivalent to LASSO} \\ \min\limits_{\gamma} \|A\gamma - B\|_2^2 + \lambda \|\gamma\|_1 \end{array}$$

Compressed Sensing in Practice

DCT coefficients of a large image are often not sparse



(8x8 blocks)

Solution: break a large image into small blocks

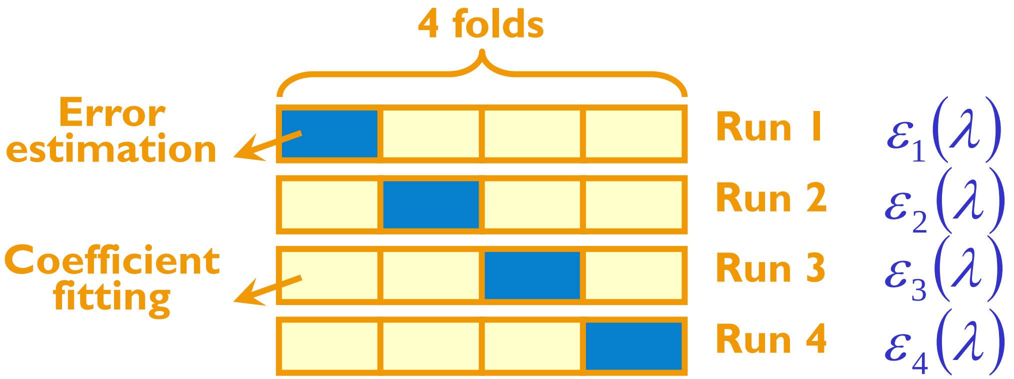
- Each small block will tend to have few non-zero DCT coefficients (different non-zero DCT coefficients for each block!)
- Try different block sizes (*K*) in your own experiments
- 8 x 8 block (64 px in block) is suggested for the small test image ("fishing boat")
- 16 x 16 block (256 px in block) is suggested for the large test image ("nature")

Choose regularization parameter λ by cross-validation

For each K x K block, sample S pixels
(these S pixels are the "sensed" pixels; pretend you don't know the other K^2-S pixels)

For each candidate λ (from the list of λ you are considering)

- Partition the block into m testing pixels and (S-m) training pixels
- Determine the DCT coefficients for the (S-m) pixels in the training set
- Estimate approximation "error" using the m pixels in the testing set
    - Use mean square error as a measurement of the "error"
    - For example, with 4-Fold cross-validation



$$Error(\lambda) = \left[\varepsilon_1(\lambda) + \varepsilon_2(\lambda) + \varepsilon_3(\lambda) + \varepsilon_4(\lambda)\right]/4$$

- Choose the λ with the lowest error (may have different λ for each block!)

Cross-Validation with Random Subsets

K-fold cross validation

- Distribute all observations into K folds such that the number of samples in every fold is equal (or as equal as possible)
- Take one fold as the test set at each iteration
- Training-and-test process is repeated K times

Cross validation with random subsets (of the S sampled pixels)

- At each iteration, randomly draw ***m*** samples (pixels) to form the test set and use the remaining **(S-m)** samples (pixels) as the training set
  - Use ***m = floor(S/6)*** in this project, where ***S*** is the total number of samples (sensed pixels in a block)
- Repeat the training-and-test process M times
  - Use ***M = 20*** in this project
- Note: With this approach to cross-validation there is a “risk” that an observation (sample, or pixel) may not be included in any of the test sets, or any of the training sets

Apply cross validation with random subsets in this project

- When the data set is small, using this method with large M tends to be more accurate than K-fold cross validation (similar benefit as L-K-fold cross-validation)
- Easy to implement if the data set cannot be divided equally into K folds

Median Filter

Median filtering replaces each pixel in an image by the median of its neighborhood

Median filtering where filter size is m×n:

- Sort all pixel values in an m×n block, centered at (x,y), to find the median
- Replace the pixel value f(x,y) by the median

123	125	126	130	140
122	124	126	127	135
118	120	150	125	134
119	115	119	123	133
111	116	110	120	130

3x3 median filtering

Neighbourhood values:

115, 119, 120, 123, 124, 125, 126, 127, 150

Median value: 124

Apply a median filter (MF) to improve the quality of recovered images

- Set filter size as 3×3
- You may use available packages/toolboxes
  - MATLAB: medfilt2 (Image processing toolbox)
  - Python: medfilt2 (SciPy multidimensional image processing – scipy.ndimage)

Recovered images (4×4 block)
(Left: w/o median filter, Right: w/ median filter)

Compare the error of the recovered image with median filtering and without median filtering

Image Recovery Summary

1) Break image into K x K blocks (an 8x8 block has 64 pixels)
2) For each block:
   - Randomly sample S pixels without replacement (number of "sensed" pixels is *S)*
     - "Without replacement" means pixels are not repeated in the set of S samples; a pixel can be "sensed" only once!
   - Determine DCT coefficients from the S samples
     - λ Is determined by cross validation using **random subsets**
       - S sampled pixels remain the same throughout cross-validation; m test pixels vary with each iteration
     - Given "optimal" λ identified through cross-validation, use all S samples to find DCT coefficients
   - Apply inverse DCT transform to recover the block
3) Combine (concatenate) all recovered blocks into a full image
4) Apply median filter to improve image quality

ADVICE:

1) This process is computationally intense. This means it may take a long time to run…
   Start Early!
2) Consider a large range of λ: 1e-6 $\rightarrow$ 1e+6, with a few values per decade [logspace]