```
% 1.1.1
function F = dip_fft2(I)
    % Check if input is complex, if not convert it to complex
    if ~isreal(I)
        I = complex(I);
    end

    % Ensure that input dimensions are powers of 2
    [M, N] = size(I);
    desiredSize = 2^nextpow2(max(M, N));
    if M ~= desiredSize || N ~= desiredSize
        I = padarray(I, [desiredSize-M, desiredSize-N], 0, 'post');
    end

    % Compute 1D FFT along columns
    F_col = zeros(size(I));
    for m = 1:size(I, 1)
        F_col(m, :) = dip_fft(I(m, :));
    end

    % Compute 1D FFT along rows
    F = zeros(size(I));
    for n = 1:size(I, 2)
        F(:, n) = dip_fft(F_col(:, n).');
    end
end

function Y = dip_fft(X)
    % Cooley-Tukey FFT algorithm
    N = length(X);
    if N <= 1
        Y = X;
    else
        X_even = dip_fft(X(1:2:end));
        X_odd = dip_fft(X(2:2:end));
        factor = exp(-2i * pi / N * (0:N/2-1));
        Y = [X_even + factor .* X_odd, X_even - factor .* X_odd];
    end
end

function I = dip_ifft2(FFT)
    % Check if input is complex, if not convert it to complex
    if ~isreal(FFT)
        FFT = complex(FFT);
    end

    % Compute 1D IFFT along columns
    I_col = zeros(size(FFT));
    for m = 1:size(FFT, 1)
        I_col(m, :) = dip_ifft(FFT(m, :));
    end

    % Compute 1D IFFT along rows
    I = zeros(size(FFT));
    for n = 1:size(FFT, 2)
        I(:, n) = dip_ifft(I_col(:, n).');
    end
end
```

```
function Y = dip_ifft(X)
    % Cooley-Tukey IFFT algorithm
    N = length(X);
    if N <= 1
        Y = X;
    else
        X_even = dip_ifft(X(1:2:end));
        X_odd = dip_ifft(X(2:2:end));
        factor = exp(2i * pi / N * (0:N/2-1));
        Y = [X_even + factor .* X_odd, X_even - factor .* X_odd];
    end
end
```