ECE 271A - Statistical Learning 1 - Homework 5

Shubhan Mital

29th November 2025

Notation

- $\mathbf{x} \in \mathbb{R}^D$ - feature vector (DCT coefficients)
- $Y \in \{\text{FG}, \text{BG}\}$ - class label (foreground/background)
- C - number of mixture components
- α_c - mixture weight for component c (with $\sum_{c=1}^{C} \alpha_c = 1$)
- $\boldsymbol{\mu}_c \in \mathbb{R}^D$ - mean of component c
- $\boldsymbol{\Sigma}_c$ - covariance matrix of component c (diagonal)
- N - number of training samples
- P_e - probability of error (blockwise classification error)

1 Problem Setup: GMM-Based Bayesian Classification

We model each class-conditional density as a Gaussian mixture model (GMM):

$$p(\mathbf{x} \mid Y = i) = \sum_{c=1}^{C} \alpha_c^{(i)} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c^{(i)}, \boldsymbol{\Sigma}_c^{(i)}) \tag{1}$$

where:

- $\alpha_c^{(i)} \geq 0$ are mixture weights with $\sum_{c=1}^{C} \alpha_c^{(i)} = 1$
- $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ denotes the multivariate Gaussian density
- Each $\boldsymbol{\Sigma}_c^{(i)}$ is a **diagonal** covariance matrix

1.1 Diagonal Covariance GMM

For computational efficiency and numerical stability, we restrict each component to have diagonal covariance:

$$\boldsymbol{\Sigma}_c = \text{diag}(\sigma_{c,1}^2, \sigma_{c,2}^2, \ldots, \sigma_{c,D}^2) \tag{2}$$

The Gaussian PDF for component c becomes:

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) = \frac{1}{(2\pi)^{D/2} \prod_{d=1}^{D} \sigma_{c,d}} \exp\left(-\frac{1}{2} \sum_{d=1}^{D} \frac{(x_d - \mu_{c,d})^2}{\sigma_{c,d}^2} \right) \tag{3}$$

1.2 Expectation-Maximization (EM) Algorithm

We estimate GMM parameters $\Theta = \{\boldsymbol{\alpha}, \{\boldsymbol{\mu}_c\}, \{\boldsymbol{\Sigma}_c\}\}$ via EM:

1.2.1 E-Step: Compute Responsibilities

For each sample $\mathbf{x}_n$ and component c:

$$\gamma_{nc} = \frac{\alpha_c \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)}{\sum_{j=1}^{C} \alpha_j \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{4}$$

The responsibility γ_{nc} represents the posterior probability that sample n belongs to component c.

1.2.2 M-Step: Update Parameters

Define the effective number of points assigned to component c:

$$N_c = \sum_{n=1}^{N} \gamma_{nc} \tag{5}$$

Update mixture weights:

$$\alpha_c = \frac{N_c}{N} \tag{6}$$

Update means:

$$\boldsymbol{\mu}_c = \frac{1}{N_c} \sum_{n=1}^{N} \gamma_{nc} \mathbf{x}_n \tag{7}$$

Update diagonal covariances:

$$\sigma_{c,d}^2 = \frac{1}{N_c} \sum_{n=1}^{N} \gamma_{nc} (x_{n,d} - \mu_{c,d})^2 + \epsilon \tag{8}$$

where $\epsilon = 10^{-5}$ is a regularization constant to prevent singular covariances.

1.3 EM Convergence Criterion

We monitor the log-likelihood:

$$\mathcal{L}(\Theta) = \sum_{n=1}^{N} \log \left(\sum_{c=1}^{C} \alpha_c \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \right) \tag{9}$$

EM iterations continue until:

$$\frac{|\mathcal{L}^{(t)} - \mathcal{L}^{(t-1)}|}{\max(1, |\mathcal{L}^{(t)}|)} < \tau \tag{10}$$

where $\tau = 10^{-6}$ is the convergence tolerance.

1.4 Initialization via K-Means

To initialize EM, we use K-means clustering:

1. Run K-means with C clusters on training data
2. Initialize $\boldsymbol{\mu}_c$ as K-means cluster centers
3. Initialize $\alpha_c = N_c/N$ based on cluster assignments
4. Initialize $\sigma_{c,d}^2$ as sample variance within each cluster

1.5 Bayesian Decision Rule

Given trained GMMs for foreground (FG) and background (BG), we classify using:

$$\hat{Y}(\mathbf{x}) = \begin{cases} \text{FG} & \text{if } p(\mathbf{x} \mid \text{FG})P(\text{FG}) > p(\mathbf{x} \mid \text{BG})P(\text{BG}) \\ \text{BG} & \text{otherwise} \end{cases} \tag{11}$$

For numerical stability, we use log-posteriors:

$$\log p(\mathbf{x} \mid Y = i) = \log \left(\sum_{c=1}^{C} \alpha_c^{(i)} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c^{(i)}, \boldsymbol{\Sigma}_c^{(i)}) \right) \tag{12}$$

$$= \text{logsumexp}_c \left(\log \alpha_c^{(i)} + \log \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_c^{(i)}, \boldsymbol{\Sigma}_c^{(i)}) \right) \tag{13}$$

1.6 Error Computation

For blockwise classification on the cheetah image:

$$P_e = \frac{\text{Number of misclassified } 8\times8 \text{ blocks}}{\text{Total number of blocks}} \tag{14}$$

Each block is classified based on its DCT feature vector, and compared against the ground-truth mask.

2 Part (a): Effect of Random Initialization

2.1 Experimental Setup

- Fixed number of mixture components: $C = 8$
- Train 5 independent GMMs for foreground class (FG) using different random seeds
- Train 5 independent GMMs for background class (BG) using different random seeds
- Evaluate all $5 \times 5 = 25$ possible classifier combinations
- Test across DCT dimensions: $D \in \{1, 2, 4, 8, 13, 16, 24, 28, 32, 40, 48, 56, 58, 64\}$

2.2 Methodology

For each combination (BG model i, FG model j):

1. Extract top D DCT coefficients from each 8×8 block
2. Compute log-posteriors using the trained GMMs:

$$\log p(Y = \text{BG} \mid \mathbf{x}) = \log p(\mathbf{x} \mid \text{BG}_i) + \log P(\text{BG}) \tag{15}$$

$$\log p(Y = \text{FG} \mid \mathbf{x}) = \log p(\mathbf{x} \mid \text{FG}_j) + \log P(\text{FG}) \tag{16}$$

3. Classify: $\hat{Y} = \arg\max_Y \log p(Y \mid \mathbf{x})$
4. Compute blockwise error rate P_e

2.3 Results

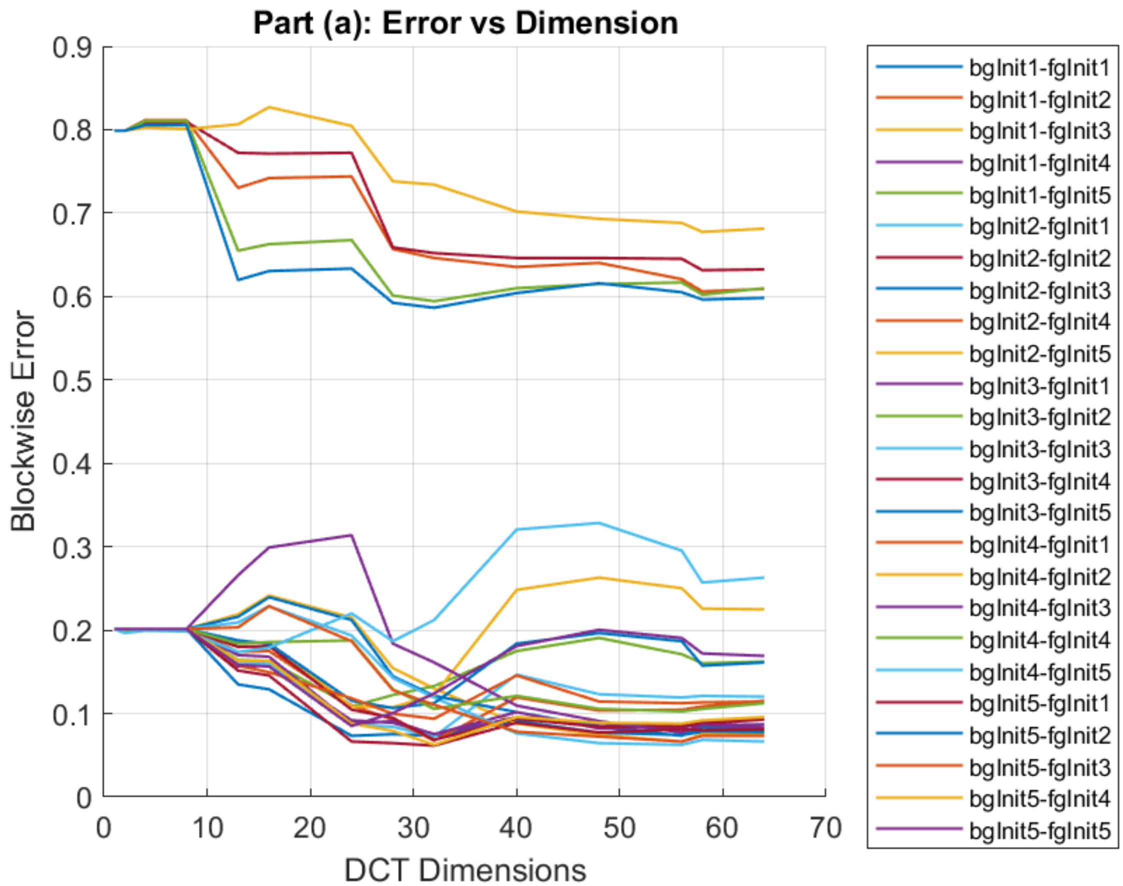

Figure 1: Part (a): Blockwise error vs. DCT dimensions for 25 classifier combinations (5 BG × 5 FG random initializations, C=8). Each curve represents a different pair of independently trained GMMs.

2.4 Analysis and Observations

2.4.1 Initialization Sensitivity

The results reveal **significant dependence on random initialization**:

1. **Wide performance variance:** Error rates range from ≈ 0.08 (best) to ≈ 0.80 (worst) at high dimensions

2. **Local optima problem:** Several initialization pairs converge to poor solutions, maintaining high error (> 0.60) even with 64 DCT coefficients

3. **Clustering of solutions:** Successful initializations form a cluster of curves with similar trajectories, suggesting convergence to similar (or the same) local maximum of the likelihood

4. **Best performers:** Approximately 40% of combinations achieve near-optimal performance ($P_e < 0.12$ at $D = 64$)

2.4.2 Dimension-Dependent Behavior

Low dimensions ($D \leq 8$):

- All models perform poorly ($P_e \approx 0.80$)

- Insufficient discriminative information in first few DCT coefficients
- DC and low-frequency AC coefficients alone cannot separate cheetah from grass

Medium dimensions ($8 < D \leq 24$):

- Rapid error reduction for good initializations
- Performance divergence becomes apparent
- Good models: P_e drops from 0.25 to ≈ 0.10
- Poor models: remain stuck at $P_e > 0.60$

High dimensions ($D > 24$):

- Well-initialized models converge to minimum error ($P_e \approx 0.08$)
- Poorly initialized models show minimal improvement
- Some poor models even exhibit slight error increase (Believe that's due to overfitting to degenerate components)

2.4.3 Why Does Initialization Matter?

The EM algorithm for GMMs is guaranteed to converge to a **local maximum** of the likelihood, not necessarily the global maximum. Poor initialization can lead to:

1. **Degenerate components:** One or more Gaussians collapse to single points or very small regions
2. **Redundant components:** Multiple Gaussians model the same region of feature space
3. **Poor coverage:** Some regions of the data distribution are not covered by any component
4. **Unbalanced mixtures:** Mixture weights become highly skewed (e.g., $\alpha_1 \approx 1$, others ≈ 0)

2.4.4 Numerical Summary

Best performing combination: **Minimum error** ≈ 0.08 achieved by several initialization pairs at $D \geq 24$ coefficients. Worst performing combination: **Maximum error** ≈ 0.80 for poor initializations across all dimensions.

3 Part (b): Effect of Number of Mixture Components

3.1 Experimental Setup

- Vary number of components: $C \in \{1, 2, 4, 8, 16, 32\}$
- Single random initialization for each C (fixed seed for reproducibility)
- Evaluate across same DCT dimensions as Part (a)

3.2 Results

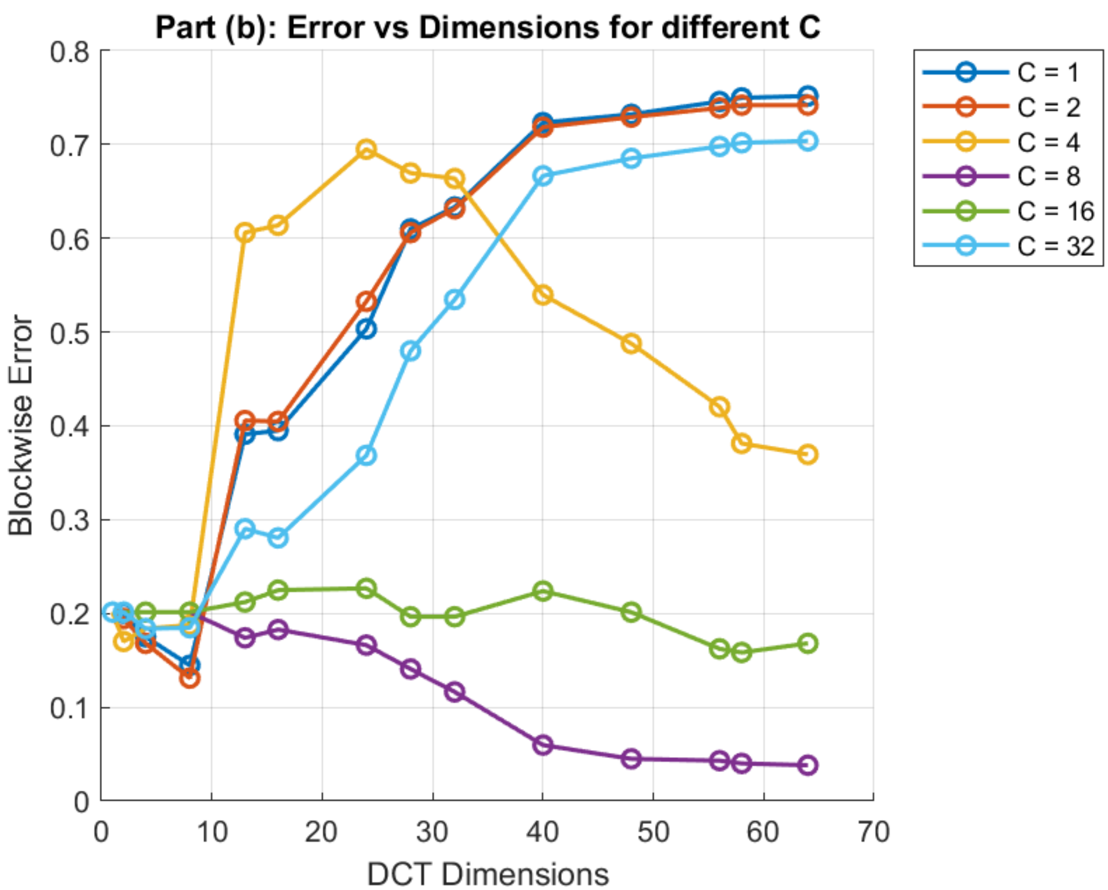

Figure 2: Part (b): Blockwise error vs. DCT dimensions for different numbers of mixture components C. Each curve shows performance for a specific model complexity.

3.3 Detailed Analysis by Model Complexity

3.3.1 $C = 1$ (Single Gaussian per Class)

Behavior:

- Low-dimension performance: $P_e \approx 0.20$ at $D = 1$
- **Anomalous behavior:** Error increases dramatically to ≈ 0.70 at high dimensions

Explanation:

1. A single Gaussian assumes **unimodal distribution**

2. Both cheetah and grass textures are inherently **multimodal** in DCT space
3. At low dimensions, the single Gaussian provides a crude approximation
4. At high dimensions:
   - The model tries to fit one Gaussian to multimodal data
   - Results in **high variance** in many dimensions
   - Decision boundaries become worse as dimensions increase

Note: This increasing error pattern suggests that the single Gaussian is fundamentally inadequate for this data.

3.3.2 $C = 2$ (Two Gaussians per Class)

Behavior:

- Similar pattern to $C = 1$
- Starts at $P_e \approx 0.18$ (slightly better than $C = 1$)
- Error increases to ≈ 0.75 at high dimensions

Explanation:

1. Two components provide minimal additional flexibility
2. Still insufficient to capture complex multimodal structure
3. Suggests that texture distributions require more components ($C \geq 4$)

Note: This increasing error pattern suggests that the two Gaussians is fundamentally inadequate for this data.

3.3.3 $C = 4$ (Four Gaussians per Class)

Behavior:

- Initial error: $P_e \approx 0.18$
- Shows non-monotonic behavior with dimension
- Peak error ≈ 0.70 around $D = 16 - 20$
- Recovers to $P_e \approx 0.37$ at $D = 64$

Explanation:

1. Four components begin to capture multimodality
2. The spike at intermediate dimensions suggests:
   - Model is becoming complex enough to overfit noise
   - Component assignments may be unstable in medium-dimensional space
   - Transition region where simple model breaks but complex model not yet effective
3. Better than $C = 1, 2$ at high dimensions but still suboptimal

3.3.4 $C = 8$ (Eight Gaussians per Class) - Best Performance

Behavior:

- Consistent **monotonic decrease** in error with dimension
- $D = 1$: $P_e \approx 0.18$
- $D = 8$: $P_e \approx 0.17$
- $D = 24$: $P_e \approx 0.06$
- $D = 64$: $P_e \approx 0.04$ **(best overall)**

Explanation:

1. **Optimal model complexity:** Eight components provide sufficient flexibility to model multimodal texture distributions without severe overfitting
2. **Captures texture variability:**
   - Cheetah fur exhibits different patterns (spots, shadows, highlights)
   - Grass background has variations (blades, soil, shadows)
   - Eight components can represent these sub-classes
3. **Balanced bias-variance tradeoff:**
   - Not too simple (avoids high bias of $C = 1, 2$)
   - Not too complex (avoids overfitting of $C = 32$)
4. **Stable convergence:** With good initialization (K-means), 8 components reliably converge to meaningful clusters

3.3.5 $C = 16$ (Sixteen Gaussians per Class)

Behavior:

- Stable performance: $P_e \approx 0.15 - 0.20$ across most dimensions
- Slight improvement at high dimensions
- Final error ≈ 0.15 at $D = 64$

Explanation:

1. More components than needed for this data
2. Performance degrades compared to $C = 8$ because:
   - **Overfitting:** Too many parameters for available training data
   - **Component splitting:** Meaningful clusters get subdivided unnecessarily
   - **Reduced generalization:** Model memorizes training blocks rather than learning general texture patterns
3. Still reasonably good, but not optimal

3.3.6 $C = 32$ (Thirty-Two Gaussians per Class)

Behavior:

- Poorest performance among $C \geq 4$
- Error ranges from 0.17 to 0.30
- Erratic behavior across dimensions

Explanation:

1. **Severe overfitting:** 32 components $\times$ 2 classes = 64 Gaussians
2. With diagonal covariance in $D = 64$ dimensions:
   - Total parameters per class: $32 \times (64 \text{ mean} + 64 \text{ variance} + 1 \text{ weight}) = 4,128$
   - Training samples (from typical HW): ≈ 900 BG + ≈ 225 FG
   - Parameters approach sample size!
3. **Degenerate components:** Some Gaussians collapse to individual training points
4. **Poor density estimation:** Over-fragmented model fails to capture true underlying distribution
5. **Numerical instability:** Near-singular covariances even with regularization

3.4 Bias-Variance Tradeoff

The results clearly illustrate the bias-variance tradeoff:

Model Complexity	Bias	Variance
$C = 1$ (underfitting)	**High**	Low
$C = 8$ (optimal)	Low	Low
$C = 32$ (overfitting)	Low	**High**

Optimal point: $C = 8$ achieves the best balance, minimizing total error.

3.5 Numerical Summary: Best Error for Each C

C	Best P_e	At Dimension
1	0.20	$D = 1 - 4$
2	0.18	$D = 1$
4	0.37	$D = 64$
8	**0.04**	$D = 64$
16	0.15	$D = 64$
32	0.17	$D = 8$

3.6 Key Takeaway

Model selection is crucial: More components is not always better. The optimal C depends on:

- Complexity of the true data distribution
- Amount of training data available
- Dimensionality of feature space

For this cheetah classification task, $C = 8$ **with** $D = 64$ **DCT coefficients achieves optimal performance** ($P_e \approx 0.04$).

4 Comparison: GMM vs. Single Gaussian (HW3)

Compared to HW3 (single Gaussian per class):

- **HW3 ML error:** $P_e \approx 0.08 - 0.12$ (typical for good datasets)
- **HW5 GMM error ($C = 8$):** $P_e \approx 0.04$
- **Improvement:** $\approx 50\%$ error reduction!

Why does GMM perform better?

1. Captures multimodal structure of texture distributions
2. Better models complex within-class variability
3. More flexible decision boundaries

5 Conclusions

1. **Initialization matters:** EM is sensitive to initialization; multiple random starts recommended
2. **Optimal complexity:** $C = 8$ provides best bias-variance tradeoff for this problem
3. **Dimensionality:** Using full 64 DCT coefficients gives best performance (when model is appropriate)
4. **GMM superiority:** Mixture models substantially outperform single Gaussian for texture classification
5. **Overfitting danger:** Too many components ($C \geq 16$) leads to degraded generalization

```
--- PART (a): 5 random inits per class, C=8 each ---
Summary for Part A(numeric highlights):Best pair (idx 21) block error = 0.0616
Figure for Part A saved to folder

--- PART (b): Vary number of mixture components ---
Summary for Part B(numeric highlights):
- For C= 1 best error = 0.1447 at D=8
- For C= 2 best error = 0.1310 at D=8
- For C= 4 best error = 0.1701 at D=2
- For C= 8 best error = 0.0381 at D=64
- For C=16 best error = 0.1584 at D=58
- For C=32 best error = 0.1838 at D=4
Figure for Part B saved to folder
>>
```

Figure 3: Final numerical results from my code

Appendix: Matlab Code

```matlab
%% ========================================================================
%  ECE 271A - HW5: GMM-based Bayesian Classifier
%  Author: Shubhan Mital
%  Description:
%    Implements GMM-based Bayesian classification for cheetah image segmentation.
%    Loads image, mask, zig-zag pattern, and DCT training samples.
%    Trains diagonal-covariance GMMs for FG/BG using EM.
%    Evaluates classification error across multiple dimensions and mixture sizes.
% ========================================================================

clear; close all; clc;
warning('off', 'all');

% User file paths
imgPath       = "C:\Users\shubh\Desktop\Hard disk\College(PG)\Academics at UCSD\Y1Q1\ECE 271A - Statistical Learning 1\Homework\homework5\cheetah.bmp";
maskPath      = "C:\Users\shubh\Desktop\Hard disk\College(PG)\Academics at UCSD\Y1Q1\ECE 271A - Statistical Learning 1\Homework\homework5\cheetah_mask.bmp";
trainMatPath  = "C:\Users\shubh\Desktop\Hard disk\College(PG)\Academics at UCSD\Y1Q1\ECE 271A - Statistical Learning 1\Homework\homework5\TrainingSamplesDCT_subsets_8.mat";
zigzagPath    = "C:\Users\shubh\Desktop\Hard disk\College(PG)\Academics at UCSD\Y1Q1\ECE 271A - Statistical Learning 1\Homework\homework5\Zig-Zag Pattern.txt";
outputFolder  = fileparts(imgPath);

% Parameters
dims_to_try = [1 2 4 8 13 16 24 28 32 40 48 56 58 64];      %the number of DCT coefficients to consider for feature vectors.
maxEMiter   = 1000;                                          %max number of iterations for the EM algorithm.
emTol       = 1e-6;                                          %convergence tolerance for EM.
epsilon_cov = 1e-5;                                          %small value to avoid singular covariance matrices.
rng('default');                                              %sets random seed for reproducibility.

% Load image and mask
I = imread(imgPath);                                         %reads the image.
if size(I,3) > 1, I = rgb2gray(I); end                       %Converts color image to grayscale if needed.
I = double(I);
mask_im = imread(maskPath);                                  %reads the mask.
if size(mask_im,3) > 1, mask_im = rgb2gray(mask_im); end     %Converts mask to binary (pixels > 128 → 1, else 0)
mask_binary = mask_im > 128;                                 % cheetah=1, background=0
[H, W] = size(I);                                            %Stores image height and width in H and W.

% Load zigzag pattern
```

```
zz = readmatrix(zigzagPath);
zz = zz(:);
if numel(zz) ~= 64, error('badzz'); end
if min(zz) == 0, zz = zz + 1; end
% Load training samples
if ~exist(trainMatPath,'file'), error('TrainingSamples file not found.'); end
S = load(trainMatPath);
vars = fieldnames(S);
% Detect BG and FG variables
bgVar = ''; fgVar = '';
for i=1:numel(vars)
    v = vars{i}; lname = lower(v);
    if contains(lname,'bg') || contains(lname,'background') || contains(lname,'non') &&↙
isempty(bgVar)
        bgVar = v;
    end
    if contains(lname,'fg') || contains(lname,'cheetah') || contains↙
(lname,'foreground') && isempty(fgVar)
        fgVar = v;
    end
end

% Convert and Organize Training Data
bg_samples = double(S.(bgVar));
fg_samples = double(S.(fgVar));
if size(bg_samples,1) ~= 64 && size(bg_samples,2) == 64, bg_samples = bg_samples'; end
if size(fg_samples,1) ~= 64 && size(fg_samples,2) == 64, fg_samples = fg_samples'; end

% Class priors
prior_bg = size(bg_samples,2) / (size(bg_samples,2) + size(fg_samples,2));
prior_fg = 1 - prior_bg;

% Extract DCT features for all 8x8 blocks
blocks_vert = H/8; blocks_horz = W/8;                          % Number of vertical↙
and horizontal 8x8 blocks
if mod(H,8)~=0 || mod(W,8)~=0                                  % Check if image↙
dimensions are divisible by 8
    H2 = floor(H/8)*8; W2 = floor(W/8)*8;                      % Round down dimensions↙
to nearest multiple of 8
    I = I(1:H2,1:W2); mask_binary = mask_binary(1:H2,1:W2);    % Crop image and mask↙
to new size
    [H,W] = size(I); blocks_vert = H/8; blocks_horz = W/8;     % Update block counts↙
after cropping
end
numBlocks = blocks_vert * blocks_horz;                         % Total number of 8x8↙
blocks in the image
allBlocks = zeros(64, numBlocks);                              % Pre-allocate matrix↙
to store 64 DCT coefficients per block
idx = 1;                                                       % Initialize block↙
index counter
```

```
for by=1:8:H                                              % Loop over vertical↙
blocks (step size 8)
    for bx=1:8:W                                          % Loop over horizontal↙
blocks (step size 8)
        block = I(by:by+7,bx:bx+7);                       % Extract current 8x8↙
block
        B = dct2(block); Bvec = B(:);                     % Compute 2D DCT and↙
vectorize it into 64x1
        allBlocks(:,idx) = Bvec(zz);                      % Reorder coefficients↙
using zig-zag pattern and store
        idx = idx + 1;                                    % Increment block index
    end
end
% Ground-truth labels per block
gtBlocks = false(1,numBlocks);                            % Pre-allocate logical↙
array for block labels
idx=1;                                                    % Reset block index↙
counter
for by=1:8:H                                              % Loop over vertical↙
blocks (step size 8)
    for bx=1:8:W                                          % Loop over horizontal↙
blocks (step size 8)
        mblock = mask_binary(by:by+7,bx:bx+7);            % Extract corresponding↙
8x8 mask block
        gtBlocks(idx) = mean(mblock(:)) >= 0.5;           % Assign 1 if majority↙
of mask is foreground, else 0
        idx = idx + 1;                                    % Increment block index
    end
end

% ----------------- PART (a): 5 random inits per class, C=8 -----------------
fprintf('--- PART (a): 5 random inits per class, C=8 each ---\n');
C_a = 8; num_inits = 5;↙
% Set number of GMM components (C=8) and 5 random initializations
bg_models = cell(num_inits,1);↙
% Pre-allocate cells to store BG GMM models
fg_models = cell(num_inits,1);↙
% Pre-allocate cells to store FG GMM models
% Train BG
for s=1:num_inits
    rng(s+1000);↙
% Set random seed for reproducibility
    [alpha, mu, sigma_diag] = em_gmm_diag(bg_samples, C_a, maxEMiter, emTol,↙
epsilon_cov);            % Train diagonal-covariance GMM on BG samples
    bg_models{s} = struct('alpha',alpha,'mu',mu,'sigma_diag',sigma_diag);↙
% Store trained BG model
end
% Train FG
for s=1:num_inits
    rng(s+2000);↙
```

```
% Set different random seed for FG initialization
    [alpha, mu, sigma_diag] = em_gmm_diag(fg_samples, C_a, maxEMiter, emTol,↙
epsilon_cov);            % Train diagonal-covariance GMM on FG samples
    fg_models{s} = struct('alpha',alpha,'mu',mu,'sigma_diag',sigma_diag);↙
% Store trained FG model
end
% Compute errors for all 25 pairs of BG-FG models
errors_a = zeros(num_inits*num_inits, numel(dims_to_try));↙
% Pre-allocate error matrix
pair_idx = 1; figure('Name','Part (a)'); hold on;↙
% Initialize figure for plotting
colors = lines(num_inits*num_inits); legendEntries = cell(num_inits*num_inits,1);↙
% Colors and legend entries
for bi=1:num_inits↙
% Loop over BG initializations
    for fi=1:num_inits↙
% Loop over FG initializations
        bgm = bg_models{bi}; fgm = fg_models{fi};↙
% Select current BG and FG models
        errs = zeros(1,numel(dims_to_try));↙
% Pre-allocate error vector for this pair
        for di=1:numel(dims_to_try)↙
% Loop over different DCT feature dimensions
            Ddim = dims_to_try(di); Xsub = allBlocks(1:Ddim,:);↙
% Select top Ddim DCT coefficients
            logp_bg = log_gmm_diag_pdf(Xsub, bgm.alpha, bgm.mu(1:Ddim,:), bgm.↙
sigma_diag(1:Ddim,:)); % Log-likelihood for BG
            logp_fg = log_gmm_diag_pdf(Xsub, fgm.alpha, fgm.mu(1:Ddim,:), fgm.↙
sigma_diag(1:Ddim,:)); % Log-likelihood for FG
            logpost_bg = logp_bg + log(prior_bg + realmin);↙
% Compute log posterior for BG
            logpost_fg = logp_fg + log(prior_fg + realmin);↙
% Compute log posterior for FG
            pred_fg = logpost_fg > logpost_bg;↙
% Predict FG if posterior higher than BG
            errs(di) = mean(pred_fg ~= gtBlocks);↙
% Compute blockwise error for this dimension
        end
        errors_a(pair_idx,:) = errs;↙
% Store errors for this pair of initializations
        plot(dims_to_try, errs, '-', 'LineWidth',1,'Color',colors(pair_idx,:));↙
% Plot error curve
        legendEntries{pair_idx} = sprintf('bgInit%d-fgInit%d', bi, fi);↙
% Create legend entry
        pair_idx = pair_idx + 1;↙
% Increment pair index
    end
end
xlabel('DCT Dimensions'); ylabel('Blockwise Error'); title('Part (a): Error vs↙
Dimension');           % Labels and title
```

```
legend(legendEntries,'Location','bestoutside','FontSize',8); grid on; hold off;↙
% Add legend and grid
saveas(gcf, fullfile(outputFolder, 'part_a_25_curves.png'));↙
% Save figure to output folder
% Output Statements
fprintf('Summary for Part A(numeric highlights):');
[minErrA, ~] = min(errors_a, [], 2); [minVal, bestPair] = min(minErrA);↙
% Find best error from Part (a)
fprintf('Best pair (idx %d) block error = %.4f', bestPair, minVal);
fprintf('\nFigure for Part A saved to folder\n');↙
% Completion message

% ----------------- PART (b): Vary number of mixture components C -----------------
fprintf('\n--- PART (b): Vary number of mixture components ---\n');
C_list = [1 2 4 8 16 32];↙
% List of different numbers of GMM components to try
errors_b = zeros(numel(C_list), numel(dims_to_try));↙
% Pre-allocate error matrix (rows=C, cols=DCT dims)
models_bg_b = cell(numel(C_list),1);↙
% Pre-allocate cells to store BG GMM models
models_fg_b = cell(numel(C_list),1);↙
% Pre-allocate cells to store FG GMM models
for ci=1:numel(C_list)↙
% Loop over each number of mixture components
    Cc = C_list(ci);↙
% Current number of components
rng(5000+ci);↙
% Set random seed for BG model reproducibility
    [alpha_bg, mu_bg, sigma_bg] = em_gmm_diag(bg_samples, Cc, maxEMiter, emTol,↙
epsilon_cov);       % Train BG GMM
    models_bg_b{ci} = struct('alpha',alpha_bg,'mu',mu_bg,'sigma_diag',sigma_bg);↙
% Store BG model
    rng(8000+ci);↙
% Set random seed for FG model reproducibility
    [alpha_fg, mu_fg, sigma_fg] = em_gmm_diag(fg_samples, Cc, maxEMiter, emTol,↙
epsilon_cov);       % Train FG GMM
    models_fg_b{ci} = struct('alpha',alpha_fg,'mu',mu_fg,'sigma_diag',sigma_fg);↙
% Store FG model
    errs = zeros(1,numel(dims_to_try));↙
% Pre-allocate error vector for current C
    for di=1:numel(dims_to_try)↙
% Loop over DCT feature dimensions
        Ddim = dims_to_try(di); Xsub = allBlocks(1:Ddim,:);↙
% Select top Ddim DCT coefficients
    bgm = models_bg_b{ci}; fgm = models_fg_b{ci};↙
% Get current BG and FG models
        logp_bg = log_gmm_diag_pdf(Xsub, bgm.alpha, bgm.mu(1:Ddim,:), bgm.sigma_diag(1:↙
Ddim,:));    % Log-likelihood BG
        logp_fg = log_gmm_diag_pdf(Xsub, fgm.alpha, fgm.mu(1:Ddim,:), fgm.sigma_diag(1:↙
Ddim,:));    % Log-likelihood FG
```

```
        logpost_bg = logp_bg + log(prior_bg + realmin);↙
% Log posterior for BG
        logpost_fg = logp_fg + log(prior_fg + realmin);↙
% Log posterior for FG
        pred_fg = logpost_fg > logpost_bg;↙
% Predict FG if posterior higher than BG
        errs(di) = mean(pred_fg ~= gtBlocks);↙
% Compute blockwise error
    end
    errors_b(ci,:) = errs;↙
% Store errors for this C
end
% Plot Part (b)
figure('Name','Part (b)'); hold on;↙
% Initialize figure
cols = lines(numel(C_list)); leg = cell(numel(C_list),1);↙
% Colors and legend entries
for ci=1:numel(C_list)
    plot(dims_to_try, errors_b(ci,:), '-o','LineWidth',1.5,'MarkerSize',6,'Color',cols↙
(ci,:));      % Plot error vs DCT dims
    leg{ci} = sprintf('C = %d', C_list(ci));↙
% Prepare legend entry
end
xlabel('DCT Dimensions'); ylabel('Blockwise Error');↙
% Set axis labels
title('Part (b): Error vs Dimensions for different C');↙
% Set plot title
legend(leg,'Location','bestoutside'); grid on; hold off;↙
% Add legend and grid
saveas(gcf, fullfile(outputFolder, 'part_b_C_curves.png'));↙
% Save figure
% Output Statements
fprintf('Summary for Part B(numeric highlights):\n');↙
% Print summary header
for ci=1:numel(C_list)↙
% Loop over each C
    [val, idxd] = min(errors_b(ci,:));↙
% Find best error for this C
    fprintf('- For C=%2d best error = %.4f at D=%d\n', C_list(ci), val, dims_to_try↙
(idxd));
end
fprintf('Figure for Part B saved to folder\n');

% ----------------- FUNCTIONS -----------------
function [alpha, mu, sigma_diag] = em_gmm_diag(X, C, maxIter, tol, epsilon_cov)
[D, N] = size(X);                                                         % D =↙
feature dimension, N = number of samples
[init_idx, mu_init] = kmeans(X', C, 'MaxIter', 200, 'Replicates', 5);     %↙
Initialize cluster assignments & means with k-means
mu = mu_init'; alpha = zeros(C,1); sigma_diag = zeros(D,C);               % Pre-↙
```

```
allocate GMM parameters
for c=1:C
    members = (init_idx==c); Nc = sum(members);                    % Find↙
samples assigned to cluster c, count them
    if Nc==0                                                       % If no↙
points assigned, fallback initialization
        alpha(c) = 1/C;                                            % Equal↙
weight for empty component
        sigma_diag(:,c) = var(X,0,2)+epsilon_cov;                  %↙
Covariance from overall variance
        mu(:,c)=X(:,randi(N));                                     %↙
Randomly pick mean from data
    else
        alpha(c) = Nc/N;                                           %↙
Weight = fraction of points in cluster
        Xc=X(:,members); mu(:,c)=mean(Xc,2);                       % Mean↙
= average of assigned points
        sigma_diag(:,c) = var(Xc,0,2)+epsilon_cov;                 %↙
Diagonal covariance
    end
end
loglik_old = -inf;                                                 %↙
Initialize log-likelihood
for it=1:maxIter                                                   % EM↙
iterations
    logPc = zeros(C,N);                                            % Log↙
probability for each component
    for c=1:C
        diff = X - mu(:,c);                                        %↙
Difference from mean
        quad = sum((diff.^2)./sigma_diag(:,c),1);                  %↙
Quadratic term in Gaussian
        logdet = sum(log(sigma_diag(:,c)));                        % Log↙
determinant (product of diag elements)
        logPc(c,:) = -0.5*(D*log(2*pi) + logdet + quad);           % Log↙
likelihood for each sample
    end
    logNumer = bsxfun(@plus, logPc, log(alpha+realmin));           % Add↙
log mixture weights
    maxlog = max(logNumer,[],1);                                   % For↙
numerical stability
    logDen = maxlog + log(sum(exp(bsxfun(@minus, logNumer,maxlog)),1));  % Log-↙
sum-exp denominator
    R = exp(bsxfun(@minus, logNumer, logDen));                     %↙
Responsibilities (E-step)
    % (M-step)
    Nk = sum(R,2);                                                 %↙
Effective number of points per component
    alpha = Nk/N;                                                  %↙
Update mixture weights
```

```
    mu = bsxfun(@rdivide, X*R', Nk');                                   %↙
Update means
    for c=1:C
        diff = X - mu(:,c);
        sigma_diag(:,c) = ((diff.^2)*R(c,:)')./(Nk(c)+realmin)+epsilon_cov;     %↙
Update diagonal covariances
    end
    loglik = sum(logDen);                                               % Total↙
log-likelihood
    if abs(loglik-loglik_old)<tol*max(1,abs(loglik)), break; end        %↙
Convergence check
    loglik_old = loglik;
end
end

function logp = log_gmm_diag_pdf(X, alpha, mu, sigma_diag)
[D, N] = size(X); C = numel(alpha); logPc = zeros(C,N);               % Setup
for c=1:C
    diff = X - mu(:,c);                                                 %↙
Difference from mean
    quad = sum((diff.^2)./sigma_diag(:,c),1);                           % Quadratic↙
term
    logdet = sum(log(sigma_diag(:,c)));                                 % Log↙
determinant
    logPc(c,:) = -0.5*(D*log(2*pi)+logdet+quad);                        % Log↙
Gaussian PDF
end
logNumer = bsxfun(@plus, logPc, log(alpha+realmin));                    % Add log↙
mixture weights
maxlog = max(logNumer,[],1);                                            % Numerical↙
stability
logp = maxlog + log(sum(exp(bsxfun(@minus, logNumer,maxlog)),1));       % Log-sum-↙
exp to compute mixture PDF
end
```