1.morphology

clc; clear; close all;

img = imread("C:\Users\Swathika\Downloads\3d-boy-cartoon-mascot-cute-600nw-2580453711.jpg");

if size(img,3)==3, gray=rgb2gray(img);

else, gray=img;

end

se=strel('line',11,90);

figure;

subplot(3,4,1), imshow(img), title('Original');

subplot(3,4,2), imshow(gray), title('Gray');

subplot(3,4,3), imshow(imerode(gray,se)), title('Erosion');

subplot(3,4,4), imshow(imdilate(gray,se)), title('Dilation');

subplot(3,4,5), imshow(imopen(gray,se)), title('Opening');

subplot(3,4,6), imshow(imclose(gray,se)), title('Closing');

subplot(3,4,7), imshow(imsubtract(imdilate(gray,se), imerode(gray,se))), title('Gradient');

subplot(3,4,8), imshow(imtophat(gray,se)), title('Top Hat');

subplot(3,4,9), imshow(imbothat(gray,se)), title('Black Hat');

subplot(3,4,10), imshow(imdilate(gray,strel('disk',6))), title('Elliptical');

subplot(3,4,11), imshow(imdilate(gray,strel('rectangle',[11 11]))), title('Rectangular');

subplot(3,4,12), imshow(imdilate(gray,strel('arbitrary',[0 1 0;1 1 1;0 1 0]))), title('Cross');

2. EDGE DETECTION

clc;

clear;

close all;

img = imread("/MATLAB Drive/elep.jpg");

if size(img,3)==3, g=rgb2gray(img);

else, g=img;

end

g = im2double(g);

figure;

t={'Canny','Sobel','Prewitt','Roberts','LoG','Zerocross'};

m={'canny','sobel','prewitt','roberts','log','zerocross'};

subplot(2,4,1), imshow(img), title('Original');

subplot(2,4,2), imshow(g), title('Grayscale');

for i=1:6, subplot(2,4,i+2), imshow(edge(g,m{i})), title([t{i} ' Edge']);

end

3.HISTOGRAM

clc; clear; close all;

img=imread("/MATLAB Drive/images (13).jpg");

if size(img,3)==3, img=rgb2gray(img); end

resized=imresize(img,0.5);

fprintf('Resized: %dx%d\n', size(resized,1), size(resized,2));

eq=histeq(resized);

imwrite([resized eq],'Histo1\_res.png');

[h,~]=imhist(resized); cdf=cumsum(h); cdf=cdf\*max(h)/max(cdf);

figure; yyaxis left, bar(h,'r'); ylabel('Histogram');

yyaxis right, plot(cdf,'b','LineWidth',1.5); ylabel('CDF');

xlabel('Intensity'); legend('Histogram','CDF'); title('Histogram & CDF'); saveas(gcf,'histo2\_cdf.png');

figure;

subplot(1,2,1), imshow(imread('Histo1\_res.png')), title('Original & Equalized');

subplot(1,2,2), imshow(imread('histo2\_cdf.png')), title('CDF & Histogram');

saveas(gcf,'histogram\_result3.png');

4.MATCHING TEMPLATE

clc;

clear;

% Read the main image in grayscale

img = imread("/MATLAB Drive/images (13).jpg");

template=imread("/MATLAB Drive/images (13).jpg")

if size(img, 3) == 3

img = rgb2gray(img);

end

% --- Choose a smaller patch as template ---

% Example: take a 50x50 region from the top-left corner

template = img(50:100, 50:100);

% Get template size

[h, w] = size(template);

% Methods list

methods = {'normxcorr2', 'ssd', 'sad', 'correlation', 'normalized\_correlation', 'normalized\_difference'};

% Create figure

figure('Name', 'Template Matching: Heatmap | Score | Detected Match', 'NumberTitle', 'off');

for i = 1:6

method = methods{i};

img\_copy = img;

switch method

case 'normxcorr2'

% Normalized Cross-Correlation

result = normxcorr2(template, img);

[max\_val, idx] = max(result(:));

[ypeak, xpeak] = ind2sub(size(result), idx);

top\_left = [xpeak - w, ypeak - h];

score = max\_val;

heatmap = result;

otherwise

% Manual sliding window methods

result = zeros(size(img) - size(template) + 1);

template\_d = double(template);

for y = 1:size(result,1)

for x = 1:size(result,2)

patch = img(y:y+h-1, x:x+w-1);

patch\_d = double(patch);

switch method

case 'ssd'

diff = patch\_d - template\_d;

result(y,x) = sum(diff(:).^2);

case 'sad'

diff = abs(patch\_d - template\_d);

result(y,x) = sum(diff(:));

case 'correlation'

result(y,x) = sum(patch\_d(:) .\* template\_d(:));

case 'normalized\_correlation'

patch\_norm = (patch\_d - mean(patch\_d(:))) / std(patch\_d(:));

template\_norm = (template\_d - mean(template\_d(:))) / std(template\_d(:));

result(y,x) = sum(patch\_norm(:) .\* template\_norm(:));

case 'normalized\_difference'

diff = patch\_d - template\_d;

result(y,x) = sum(diff(:).^2) / sum(template\_d(:).^2);

end

end

end

% Find best score

if ismember(method, {'ssd', 'sad', 'normalized\_difference'})

[score, idx] = min(result(:));

else

[score, idx] = max(result(:));

end

[y, x] = ind2sub(size(result), idx);

top\_left = [x, y];

heatmap = result;

end

bottom\_right = top\_left + [w - 1, h - 1];

% --- Normalize heatmap for visualization ---

heatmap = mat2gray(heatmap);

% Heatmap subplot

subplot(6, 3, (i-1)\*3 + 1);

imagesc(heatmap); colormap gray;

title('Matching Heatmap'); axis off;

% Score subplot

subplot(6, 3, (i-1)\*3 + 2);

text(0.5, 0.5, {method, sprintf('Score: %.4f', score)}, ...

'HorizontalAlignment', 'center', 'FontSize', 12, 'BackgroundColor', 'yellow');

axis off;

% Detected Match subplot

subplot(6, 3, (i-1)\*3 + 3);

imshow(img\_copy); hold on;

rectangle('Position', [top\_left(1), top\_left(2), w, h], 'EdgeColor', 'r', 'LineWidth', 2);

title('Detected Match'); hold off;

end

5.HARRIS CORNER

filename = "/MATLAB Drive/2i3TK603pRdrEiN7mnoSa6EurvY-mobile.jpg";

img = imread(filename);

gray = double(rgb2gray(img));

cStrength = imdilate(cornermetric(gray, 'Harris'), strel('disk',1));

mask = cStrength > 0.01 \* max(cStrength(:));

cornerImg = img;

cornerImg(repmat(mask, [1,1,3])) = 0;

cornerImg(:,:,1) = cornerImg(:,:,1) + uint8(255 \* mask);

figure, imshow(img), title('Original Image');

figure, imshow(cornerImg), title('Harris Corners')

6.BRUTE FORCE

% ---- Read and convert images to grayscale ----

img1 = rgb2gray(imread('/MATLAB Drive/download (15).jpg'));

img2 = rgb2gray(imread('/MATLAB Drive/heid22.jpg'));

% ---- Detect corners (no toolbox) ----

corners1 = corner(img1, 150); % top 150 corners

corners2 = corner(img2, 150);

% ---- Extract small patches for brute-force matching ----

patchSize = 21;

halfSize = floor(patchSize/2);

features1 = extractCornerPatches(img1, corners1, halfSize);

features2 = extractCornerPatches(img2, corners2, halfSize);

% ---- Match features (nearest neighbor) ----

indexPairs = knnsearch(features2, features1);

% ---- Visualization WITHOUT showMatchedFeatures ----

figure;

imshowpair(img1, img2, 'montage');

hold on;

% Shift x-coordinates of second image for plotting

offset = size(img1,2);

for i = 1:size(corners1,1)

pt1 = corners1(i,:);

pt2 = corners2(indexPairs(i),:);

pt2(1) = pt2(1) + offset; % shift right

plot(pt1(1), pt1(2), 'ro');

plot(pt2(1), pt2(2), 'go');

line([pt1(1) pt2(1)], [pt1(2) pt2(2)], 'Color','y');

end

title('Brute-Force Corner Matching (No Toolbox)');

% ---- Helper function ----

function features = extractCornerPatches(img, pts, halfSize)

numPts = size(pts,1);

features = zeros(numPts, (2\*halfSize+1)^2);

for i = 1:numPts

r = round(pts(i,2));

c = round(pts(i,1));

r1 = max(1, r-halfSize); r2 = min(size(img,1), r+halfSize);

c1 = max(1, c-halfSize); c2 = min(size(img,2), c+halfSize);

patch = imcrop(img, [c1 r1 c2-c1 r2-r1]);

patch = imresize(patch, [2\*halfSize+1, 2\*halfSize+1]);

features(i,:) = double(patch(:))';

end

end

7.WATERSHED

clc;

clear;

close all;

% --------- Load image ----------

% ✅ Fix: use correct path and quotes

img = imread('MATLAB Drive/pen.jpg');

img = im2double(img);

[h, w, ~] = size(img);

% --------- Gradient for watershed ----------

I = rgb2gray(img);

I = imgaussfilt(I, 1.0);

G = imgradient(I);

% --------- Ask user how many regions ----------

numRegions = input('Enter the number of regions to segment: ');

% --------- Interactive marker drawing ----------

figure('Name','Draw markers (double-click to finish each)');

imshow(img);

title('Draw ROI markers (double-click each to finish)');

hold on;

markerMask = zeros(h, w, 'uint16');

markerColors = containers.Map('KeyType','double','ValueType','any');

for markerID = 1:numRegions

% Assign random bright color

c = 0.2 + 0.8\*rand(1,3);

markerColors(markerID) = c;

% Let user draw ROI

disp(['Draw marker: ' num2str(markerID)]);

hRoi = drawfreehand('Color', c, 'FaceAlpha', 0.15, 'LineWidth', 1.5);

mask = createMask(hRoi);

markerMask(mask) = markerID;

% Display marker label

pos = mean(hRoi.Position,1);

text(pos(1), pos(2), ['Marker: ' num2str(markerID)], ...

'Color','w','FontSize',12,'FontWeight','bold','BackgroundColor','k');

end

% --------- Show markers ----------

allIDs = 1:numRegions;

colorArray = zeros(numRegions, 3);

for k = allIDs

if isKey(markerColors, k)

colorArray(k, :) = markerColors(k);

else

colorArray(k, :) = [0 0 0]; % unused ID is black

end

end

figure;

imshow(label2rgb(markerMask, colorArray, 'w'));

title('User Markers');

% --------- Watershed segmentation ----------

% ✅ Ensure we are using the built-in watershed

clear watershed; % clears any variable named 'watershed'

G\_forced = imimposemin(G, markerMask > 0);

Lw = watershed(G\_forced); % should now work properly

% --------- Assign basins to markers ----------

assignedMap = zeros(h, w, 'uint16');

basins = unique(Lw);

basins(basins == 0) = [];

for k = 1:numel(basins)

lab = basins(k);

pix = (Lw == lab);

mvals = markerMask(pix);

mvals = mvals(mvals > 0);

if ~isempty(mvals)

assignedMap(pix) = mode(mvals);

end

end

% --------- Fill unassigned basins ----------

unassigned = (assignedMap == 0);

if any(unassigned(:))

[~, idx] = bwdist(markerMask > 0);

nearestMarkerId = markerMask(idx);

assignedMap(unassigned) = nearestMarkerId(unassigned);

end

% --------- Build segmentation ----------

segRGB = zeros(h, w, 3);

for id = 1:numRegions

if isKey(markerColors, id)

c = markerColors(id);

m = (assignedMap == id);

segRGB(:,:,1) = segRGB(:,:,1) + m \* c(1);

segRGB(:,:,2) = segRGB(:,:,2) + m \* c(2);

segRGB(:,:,3) = segRGB(:,:,3) + m \* c(3);

end

end

% --------- Overlay ----------

alpha = 0.4;

overlay = (1 - alpha) \* img + alpha \* segRGB;

boundaries = (Lw == 0);

overlay(repmat(boundaries, [1 1 3])) = 1;

figure;

imshow(overlay);

title('Watershed Segmentation with Markers');

8.SVM

clc; clear; close all;

% ===== Parameters =====

SZ = 20; % digit size (20x20)

CLASS\_N = 10; % number of classes (digits 0-9)

DIGITS\_FN = "/MATLAB Drive/digits.png";

%% ===== Utilities =====

function vis = mosaic(w, imgs)

n = numel(imgs);

[h, w\_img, c] = size(imgs{1});

vis = zeros(h\*w, w\_img\*w, c, 'like', imgs{1});

for i = 1:n

x = mod(i-1, w);

y = floor((i-1) / w);

if y >= w, break; end

vis( y\*h+1:(y+1)\*h, x\*w\_img+1:(x+1)\*w\_img, : ) = imgs{i};

end

end

function cells = split2d(img, cell\_size, flatten)

if nargin < 3, flatten = true; end

[h, w] = size(img);

sx = cell\_size(1); sy = cell\_size(2);

cells = mat2cell(img, repmat(sy,1,h/sy), repmat(sx,1,w/sx));

if flatten

cells = reshape(cells, [],1);

end

end

function img2 = deskew(img, SZ)

[X,Y] = meshgrid(1:SZ, 1:SZ);

m00 = sum(img(:));

if m00 == 0

img2 = img; return;

end

m10 = sum(X(:).\*double(img(:)));

m01 = sum(Y(:).\*double(img(:)));

m20 = sum((X(:).^2).\*double(img(:)));

m02 = sum((Y(:).^2).\*double(img(:)));

m11 = sum((X(:).\*Y(:)).\*double(img(:)));

mu02 = m02/m00 - (m01/m00)^2;

mu11 = m11/m00 - (m10/m00)\*(m01/m00);

if abs(mu02) < 1e-2

img2 = img; return;

end

skew = mu11/mu02;

M = [1 skew -0.5\*SZ\*skew; 0 1 0];

tform = affine2d(M');

img2 = imwarp(img, tform, 'OutputView', imref2d([SZ SZ]));

end

function out = highlightError(img, correct)

if size(img,3) == 1

img\_rgb = repmat(img,1,1,3);

else

img\_rgb = img;

end

if correct

out = img\_rgb;

out(:,:,2:3) = 0;

else

out = img\_rgb;

end

end

%% ===== Load Data =====

disp("Loading digits...");

digits\_img = imread(DIGITS\_FN);

if size(digits\_img,3)==3

digits\_img = rgb2gray(digits\_img);

end

digits = split2d(digits\_img, [SZ SZ], true);

labels = repelem((0:CLASS\_N-1)', numel(digits)/CLASS\_N);

%% ===== Preprocessing =====

disp("Preprocessing...");

rng(321);

idx = randperm(numel(digits));

digits = digits(idx);

labels = labels(idx);

% Deskew

digits\_deskewed = cellfun(@(x) deskew(x,SZ), digits, 'UniformOutput', false);

% Deskew and convert to single precision

digits\_deskewed = cellfun(@(x) im2single(deskew(x,SZ)), digits, 'UniformOutput', false);

% Define small cell size for HOG

cellSize = [4 4];

% Extract HOG features safely

hogFeatures = cellfun(@(x) safeHOG(x, cellSize), digits\_deskewed, 'UniformOutput', false);

samples = vertcat(hogFeatures{:});

% Safe HOG extraction function

function features = safeHOG(img, cellSize)

try

features = extractHOGFeatures(img, 'CellSize', cellSize);

catch

features = zeros(1, prod(cellSize)); % fallback in case of failure

end

end

%% ===== Train/Test Split =====

train\_n = round(0.9 \* size(samples,1));

digits\_train = digits\_deskewed(1:train\_n);

digits\_test = digits\_deskewed(train\_n+1:end);

samples\_train = samples(1:train\_n,:);

samples\_test = samples(train\_n+1:end,:);

labels\_train = labels(1:train\_n);

labels\_test = labels(train\_n+1:end);

figure;

imshow(mosaic(25, cellfun(@(x) repmat(x,1,1,3), digits\_test, 'UniformOutput', false)));

title("Test Set (10%)");

%% ===== k-Nearest Neighbors =====

disp("Training kNN...");

knn = fitcknn(samples\_train, labels\_train, 'NumNeighbors', 4);

resp\_knn = predict(knn, samples\_test);

err\_knn = mean(labels\_test ~= resp\_knn);

fprintf('kNN Error: %.2f %%\n', err\_knn\*100);

conf\_knn = confusionmat(labels\_test, resp\_knn);

disp('Confusion matrix (kNN):'); disp(conf\_knn);

vis\_knn = cellfun(@(img,correct) highlightError(img, correct), digits\_test, ...

num2cell(resp\_knn==labels\_test), 'UniformOutput', false);

figure;

imshow(mosaic(25, vis\_knn));

title("kNN Result");

%% ===== SVM =====

disp("Training SVM...");

svm = fitcecoc(samples\_train, labels\_train, ...

'Learners', templateSVM('KernelFunction','rbf','BoxConstraint',2.67,'KernelScale',1/5.383));

resp\_svm = predict(svm, samples\_test);

err\_svm = mean(labels\_test ~= resp\_svm);

fprintf('SVM Error: %.2f %%\n', err\_svm\*100);

conf\_svm = confusionmat(labels\_test, resp\_svm);

disp('Confusion matrix (SVM):'); disp(conf\_svm);

vis\_svm = cellfun(@(img,correct) highlightError(img, correct), digits\_test, ...

num2cell(resp\_svm==labels\_test), 'UniformOutput', false);

figure;

imshow(mosaic(25, vis\_svm));

title("SVM Result");

%% ===== Save SVM Model =====

disp("Saving SVM model to digits\_svm.mat...");

save('digits\_svm.mat','svm');

KMEANS

function kmeans\_visualization()

cluster\_n=5; img\_size=512; colors=hsv2rgb([linspace(0,1,cluster\_n)',ones(cluster\_n,2)]);

while true

points=[]; labels=[];

for i=1:cluster\_n

c=randi([img\_size\*0.25,img\_size\*0.75],1,2); pts=randn(floor(img\_size/cluster\_n\*5),2)\*randi([10,40])+c;

points=[points;pts]; labels=[labels;repmat(i,size(pts,1),1)];

end

[lbl,~]=kmeans(points,cluster\_n,'MaxIter',100); figure(1); clf; hold on;

for i=1:cluster\_n, scatter(points(lbl==i,1),points(lbl==i,2),5,repmat(colors(i,:),sum(lbl==i),1),'filled'); end

if waitforbuttonpress && double(get(gcf,'CurrentCharacter'))==27, break; end

end

end

9.DIMENSION

clc; clear; close all;

%% ---------- Parameters ----------

imgSize = [64, 64]; % resize images to 64x64

nPCA = 200; % number of PCA components

nDict = 200; % number of dictionary atoms

sparsity = 20; % OMP max non-zero coefficients

nShow = 5; % number of faces to display

%% ---------- Step 1: Load and preprocess images ----------

url = 'https://www.cl.cam.ac.uk/research/dtg/attarchive/pub/data/att\_faces.zip';

zipFile = 'att\_faces.zip';

dataFolder = 'att\_faces';

if ~exist(dataFolder,'dir')

if ~exist(zipFile,'file')

fprintf('Downloading Olivetti Faces dataset...\n');

websave(zipFile,url);

end

unzip(zipFile,dataFolder);

end

% Read and resize images

imageFiles = dir(fullfile(dataFolder,'s\*','\*.pgm'));

num\_samples = length(imageFiles);

X = zeros(num\_samples, prod(imgSize));

for i = 1:num\_samples

img = imread(fullfile(imageFiles(i).folder,imageFiles(i).name));

img = imresize(img, imgSize); % resize

X(i,:) = double(img(:))'; % flatten

end

fprintf('Loaded and resized %d images.\n', num\_samples);

%% ---------- Step 2: PCA ----------

[coeff, score, ~, ~, ~, mu] = pca(X, 'NumComponents', nPCA);

X\_pca = score; % num\_samples x nPCA

%% ---------- Step 3: Dictionary Learning (k-means approximation) ----------

[idx, D] = kmeans(X\_pca, nDict, 'MaxIter', 500, 'Replicates', 3);

D = D'; % columns = dictionary atoms

D = D ./ vecnorm(D); % normalize columns

%% ---------- Step 4: Sparse coding using OMP ----------

X\_dict = zeros(nDict, num\_samples);

for i = 1:num\_samples

X\_dict(:,i) = omp(D, X\_pca(i,:)', sparsity);

end

%% ---------- Step 5: Reconstruct images ----------

X\_reconstructed\_pca = D \* X\_dict;

X\_reconstructed = (X\_reconstructed\_pca' \* coeff') + mu;

%% ---------- Step 6: Display ----------

figure('Name','Olivetti Face Reconstruction','Position',[100 100 1200 400]);

for i = 1:nShow

% Original

subplot(2,nShow,i);

imshow(reshape(X(i,:), imgSize), []);

title('Original');

% Reconstructed

subplot(2,nShow,i+nShow);

imshow(reshape(X\_reconstructed(i,:), imgSize), []);

title('Reconstructed');

end

%% -------- Helper Function: Orthogonal Matching Pursuit --------

function coef = omp(D, y, sparsity)

n\_atoms = size(D,2);

coef = zeros(n\_atoms,1);

residual = y;

support = [];

for k = 1:sparsity

projections = abs(D' \* residual);

[~, idx] = max(projections);

if ismember(idx, support), break; end

support = [support idx];

Ds = D(:,support);

coef\_support = Ds \ y;

coef(support) = coef\_support;

residual = y - Ds \* coef\_support;

if norm(residual) < 1e-6, break; end

end

end

10.EVALUATING

clc; clear; close all;

%% Step 1: Load digit dataset

digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos','nndatasets','DigitDataset');

imds = imageDatastore(digitDatasetPath, ...

'IncludeSubfolders',true, ...

'LabelSource','foldernames');

% Resize to 8x8 to match Python example

imds.ReadFcn = @(filename) imresize(im2double(imread(filename)), [8 8]);

% Convert to numeric arrays

numSamples = numel(imds.Files);

X = zeros(numSamples, 64);

y = zeros(numSamples,1);

for i = 1:numSamples

img = readimage(imds,i);

X(i,:) = img(:)'; % flatten 8x8 -> 64

y(i) = double(imds.Labels(i));

end

%% Step 2: Train-test split (70/30)

cv = cvpartition(y, 'HoldOut', 0.3);

X\_train = X(training(cv),:);

y\_train = y(training(cv),:);

X\_test = X(test(cv),:);

y\_test = y(test(cv),:);

%% Step 3: Train Random Forest classifier

model = TreeBagger(100, X\_train, y\_train, 'OOBPrediction', 'On', 'Method', 'classification');

%% Step 4: Predict on test set

y\_pred = str2double(predict(model, X\_test));

%% Step 5: Confusion matrix and accuracy

confMat = confusionmat(y\_test, y\_pred);

disp('Confusion Matrix:');

disp(confMat);

accuracy = sum(y\_pred == y\_test)/numel(y\_test);

fprintf('Overall Accuracy: %.4f\n\n', accuracy);

%% Step 6: Per-class Precision, Recall, F1

numClasses = size(confMat,1);

precision = zeros(numClasses,1);

recall = zeros(numClasses,1);

f1 = zeros(numClasses,1);

support = sum(confMat,2); % true labels per class

for i = 1:numClasses

TP = confMat(i,i);

FP = sum(confMat(:,i)) - TP;

FN = sum(confMat(i,:)) - TP;

precision(i) = TP / (TP + FP);

recall(i) = TP / (TP + FN);

f1(i) = 2 \* (precision(i)\*recall(i)) / (precision(i)+recall(i));

end

% Display table

fprintf('Class\tPrecision\tRecall\t\tF1-Score\tSupport\n');

for i = 1:numClasses

fprintf('%d\t%.3f\t\t%.3f\t\t%.3f\t\t%d\n', i-1, precision(i), recall(i), f1(i), support(i));

end

%% Step 7: Macro-average

macro\_precision = mean(precision);

macro\_recall = mean(recall);

macro\_f1 = mean(f1);

%% Step 8: Weighted-average

weighted\_precision = sum(precision .\* support) / sum(support);

weighted\_recall = sum(recall .\* support) / sum(support);

weighted\_f1 = sum(f1 .\* support) / sum(support);

%% Step 9: Micro-average

TP\_micro = sum(diag(confMat));

FP\_micro = sum(confMat(:)) - TP\_micro;

FN\_micro = FP\_micro;

micro\_precision = TP\_micro / (TP\_micro + FP\_micro);

micro\_recall = TP\_micro / (TP\_micro + FN\_micro);

micro\_f1 = 2 \* (micro\_precision \* micro\_recall) / (micro\_precision + micro\_recall);

%% Step 10: Display averages

fprintf('\nAverages:\n');

fprintf('Macro Precision: %.3f, Recall: %.3f, F1: %.3f\n', macro\_precision, macro\_recall, macro\_f1);

fprintf('Weighted Precision: %.3f, Recall: %.3f, F1: %.3f\n', weighted\_precision, weighted\_recall, weighted\_f1);

fprintf('Micro Precision: %.3f, Recall: %.3f, F1: %.3f\n', micro\_precision, micro\_recall, micro\_f1);

%% Step 11: Plot confusion matrix

figure;

cm = confusionchart(y\_test, y\_pred);

cm.Title = 'Digit Classification Confusion Matrix';

cm.RowSummary = 'row-normalized';

cm.ColumnSummary = 'column-normalized';