MATLAB Project 2: Image Processing

Due Monday, Oct 6, 2025

Kasper Hong

jinyikah

Collaborators: None

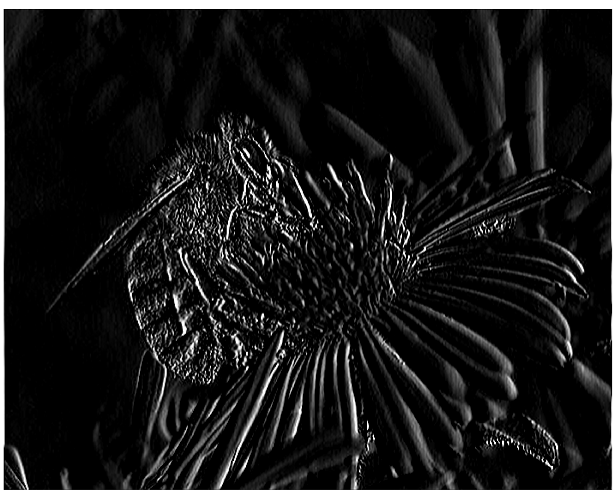
```
% Code to import image
image = double(imread("Bee.png"));
imshow(uint8(image));
```

```
% Code for #2
% Uncomment and complete the lines below once you've written your
% convolution function

kernel = ones(7,7) / 49;   % 7x7 averaging kernel
newImage = convolution(image, kernel);
imshow(uint8(newImage));
```

```
% Code for #3

% Using kernel for part (a)
kernel = ones(15,15) / 225;
newImage = convolution(image, kernel);
imshow(uint8(newImage));
```

```
% Using kernel for part (b)
kernel = [0.5];
newImage = convolution(image, kernel);
imshow(uint8(newImage));
```

```
% Using kernel for part (c)
kernel = [1 0 -1; 2 0 -2; 1 0 -1];
newImage = convolution(image, kernel);
imshow(uint8(newImage));
```



3. [Written response for #3]

(a) Strong blur since large averaging window smooths detail.

(b) Scales pixel intensity by 0.5 → darkens image.

(c) Edge detection since it subtracts left vs right neighbors;

uniform regions ≈ 0, edges pop out.

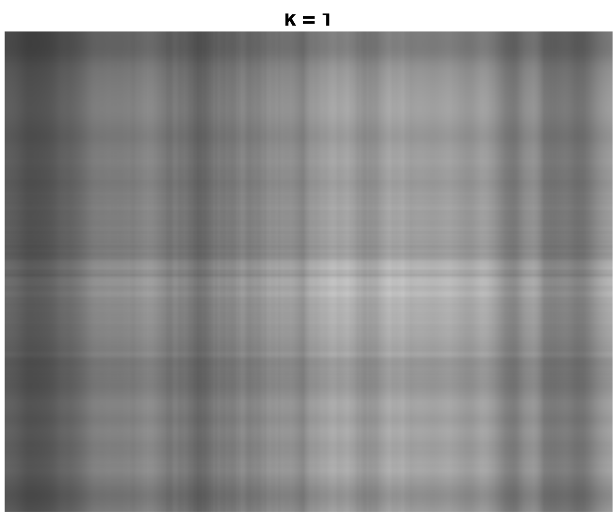
4. [Written response for #4]

Padding ensures submatrices exist for border pixels.

(k–1)/2 padding is correct so every pixel can be the center of a k×k window.

```
% Test case for #5
test = [2 4 -1 3; 1 1 1 1; -5 0 2 4];

% Test case for #6
approx1 = approximation(test,1);
approx2 = approximation(test,2);
approx3 = approximation(test,3);
```

```
% Code for #7
for k = [1,10,20,50,100,480]
    approxImg = approximation(image,k);
    figure; imshow(uint8(approxImg));
    title(['k = ' num2str(k)]);
end
```



k = 10

k = 20

k = 50

k = 100

k = 480

```
% Code for #8
% Complete the code below to extract the parts of s, U, and V necessary to
% get 20 terms of the approximation. The whos function will display the
% size of each variable, which you can use to find the compression ratio.
% Then do it all again for k = 50.

uncompressedImage = uint8(image);
[s,U,V] = singularValue(image);
k = 20;
spart = single(s(1:k));
Upart = single(U(:,1:k));
Vpart = single(V(:,1:k));
whos spart Upart Vpart uncompressedImage
```

```
  Name                      Size              Bytes  Class     Attributes

  Upart                   480x20              38400  single
  Vpart                   608x20              48640  single
  spart                    20x1                  80  single
  uncompressedImage       480x608            291840  uint8
```

```
k = 50;
spart = single(s(1:k));
Upart = single(U(:,1:k));
Vpart = single(V(:,1:k));
whos spart Upart Vpart uncompressedImage
```

```
  Name                      Size              Bytes  Class     Attributes

  Upart                   480x50              96000  single
  Vpart                   608x50             121600  single
  spart                    50x1                 200  single
  uncompressedImage       480x608            291840  uint8
```

8. [Written response for #8]

Compression ratio = (size of uncompressed image) / (size of spart+Upart+Vpart).

k=20 gives high compression but lower quality; k=50 is larger but visually better.

Function Definitions

addPadding (for #1)

```
function padded = addPadding(A,p)
    [m,n] = size(A);                % get dimensions of A
    rows = 128 * ones(p,n);         % p rows of gray (128)
    A = [rows; A; rows];            % add rows to top and bottom
    cols = 128 * ones(m+2*p,p);     % p columns of gray
    padded = [cols A cols];         % add columns left and right
end
```

convolution (for #2)

```
function result = convolution(A, kernel)
    k = size(kernel,1);             % assume square kernel
    [m,n] = size(A);
    pad = (k-1)/2;
    Apadded = addPadding(A,pad);    % pad image
    result = zeros(m,n);            % initialize result

    % Main loop to compute the result matrix
    for i = 1:m
        for j = 1:n
            subA = Apadded(i:i+k-1, j:j+k-1);   % kxk neighborhood
```

```
            result(i,j) = sum(sum(kernel .* subA)); % elementwise multiply
then sum
        end
    end
end
```

singularValue (for #5)

```
function [s,U,V] = singularValue(A)
    [U,S,V] = svd(A);             % MATLAB SVD
    s = diag(S);                  % singular values as vector
end
```

approximation (for #6)

```
function approx = approximation(A,k)
    [s,U,V] = singularValue(A);   % data from SVD of A
    [m,n] = size(A);
    approx = zeros(m,n);
    for i = 1:k
        approx = approx + s(i) * U(:,i) * V(:,i)';  % rank-k reconstruction
    end
end
```