APPENDIX

Code

main_train

```
% main_train.m - ES-CCGAN simplified training script for remote sensing dehazing
clear; clc;
%% Setup paths
hazyFolder = fullfile('dataset', 'hazy');
clearFolder = fullfile('dataset', 'clear');
outputFolder = fullfile('outputs');
modelFolder = fullfile('models');
if ~exist(outputFolder, 'dir'); mkdir(outputFolder); end
if ~exist(modelFolder, 'dir'); mkdir(modelFolder); end
%% Load image list
hazyFiles = dir(fullfile(hazyFolder, '*.png'));
clearFiles = dir(fullfile(clearFolder, '*.png'));
assert(numel(hazyFiles) == numel(clearFiles), 'Mismatch in dataset.');
%% Network Parameters
numEpochs = 2;
learningRate = 1e-4;
%% Create minimal Generator G and F
inputSize = [256 256 3];
numChannels = 3;
layersG = [
    imageInputLayer(inputSize, 'Normalization','none')
    convolution2dLayer(3,64,'Padding','same')
    reluLayer
    convolution2dLayer(3,numChannels,'Padding','same')
```

```
    tanhLayer
];
G = dlnetwork(layerGraph(layersG));
F = dlnetwork(layerGraph(layersG));
%% Discriminators Dx and Dy
layersD = [
    imageInputLayer(inputSize, 'Normalization', 'none')
    convolution2dLayer(3, 64, 'Padding', 'same')
    reluLayer
    fullyConnectedLayer(1)
    sigmoidLayer
];
Dx = dlnetwork(layerGraph(layersD));
Dy = dlnetwork(layerGraph(layersD));
%% Load VGG16 for perceptual loss
vgg = vgg16;
featureLayer = 'relu3_3';
%% Training loop
for epoch = 1:numEpochs
    fprintf('Epoch %d/%d\n', epoch, numEpochs);
    for i = 1:numel(hazyFiles)
        % Load and preprocess images
        hazy = im2single(imread(fullfile(hazyFolder, hazyFiles(i).name)));
        clearGT = im2single(imread(fullfile(clearFolder, clearFiles(i).name)));
        hazy = imresize(hazy, inputSize(1:2));
        clearGT = imresize(clearGT, inputSize(1:2));
        dlHazy = dlarray(hazy * 2 - 1, 'SSC');
```

```
dlClearGT = dlarray(clearGT * 2 - 1, 'SSC');

% Forward pass

dlClearFake = forward(G, dlHazy);

dlHazyFake = forward(F, dlClearGT);

dlHazyRec = forward(F, dlClearFake);

dlClearRec = forward(G, dlHazyFake);

% Discriminator outputs

predFakeY = forward(Dy, dlClearFake);

predRealY = forward(Dy, dlClearGT);

predFakeX = forward(Dx, dlHazyFake);

predRealX = forward(Dx, dlHazy);

% GAN loss

bce = @(p, t) -mean(t .* log(p + 1e-8) + (1 - t) .* log(1 - p + 1e-8));

lossGAN_G = bce(predFakeY, ones(size(predFakeY)));

lossGAN_F = bce(predFakeX, ones(size(predFakeX)));

lossGAN_Dy = (bce(predRealY, ones(size(predRealY))) + bce(predFakeY,
zeros(size(predFakeY)))) / 2;

lossGAN_Dx = (bce(predRealX, ones(size(predRealX))) + bce(predFakeX,
zeros(size(predFakeX)))) / 2;

% Cycle-Consistency Loss

lossCycle = mean(abs(dlHazy - dlHazyRec), 'all') + mean(abs(dlClearGT -
dlClearRec), 'all');

% Edge-Sharpening Loss

edgeGrad = @(img) sqrt(imfilter(extractdata(img), fspecial('sobel'),
'replicate').^2);

lossEdge = mean(abs(edgeGrad(dlHazy) - edgeGrad(dlClearFake)), 'all');

% Perceptual Loss using VGG16

fReal = activations(vgg, im2uint8((extractdata(dlClearGT) + 1) / 2),
featureLayer);
```

```
        fFake = activations(vgg, im2uint8((extractdata(dlClearFake) + 1) / 2),
featureLayer);

        lossPercep = mean(abs(fReal(:) - fFake(:)));

        % Total Generator Loss

        lossG = lossGAN_G + 10 * lossCycle + 5 * lossEdge + 0.1 * lossPercep;

        % Display training progress

        fprintf('Image %d | G-Loss: %.4f | Cycle: %.4f | Edge: %.4f | Perc: %.4f\n', ...

            i, lossG, lossCycle, lossEdge, lossPercep);

    end

end

%% Save generator model

save(fullfile(modelFolder, 'generator.mat'), 'G');

disp('Model saved.');
```

inference_dehaze

```
clc; clear; close all;

fprintf('🔁 Loading generator model...\n');

% Load the trained generator

load("models/generator.mat", "G");

fprintf('✅ Generator loaded successfully.\n');

% Set paths

hazyFolder = fullfile('dataset', 'hazy');

cleanFolder = fullfile('dataset', 'clear');

outputFolder = fullfile('outputs');

if ~exist(outputFolder, 'dir')

    mkdir(outputFolder);

end

% Get hazy image files

hazyFiles = dir(fullfile(hazyFolder, '*.png'));

nFiles = length(hazyFiles);

fprintf('🔄 Processing %d images...\n', nFiles);

% Initialize accumulators

metrics = [];

for i = 1:nFiles

    hazyName = hazyFiles(i).name;

    hazyPath = fullfile(hazyFolder, hazyName);
```

```
hazyImg = im2double(imread(hazyPath));
% Resize and ensure 3 channels
inputImg = imresize(hazyImg, [256 256]);
if size(inputImg, 3) == 1
    inputImg = repmat(inputImg, [1 1 3]);
end
% Inference
tic;
inputDL = dlarray(single(inputImg), 'SSC');
outputDL = predict(G, inputDL);
dehazedImg = double(gather(extractdata(outputDL)));
dehazedImg = min(max(dehazedImg, 0), 1);
inferenceTime = toc;
% Enhancement
for c = 1:3
    channel = dehazedImg(:,:,c);
    pLow = prctile(channel(:), 1);
    pHigh = prctile(channel(:), 99);
    if pHigh > pLow
        channel = imadjust(channel, [pLow pHigh], []);
    else
        channel = mat2gray(channel);
    end
```

```
    dehazedImg(:,:,c) = channel;
end
try
    lab = rgb2lab(dehazedImg);
    L = lab(:,:,1) / 100;
    L_eq = adapthisteq(L, 'ClipLimit', 0.01, 'NumTiles', [8 8]);
    lab(:,:,1) = L_eq * 100;
    dehazedImg = lab2rgb(lab);
    dehazedImg = min(max(dehazedImg, 0), 1);
catch
    warning('⚠ LAB enhancement failed on %s', hazyName);
end
% Gamma correction & sharpening
gamma = 1.1;
dehazedImg = dehazedImg .^ (1 / gamma);
dehazedImg = imsharpen(dehazedImg, 'Radius', 1.2, 'Amount', 0.8);
dehazedImg = imgaussfilt(dehazedImg, 0.2);
dehazedImg = min(max(dehazedImg, 0), 1);
% Load Ground Truth
[~, baseName, ~] = fileparts(hazyName);
if contains(baseName, '_hazy')
    baseName = erase(baseName, '_hazy');
end
```

```
gtName = [baseName '_GT.png'];
cleanPath = fullfile(cleanFolder, gtName);
if exist(cleanPath, 'file')
    groundTruth = im2double(imread(cleanPath));
    groundTruth = imresize(groundTruth, [256 256]);
    try
        dehazedImg = imhistmatch(dehazedImg, groundTruth);
    catch
        warning('⚠ Histogram match failed on %s', hazyName);
    end
    % Metrics
    dehazedImg = im2double(dehazedImg);
    groundTruth = im2double(groundTruth);
    try
        psnrVal = psnr(dehazedImg, groundTruth);
    catch
        psnrVal = NaN;
    end


    try
        ssimVal = ssim(dehazedImg, groundTruth);
    catch
        ssimVal = NaN;
```

```
    end
    try
        fsimVal = fsim(dehazedImg, groundTruth);
    catch
        fsimVal = NaN;
    end
    mseVal = immse(dehazedImg, groundTruth);
    maeVal = mean(abs(dehazedImg(:) - groundTruth(:)));
    accuracy = 100 * (1 - maeVal);
    % Store metrics
    metrics(end+1,:) = [psnrVal, ssimVal, fsimVal, mseVal, maeVal, accuracy,
inferenceTime];
    % === Visualization ===
    figure('Name', ['Dehazing Result: ' hazyName], 'Color', 'k');
    subplot(1,3,1); imshow(inputImg);      title('Hazy', 'Color', 'w');
    subplot(1,3,2); imshow(dehazedImg);    title('Dehazed', 'Color', 'w');
    subplot(1,3,3); imshow(groundTruth);   title('Ground Truth', 'Color', 'w');

    comparisonImg = cat(2, inputImg, dehazedImg, groundTruth);
    imwrite(comparisonImg, fullfile(outputFolder, ['comparison_' baseName
'.png']));
    % === Bar Graph with Values ===
    figure('Name', ['Metrics: ' hazyName], 'Color', 'w');
```

```
        barVals = [psnrVal, ssimVal, fsimVal, mseVal, maeVal, accuracy,
inferenceTime];
        labels = {'PSNR','SSIM','FSIM','MSE','MAE','Accuracy(%)','Infer(sec)'};
        b = bar(barVals, 'FaceColor', 'flat');
        set(gca, 'XTickLabel', labels, 'XTickLabelRotation', 30, 'FontSize', 10);
        title(['Metrics - ' hazyName], 'Interpreter','none');
        xtips = b.XEndPoints;
        ytips = b.YEndPoints;
        for j = 1:numel(barVals)
            text(xtips(j), ytips(j)+0.01, sprintf('%.3f', barVals(j)), ...
                'HorizontalAlignment','center','FontSize',9, 'Color','k');
        end
        saveas(gcf, fullfile(outputFolder, ['metrics_' baseName '.png']));
    else
        fprintf('⚠ Ground truth not found: %s\n', gtName);
    end
    % Save dehazed image
    outputPath = fullfile(outputFolder, ['dehazed_' hazyName]);
    imwrite(dehazedImg, outputPath);
end
% === Summary CSV ===
avgMetrics = mean(metrics, 1, 'omitnan');
headers = {'PSNR','SSIM','FSIM','MSE','MAE','Accuracy','InferenceTime'};
```

```
csvPath = fullfile(outputFolder, 'metrics_summary.csv');

fid = fopen(csvPath, 'w');

fprintf(fid, '%s,', headers{1:end-1});

fprintf(fid, '%s\n', headers{end});

fprintf(fid, '%.4f,', avgMetrics(1:end-1));

fprintf(fid, '%.4f\n', avgMetrics(end));

fclose(fid);

fprintf('\n === AVERAGE METRICS ===\n');

fprintf(' Avg PSNR       : %.2f dB\n', avgMetrics(1));

fprintf(' Avg SSIM       : %.4f\n', avgMetrics(2));

fprintf(' Avg FSIM       : %.4f\n', avgMetrics(3));

fprintf(' Avg MSE        : %.6f\n', avgMetrics(4));

fprintf(' Avg MAE        : %.6f\n', avgMetrics(5));

fprintf('Avg Accuracy   : %.2f %%\n', avgMetrics(6));

fprintf(' Avg Inference  : %.3f sec/image\n', avgMetrics(7));

fprintf(' All metrics saved to: %s\n', csvPath);
```