```
% ========================================================================
% Practical 1: 2D Convolution Analysis
% ========================================================================
%
% GROUP NUMBER: 24
%
% MEMBERS:
%   - Member 1 Nyakallo Peete, PTXNYA001
%   - Member 2 Samson Okuthe, OKTSAM001

%% ========================================================================
%  PART 3: Testing and Analysis (MAIN FUNCTION)
%  ========================================================================
function run_analysis()
    clc; close all;
    fprintf('Starting Benchmarking Analysis for Group 24...\n\n');

    % TODO 1: Load all the sample images from the 'sample_images' folder
    img_names = {'image_128x128.png', 'image_256x256.png', 'image_512x512.png',
'image_1024x1024.png', 'image_2048x2048.png'};
    num_images = length(img_names);

    % Pre-allocate cell array for images
    imgs = cell(1, num_images);
    % Arrays to store results for our table/graphs
    pixels_count = zeros(1, num_images);

    fprintf('Loading Images...\n');
    for i = 1:num_images
        path = fullfile('sample_images', img_names{i});

        % Read image
        temp_img = imread(path);

        % Convert to grayscale if it is RGB (3 channels)
        if size(temp_img, 3) == 3
            temp_img = rgb2gray(temp_img);
        end

        % Convert to double to prevent math overflow during convolution
        imgs{i} = double(temp_img);

        % Store number of pixels for plotting X-axis later
        [r, c] = size(imgs{i});
        pixels_count(i) = r * c;
        fprintf('  Loaded: %s (%dx%d)\n', img_names{i}, r, c);
    end
    fprintf('\n');

    % TODO 2: Define edge detection kernels (Sobel kernel)
    % horizontal edge detector (Matches PDF Eq 1)
    Gx = [-1,  0,  1;
          -2,  0,  2;
          -1,  0,  1];

    % vertical edge detector (Matches PDF Eq 2)
    Gy = [ 1,  2,  1;
           0,  0,  0;
          -1, -2, -1];

    % TODO 3: Testing and Benchmarking
```

```
    manual_times = zeros(1, num_images);
    builtin_times = zeros(1, num_images);
    speedups = zeros(1, num_images);

    % Number of runs to average execution time for better accuracy
    num_runs = 3;

    for i = 1:num_images
        fprintf('Processing Image %d/%d (%s)...\n', i, num_images, img_names{i});
        img = imgs{i};

        % a. Measure execution time of my_conv2
        tic;
        for r = 1:num_runs
            man_Gx = my_conv2(img, Gx);
            man_Gy = my_conv2(img, Gy);
        end
        manual_times(i) = toc() / num_runs;

        % b. Measure execution time of inbuilt_conv2
        tic;
        for r = 1:num_runs
            built_Gx = inbuilt_conv2(img, Gx);
            built_Gy = inbuilt_conv2(img, Gy);
        end
        builtin_times(i) = toc() / num_runs;

        % c. Compute speedup ratio
        speedups(i) = manual_times(i) / builtin_times(i);

        % d. Verify output correctness
        % Check max absolute difference. Should be nearly 0.
        diff_x = max(abs(man_Gx(:) - built_Gx(:)));
        diff_y = max(abs(man_Gy(:) - built_Gy(:)));

        if diff_x > 1e-6 || diff_y > 1e-6
            warning('VERIFICATION FAILED: Manual does not match Built-in!');
        else
            fprintf('  -> Verification: PASSED! Outputs match.\n');
        end

        % e. Store and display results
        fprintf('  -> Manual Time:  %.4f seconds\n', manual_times(i));
        fprintf('  -> Built-in Time: %.4f seconds\n', builtin_times(i));
        fprintf('  -> Speedup Ratio: %.2fx\n\n', speedups(i));
    end

    % Display Summary Table
    disp('=== FINAL BENCHMARKING RESULTS ===');
    T = table(img_names', pixels_count', manual_times', builtin_times', speedups', ...
        'VariableNames', {'Image_Name', 'Total_Pixels', 'Manual_Time_s', 'BuiltIn_Time_s',
'Speedup'});
    disp(T);

    % f. Plot and compare results
    figure('Name', 'Performance Benchmarking', 'NumberTitle', 'off');

    % Plot 1: Execution Time vs Number of Pixels
    subplot(1, 2, 1);
    plot(pixels_count, manual_times, '-ro', 'LineWidth', 2, 'MarkerSize', 6);
    hold on;
```

```
    plot(pixels_count, builtin_times, '-bs', 'LineWidth', 2, 'MarkerSize', 6);
    grid on;
    title('Execution Time vs Image Size');
    xlabel('Image Size (Total Pixels)');
    ylabel('Execution Time (Seconds)');
    legend('Manual (my\_conv2)', 'Built-in (conv2)', 'Location', 'northwest');

    % Plot 2: Speedup vs Number of Pixels
    subplot(1, 2, 2);
    plot(pixels_count, speedups, '-m^', 'LineWidth', 2, 'MarkerSize', 6);
    grid on;
    title('Speedup Ratio vs Image Size');
    xlabel('Image Size (Total Pixels)');
    ylabel('Speedup Factor (Manual / Built-in)');

    sgtitle('Benchmarking: Manual vs Built-in Convolution');
end

%% ========================================================================
%  PART 1: Manual 2D Convolution Implementation (HELPER FUNCTION)
%  ========================================================================
function output = my_conv2(img, kernel)
    % Get dimensions of image and kernel
    [img_rows, img_cols] = size(img);
    [k_rows, k_cols] = size(kernel);

    % True mathematical convolution rotates the kernel by 180 degrees.
    % We MUST do this so our manual output perfectly matches MATLAB's conv2!
    kernel = kernel(end:-1:1, end:-1:1);

    % Calculate padding sizes to mimic the 'same' parameter behavior
    pad_r = floor(k_rows / 2);
    pad_c = floor(k_cols / 2);

    % Manually pad the image with zeros around the borders
    padded_img = zeros(img_rows + 2*pad_r, img_cols + 2*pad_c);
    padded_img(pad_r+1 : end-pad_r, pad_c+1 : end-pad_c) = img;

    % Pre-allocate output image matrix
    output = zeros(img_rows, img_cols);

    % Perform 2D Convolution using strictly FOR loops
    for i = 1:img_rows
        for j = 1:img_cols
            % Compute the sum of element-wise multiplication in the neighborhood
            sum_val = 0;
            for ki = 1:k_rows
                for kj = 1:k_cols
                    pixel_val = padded_img(i + ki - 1, j + kj - 1);
                    weight = kernel(ki, kj);
                    sum_val = sum_val + (pixel_val * weight);
                end
            end
            % Assign computed value to output pixel
            output(i, j) = sum_val;
        end
    end
end

%% ========================================================================
%  PART 2: Built-in 2D Convolution Implementation (HELPER FUNCTION)
```

```
%  ========================================================================
function output = inbuilt_conv2(img, kernel)
    % Wrapper for built-in function.
    % 'same' ensures output size matches input size, like our manual version.
    output = conv2(img, kernel, 'same');
end
```