**Code 1 selecting Best 300 images in healthy patients**

% Define the main folder containing subfolders for each patient

main\_folder = 'E:\IRS New Data\Healthy Patients'; % Update with your path

output\_folder = 'E:\IRS New Data\Updated Healthy Patients'; % Folder to save selected images

% Create the output directory if it doesn't exist

if ~exist(output\_folder, 'dir')

mkdir(output\_folder);

end

% Get all patient subfolders

patient\_folders = dir(fullfile(main\_folder, '\*'));

patient\_folders = patient\_folders([patient\_folders.isdir] & ~ismember({patient\_folders.name}, {'.', '..'}));

% Loop through each patient's subfolder

for i = 1:length(patient\_folders)

patient\_name = patient\_folders(i).name;

patient\_folder = fullfile(main\_folder, patient\_name);

image\_files = dir(fullfile(patient\_folder, '\*.dcm')); % DICOM format

num\_images = length(image\_files);

if num\_images < 300

fprintf('Skipping %s (less than 300 images)\n', patient\_name);

continue;

end

% Initialize storage for images and scores

image\_data = cell(1, num\_images);

intensity\_scores = zeros(1, num\_images);

entropy\_scores = zeros(1, num\_images);

sharpness\_scores = zeros(1, num\_images);

fprintf('Processing patient: %s with %d images\n', patient\_name, num\_images);

% Load and process each MRI image

for j = 1:num\_images

img\_path = fullfile(image\_files(j).folder, image\_files(j).name);

% Read DICOM image

img = dicomread(img\_path);

% Convert to double for processing

img = double(img);

% Ensure image is 2D (extract the middle slice if needed)

if ndims(img) > 2

img = squeeze(img(:,:,round(size(img,3)/2))); % Middle slice

end

% Check for empty or invalid images

if isempty(img) || all(img(:) == 0)

fprintf('Skipping image %s (empty or invalid)\n', img\_path);

continue;

end

% Normalize intensity safely

min\_val = min(img(:));

max\_val = max(img(:));

if max\_val > min\_val

img = 255 \* (img - min\_val) / (max\_val - min\_val);

else

img = zeros(size(img)); % Set to black if all pixel values are the same

end

% Store image for saving later

image\_data{j} = uint8(img); % Convert back to uint8 for saving

% Compute features

intensity\_scores(j) = mean(img(:)); % Mean intensity

entropy\_scores(j) = entropy(img); % Entropy

sharpness\_scores(j) = compute\_sharpness(img); % Sharpness

end

% Normalize feature scores to [0, 1]

intensity\_scores = normalize\_scores(intensity\_scores);

entropy\_scores = normalize\_scores(entropy\_scores);

sharpness\_scores = normalize\_scores(sharpness\_scores);

% Combine scores with equal weights

combined\_scores = 0.4 \* intensity\_scores + 0.3 \* entropy\_scores + 0.3 \* sharpness\_scores;

% Ensure no NaN values in combined scores

combined\_scores(isnan(combined\_scores)) = 0;

% Select the top 300 images with the highest combined scores

[~, sorted\_indices] = sort(combined\_scores, 'descend');

best\_indices = sorted\_indices(1:min(300, num\_images));

% Save the selected images to the output folder

patient\_output\_folder = fullfile(output\_folder, patient\_name);

if ~exist(patient\_output\_folder, 'dir')

mkdir(patient\_output\_folder);

end

for k = 1:length(best\_indices)

best\_image = image\_data{best\_indices(k)};

output\_filename = fullfile(patient\_output\_folder, sprintf('Selected\_%03d.dcm', k));

dicomwrite(best\_image, output\_filename); % Save in DICOM format

end

fprintf('Saved best 300 images for %s\n', patient\_name);

end

disp('Image selection complete.');

%% Helper Functions

function norm\_scores = normalize\_scores(scores)

% Normalize scores to the range [0, 1], handling zero division

min\_val = min(scores);

max\_val = max(scores);

if max\_val > min\_val

norm\_scores = (scores - min\_val) / (max\_val - min\_val);

else

norm\_scores = zeros(size(scores)); % Avoid NaN issues

end

end

function score = compute\_sharpness(img)

% Compute image sharpness using the variance of the gradient magnitude

[Gx, Gy] = imgradientxy(double(img)); % Compute gradients in X and Y directions

gradient\_magnitude = sqrt(Gx.^2 + Gy.^2); % Gradient magnitude

score = var(gradient\_magnitude(:)); % Variance as a sharpness measure

end

**Code 2 Feature extraction of 300 best images of healthy patients**

% Define the path to the folder with selected MRI images

main\_folder = 'E:\IRS New Data\Updated Healthy Patients'; % Update with correct path

% Get all subfolders (patients) inside the main folder

patient\_folders = dir(fullfile(main\_folder, '\*'));

patient\_folders = patient\_folders([patient\_folders.isdir] & ~ismember({patient\_folders.name}, {'.', '..'}));

% Initialize storage for all features

all\_features = [];

patient\_names = {}; % Store patient names

fprintf('Starting feature extraction process...\n');

% Loop through each patient folder (subfolder)

for i = 1:length(patient\_folders)

patient\_name = patient\_folders(i).name;

patient\_folder = fullfile(main\_folder, patient\_name);

fprintf('Processing patient: %s\n', patient\_name);

% Get all selected image files in the current patient folder

image\_files = dir(fullfile(patient\_folder, '\*.dcm')); % Assuming DICOM format

num\_images = length(image\_files);

if num\_images < 1

fprintf('No images found for patient: %s. Skipping...\n', patient\_name);

continue; % Skip empty folders

end

% Loop through each image of the current patient

for j = 1:num\_images

img\_path = fullfile(patient\_folder, image\_files(j).name);

fprintf('Processing image %d of %d for patient: %s\n', j, num\_images, patient\_name);

% Read DICOM image

img = dicomread(img\_path);

img = double(img);

% Ensure image is 2D (extract the middle slice if needed)

if ndims(img) > 2

img = squeeze(img(:,:,round(size(img,3)/2))); % Middle slice

end

% Ensure the image is normalized

min\_val = min(img(:));

max\_val = max(img(:));

if max\_val > min\_val

img = 255 \* (img - min\_val) / (max\_val - min\_val);

else

img = zeros(size(img)); % Avoid division errors

end

% Feature Extraction

% --- 1. Intensity-based Features ---

mean\_intensity = mean(img(:));

std\_intensity = std(img(:));

entropy\_val = entropy(img);

% --- 2. Texture Features (GLCM) ---

glcm = graycomatrix(uint8(img), 'Offset', [0 1]);

stats = graycoprops(glcm, {'contrast', 'correlation', 'energy', 'homogeneity'});

contrast = stats.Contrast;

correlation = stats.Correlation;

energy = stats.Energy;

homogeneity = stats.Homogeneity;

% --- 3. Morphological Features ---

BW = imbinarize(uint8(img));

stats\_morph = regionprops(BW, 'Area', 'Perimeter', 'Solidity');

% Handle cases where regionprops might return empty

if isempty(stats\_morph)

area = 0;

perimeter = 0;

solidity = 0;

else

area = stats\_morph.Area;

perimeter = stats\_morph.Perimeter;

solidity = stats\_morph.Solidity;

end

% Combine all features into a single vector

combined\_features = [mean\_intensity, std\_intensity, entropy\_val, ...

contrast, correlation, energy, homogeneity, ...

area, perimeter, solidity];

% Ensure feature vector consistency before concatenation

if isempty(all\_features)

all\_features = combined\_features; % Initialize with first image's features

else

if length(combined\_features) == size(all\_features, 2)

all\_features = [all\_features; combined\_features];

else

disp(['Feature vector size mismatch for ' image\_files(j).name]);

end

end

% Store patient name

patient\_names = [patient\_names; {patient\_name}];

end

fprintf('Completed processing for patient: %s\n', patient\_name);

end

% Convert to table with appropriate column names

T = table(patient\_names, all\_features(:,1), all\_features(:,2), all\_features(:,3), ...

all\_features(:,4), all\_features(:,5), all\_features(:,6), all\_features(:,7), ...

all\_features(:,8), all\_features(:,9), all\_features(:,10), ...

'VariableNames', {'Patient Name', 'Mean Intensity', 'Std Intensity', 'Entropy', ...

'Contrast', 'Correlation', 'Energy', 'Homogeneity', ...

'Area', 'Perimeter', 'Solidity'});

% Save the extracted features into an Excel file

writetable(T, 'Healthy\_patients\_MRI\_Features\_Updated.xlsx');

disp('Feature extraction complete. Data saved to Healthy\_patients\_MRI\_Features\_Updated.xlsx');

**Code 3 selecting Best 300 images in healthy patients**

% Define the main folder containing subfolders for each patient

main\_folder = 'E:\IRS New Data\UnHealthy Patients'; % Update with your path

output\_folder = 'E:\IRS New Data\Updated UnHealthy Patients'; % Folder to save selected images

% Create the output directory if it doesn't exist

if ~exist(output\_folder, 'dir')

mkdir(output\_folder);

end

% Get all patient subfolders

patient\_folders = dir(fullfile(main\_folder, '\*'));

patient\_folders = patient\_folders([patient\_folders.isdir] & ~ismember({patient\_folders.name}, {'.', '..'}));

% Loop through each patient's subfolder

for i = 1:length(patient\_folders)

patient\_name = patient\_folders(i).name;

patient\_folder = fullfile(main\_folder, patient\_name);

image\_files = dir(fullfile(patient\_folder, '\*.dcm')); % DICOM format

num\_images = length(image\_files);

if num\_images < 300

fprintf('Skipping %s (less than 300 images)\n', patient\_name);

continue;

end

% Initialize storage for images and scores

image\_data = cell(1, num\_images);

intensity\_scores = zeros(1, num\_images);

entropy\_scores = zeros(1, num\_images);

sharpness\_scores = zeros(1, num\_images);

fprintf('Processing patient: %s with %d images\n', patient\_name, num\_images);

% Load and process each MRI image

for j = 1:num\_images

img\_path = fullfile(image\_files(j).folder, image\_files(j).name);

% Read DICOM image

img = dicomread(img\_path);

% Convert to double for processing

img = double(img);

% Ensure image is 2D (extract the middle slice if needed)

if ndims(img) > 2

img = squeeze(img(:,:,round(size(img,3)/2))); % Middle slice

end

% Check for empty or invalid images

if isempty(img) || all(img(:) == 0)

fprintf('Skipping image %s (empty or invalid)\n', img\_path);

continue;

end

% Normalize intensity safely

min\_val = min(img(:));

max\_val = max(img(:));

if max\_val > min\_val

img = 255 \* (img - min\_val) / (max\_val - min\_val);

else

img = zeros(size(img)); % Set to black if all pixel values are the same

end

% Store image for saving later

image\_data{j} = uint8(img); % Convert back to uint8 for saving

% Compute features

intensity\_scores(j) = mean(img(:)); % Mean intensity

entropy\_scores(j) = entropy(img); % Entropy

sharpness\_scores(j) = compute\_sharpness(img); % Sharpness

end

% Normalize feature scores to [0, 1]

intensity\_scores = normalize\_scores(intensity\_scores);

entropy\_scores = normalize\_scores(entropy\_scores);

sharpness\_scores = normalize\_scores(sharpness\_scores);

% Combine scores with equal weights

combined\_scores = 0.4 \* intensity\_scores + 0.3 \* entropy\_scores + 0.3 \* sharpness\_scores;

% Ensure no NaN values in combined scores

combined\_scores(isnan(combined\_scores)) = 0;

% Select the top 300 images with the highest combined scores

[~, sorted\_indices] = sort(combined\_scores, 'descend');

best\_indices = sorted\_indices(1:min(300, num\_images));

% Save the selected images to the output folder

patient\_output\_folder = fullfile(output\_folder, patient\_name);

if ~exist(patient\_output\_folder, 'dir')

mkdir(patient\_output\_folder);

end

for k = 1:length(best\_indices)

best\_image = image\_data{best\_indices(k)};

output\_filename = fullfile(patient\_output\_folder, sprintf('Selected\_%03d.dcm', k));

dicomwrite(best\_image, output\_filename); % Save in DICOM format

end

fprintf('Saved best 300 images for %s\n', patient\_name);

end

disp('Image selection complete.');

%% Helper Functions

function norm\_scores = normalize\_scores(scores)

% Normalize scores to the range [0, 1], handling zero division

min\_val = min(scores);

max\_val = max(scores);

if max\_val > min\_val

norm\_scores = (scores - min\_val) / (max\_val - min\_val);

else

norm\_scores = zeros(size(scores)); % Avoid NaN issues

end

end

function score = compute\_sharpness(img)

% Compute image sharpness using the variance of the gradient magnitude

[Gx, Gy] = imgradientxy(double(img)); % Compute gradients in X and Y directions

gradient\_magnitude = sqrt(Gx.^2 + Gy.^2); % Gradient magnitude

score = var(gradient\_magnitude(:)); % Variance as a sharpness measure

end

**Code 4 Feature extraction of 300 best images of healthy patients**

% Define the path to the folder with selected MRI images

main\_folder = 'E:\IRS New Data\Updated UnHealthy Patients

; % Update with correct path

% Get all subfolders (patients) inside the main folder

patient\_folders = dir(fullfile(main\_folder, '\*'));

patient\_folders = patient\_folders([patient\_folders.isdir] & ~ismember({patient\_folders.name}, {'.', '..'}));

% Initialize storage for all features

all\_features = [];

patient\_names = {}; % Store patient names

fprintf('Starting feature extraction process...\n');

% Loop through each patient folder (subfolder)

for i = 1:length(patient\_folders)

patient\_name = patient\_folders(i).name;

patient\_folder = fullfile(main\_folder, patient\_name);

fprintf('Processing patient: %s\n', patient\_name);

% Get all selected image files in the current patient folder

image\_files = dir(fullfile(patient\_folder, '\*.dcm')); % Assuming DICOM format

num\_images = length(image\_files);

if num\_images < 1

fprintf('No images found for patient: %s. Skipping...\n', patient\_name);

continue; % Skip empty folders

end

% Loop through each image of the current patient

for j = 1:num\_images

img\_path = fullfile(patient\_folder, image\_files(j).name);

fprintf('Processing image %d of %d for patient: %s\n', j, num\_images, patient\_name);

% Read DICOM image

img = dicomread(img\_path);

img = double(img);

% Ensure image is 2D (extract the middle slice if needed)

if ndims(img) > 2

img = squeeze(img(:,:,round(size(img,3)/2))); % Middle slice

end

% Ensure the image is normalized

min\_val = min(img(:));

max\_val = max(img(:));

if max\_val > min\_val

img = 255 \* (img - min\_val) / (max\_val - min\_val);

else

img = zeros(size(img)); % Avoid division errors

end

% Feature Extraction

% --- 1. Intensity-based Features ---

mean\_intensity = mean(img(:));

std\_intensity = std(img(:));

entropy\_val = entropy(img);

% --- 2. Texture Features (GLCM) ---

glcm = graycomatrix(uint8(img), 'Offset', [0 1]);

stats = graycoprops(glcm, {'contrast', 'correlation', 'energy', 'homogeneity'});

contrast = stats.Contrast;

correlation = stats.Correlation;

energy = stats.Energy;

homogeneity = stats.Homogeneity;

% --- 3. Morphological Features ---

BW = imbinarize(uint8(img));

stats\_morph = regionprops(BW, 'Area', 'Perimeter', 'Solidity');

% Handle cases where regionprops might return empty

if isempty(stats\_morph)

area = 0;

perimeter = 0;

solidity = 0;

else

area = stats\_morph.Area;

perimeter = stats\_morph.Perimeter;

solidity = stats\_morph.Solidity;

end

% Combine all features into a single vector

combined\_features = [mean\_intensity, std\_intensity, entropy\_val, ...

contrast, correlation, energy, homogeneity, ...

area, perimeter, solidity];

% Ensure feature vector consistency before concatenation

if isempty(all\_features)

all\_features = combined\_features; % Initialize with first image's features

else

if length(combined\_features) == size(all\_features, 2)

all\_features = [all\_features; combined\_features];

else

disp(['Feature vector size mismatch for ' image\_files(j).name]);

end

end

% Store patient name

patient\_names = [patient\_names; {patient\_name}];

end

fprintf('Completed processing for patient: %s\n', patient\_name);

end

% Convert to table with appropriate column names

T = table(patient\_names, all\_features(:,1), all\_features(:,2), all\_features(:,3), ...

all\_features(:,4), all\_features(:,5), all\_features(:,6), all\_features(:,7), ...

all\_features(:,8), all\_features(:,9), all\_features(:,10), ...

'VariableNames', {'Patient Name', 'Mean Intensity', 'Std Intensity', 'Entropy', ...

'Contrast', 'Correlation', 'Energy', 'Homogeneity', ...

'Area', 'Perimeter', 'Solidity'});

% Save the extracted features into an Excel file

writetable(T, 'UnHealthy\_patients\_MRI\_Features\_Updated.xlsx');

disp('Feature extraction complete. Data saved to UnHealthy\_patients\_MRI\_Features\_Updated.xlsx');

**Code 5 risk levels of 300 best images of healthy patients**

% Step 1: Load the features from the Excel file while preserving column headers

filename = "E:\Data files\Healthy\_patients\_MRI\_Features\_Updated.xlsx"; % Specify the correct path to your Excel file

opts = detectImportOptions(filename); % Detect import options for the Excel file

opts.VariableNamingRule = 'preserve'; % Preserve original column names

% Read the data using the import options

data = readtable(filename, opts);

% Check the column names

disp(data.Properties.VariableNames);

% Assuming the first column is Patient IDs and the rest are features

features = data{:, 2:end}; % Extract the feature matrix (excluding Patient IDs)

% Normalize the features for consistency

features\_normalized = normalize(features);

% Step 2: Apply K-Means clustering to divide patients into 3 risk groups

num\_clusters = 3; % Define the number of clusters (Low, Medium, High risk)

[cluster\_idx, ~] = kmeans(features\_normalized, num\_clusters);

% Step 3: Assign risk labels to clusters

risk\_labels = cell(size(cluster\_idx));

for i = 1:numel(cluster\_idx)

if cluster\_idx(i) == 1

risk\_labels{i} = 'Low'; % Low risk

elseif cluster\_idx(i) == 2

risk\_labels{i} = 'Medium'; % Medium risk

else

risk\_labels{i} = 'High'; % High risk

end

end

% Step 4: Visualize the clusters (if 2D or 3D data)

figure;

gscatter(features\_normalized(:,1), features\_normalized(:,2), cluster\_idx, 'rgb', 'o', 8);

title('K-Means Clustering of Patients');

xlabel('Feature 1');

ylabel('Feature 2');

legend('Cluster 1 (Low)', 'Cluster 2 (Medium)', 'Cluster 3 (High)');

% Step 5: Save the results to an Excel file

patient\_ids = data{:, 1}; % Assuming Patient IDs are in the first column

results\_table = table(patient\_ids, cluster\_idx, risk\_labels);

% Save to Excel file

output\_filename = 'Risk\_Level\_Prediction\_KMeans\_31Patients.xlsx'; % Output file name

writetable(results\_table, output\_filename);

disp('Results saved to Excel file.');

**Code 6 risk levels of 300 best images of healthy patients**

% Assuming you have already applied K-means clustering and obtained the cluster\_idx for each image

% 'patient\_ids' is a list of patient IDs corresponding to each image

% 'cluster\_idx' is the cluster label assigned to each image (Low, Medium, High)

% 'features' contains the extracted features

% Step 1: Prepare the data for processing

% Here, we assume you have 'patient\_ids' and 'cluster\_idx' from the K-means clustering

% For demonstration, let's assume you have these variables:

% patient\_ids: List of patient IDs corresponding to each image (same size as cluster\_idx)

% cluster\_idx: K-means cluster labels (1=Low, 2=Medium, 3=High)

% Example:

% patient\_ids = {'Patient1', 'Patient1', 'Patient2', 'Patient2', ...};

% cluster\_idx = [1, 1, 2, 3, ...]; % Cluster labels for each image

% Step 2: Aggregate the labels for each patient

unique\_patients = unique(patient\_ids); % Get list of unique patient IDs

patient\_risk = cell(length(unique\_patients), 2); % Store patient ID and their risk label

for i = 1:length(unique\_patients)

% Find all the images corresponding to the current patient

patient\_images = find(strcmp(patient\_ids, unique\_patients{i}));

% Get the cluster labels for the current patient

patient\_labels = cluster\_idx(patient\_images);

% Calculate the most common cluster label (majority vote)

[mode\_label, ~] = mode(patient\_labels);

% Assign the risk level based on the mode label

if mode\_label == 1

patient\_risk{i, 1} = unique\_patients{i}; % Patient ID

patient\_risk{i, 2} = 'Low'; % Low risk

elseif mode\_label == 2

patient\_risk{i, 1} = unique\_patients{i}; % Patient ID

patient\_risk{i, 2} = 'Medium'; % Medium risk

else

patient\_risk{i, 1} = unique\_patients{i}; % Patient ID

patient\_risk{i, 2} = 'High'; % High risk

end

end

% Step 3: Save the patient risk labels into an Excel file

patient\_risk\_table = cell2table(patient\_risk, 'VariableNames', {'PatientID', 'RiskLevel'});

% Save the table to an Excel file

output\_filename = 'Patient\_Risk\_Levels.xlsx'; % Output file name

writetable(patient\_risk\_table, output\_filename);

disp('Patient risk levels saved to ExcelLL file.');