# CMPE 362

# SIGNAL PROCESSING

# HOMEWORK 3 - REPORT

**Emre KOÇ**
2016400327

27.06.2020

## Question 1

where img, the input image is a two-dimensional matrix of grayscale pixel values between 0 and 255. Blurring is to be carried out by averaging the pixel values in the vicinity of every pixel. Specifically, the output pixel value is the mean of the pixels in a square submatrix of size 2w+1 where the given pixel sits in the center. For example, if w is 1, then we use a 3x3 matrix, that is, we average all the neighboring pixels of the given pixel and itself. Only use valid pixels when portions of the blurring matrix fall outside the image. For example, the blurred value corresponding to w = 1 at index (1,1) would be the mean of of elements (1,1), (1, 2), (2,1) and (2, 2). Both input img and output output are of type uint8.

```
img = imread("jokerimage.png");
img = uint8(img);
hFig = figure;
hFig.WindowState = 'maximized';

% generate blur matrix using filter method
kernel = [1 1 1;1 1 1;1 1 1];
blur = filter(img, kernel, 9);
subplot(2, 2, 1);
imshow(blur);
title('Blurred', 'FontSize', 20);

% generate sharpenned matrix using filter method
kernel = [0 -1 0;-1 5 -1;0 -1 0];
sharpen = filter(blur, kernel, 1);
subplot(2, 2, 2);
imshow(sharpen);
title('Sharpen', 'FontSize', 20);

% generate matrix for edges using filter method
kernel = [-1 -1 -1;-1 8 -1;-1 -1 -1];
edges = filter(img, kernel, -1);
subplot(2, 2, 3);
```

```
imshow(edges);
title('Edges', 'FontSize', 20);

% generate embossed matrix using filter method
kernel = [-2 -1 0;-1 1 1;0 1 2];
embossed = filter(img, kernel, 1);
subplot(2, 2, 4);
imshow(embossed);
title('Embossed', 'FontSize', 20);

function output = filter(img, kernel, divisor)
   [rows, columns] = size(img);
   output = img;
   kernel_dim = size(kernel);
   % loop all colums and rows for 3 color
   for mat = 1 : 3
      for col = 1 : (columns/3 - kernel_dim + 1)
         for row = 1 : (rows - kernel_dim + 1)
            % rot90(kernel, 2) = flipud(fliplr(kernel))
            a = rot90(kernel, 2);
            % 3x3 part of the img
            b = img(col:col+kernel_dim - 1, row:row+kernel_dim - 1, mat);
            % calculate the central pixel's value
            value = transpose(reshape(double(b), [], 1)) * reshape(a, [], 1) / divisor;
            % assign central pixel value
            output(col+1, row+1, mat) = value;
         end
      end
   end
   output = uint8(output);
end
```

The general expression of a convolution is

$$g(x, y) = \omega * f(x, y)$$

where $g(x, y)$ is the filtered image, $f(x, y)$is the original image, $\omega$ is the filter kernel.

Depending on the element values, a kernel can cause a wide range of effects

## Convolution

Convolution is the process of adding each element of the image to its local neighbors, weighted by the kernel. This is related to a form of mathematical convolution. The matrix operation being performed—convolution—is not traditional matrix multiplication

For example, if we have two three-by-three matrices, the first a kernel, and the second an image piece, convolution is the process of flipping both the rows and columns of the kernel and multiplying locally similar entries and summing. The element at coordinates [2, 2] (that is, the central element) of the resulting image would be a weighted combination of all the entries of the image matrix, with weights given by the kernel:

$$\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}\right)[2,2] = (i\cdot 1) + (h\cdot 2) + (g\cdot 3) + (f\cdot 4) + (e\cdot 5) + (d\cdot 6) + (c\cdot 7) + (b\cdot 8) + (a\cdot 9)$$

The other entries would be similarly weighted, where we position the center of the kernel on each of the boundary points of the image, and compute a weighted sum.

The values of a given pixel in the output image are calculated by multiplying each kernel value by the corresponding input image pixel values.

Tried these kernels to achive requested images:
For blur

kernel = [1 1 1;1 1 1;1 1 1];

For sharpen

kernel = [0 -1 0;-1 5 -1;0 -1 0];

For edges

kernel = [-1 -1 -1;-1 8 -1;-1 -1 -1];

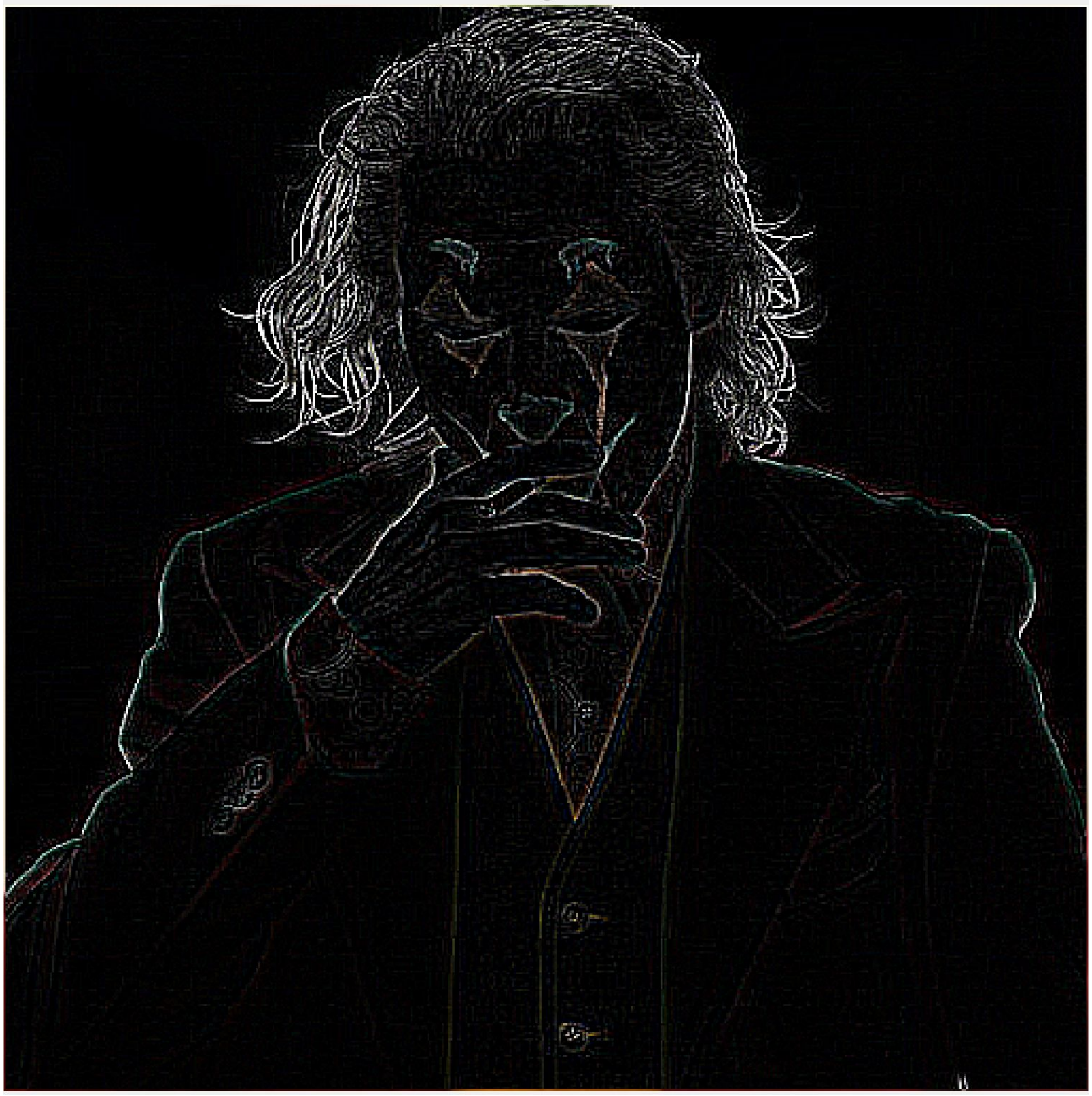
For embossed

kernel = [-2 -1 0;-1 1 1;0 1 2];

Blur

Sharpen

## Edges



**Embossed**

## Question 2

Steps of detect an image

-Read the reference image containing the object of interest.

-Read the target image containing a cluttered scene.

-Detect feature points in both images.

-Extract feature descriptors at the interest points in both images.

-Match the features using their descriptors.

-Calculate the transformation relating the matched points, while eliminating outliers. This transformation allows us to localize the object in the scene.

-Get the bounding polygon of the reference image.

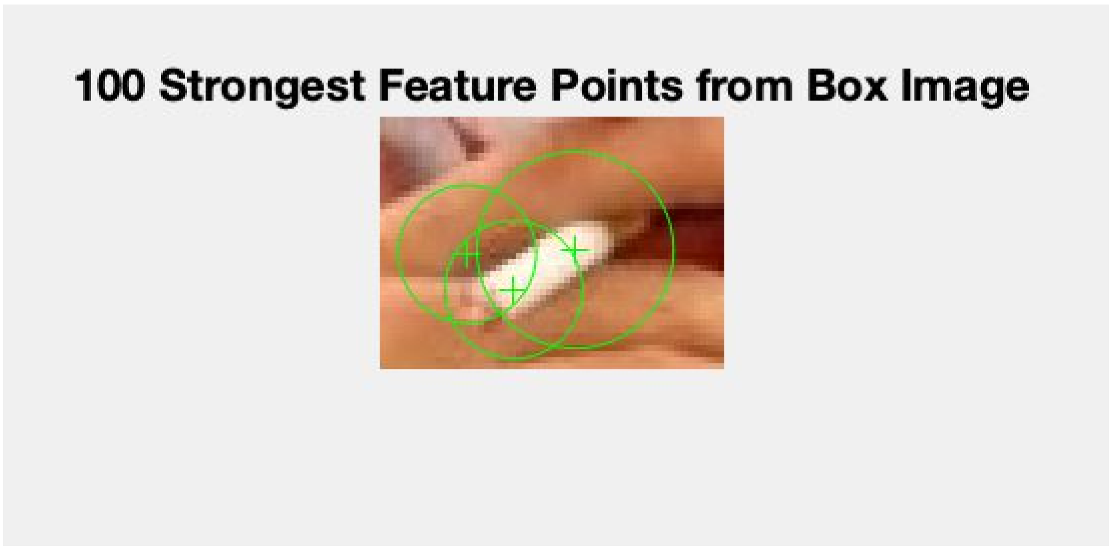
-Transform the polygon into the coordinate system of the target image. The transformed polygon indicates the location of the object in the scene.

-Replace our image with the detected part of the original image

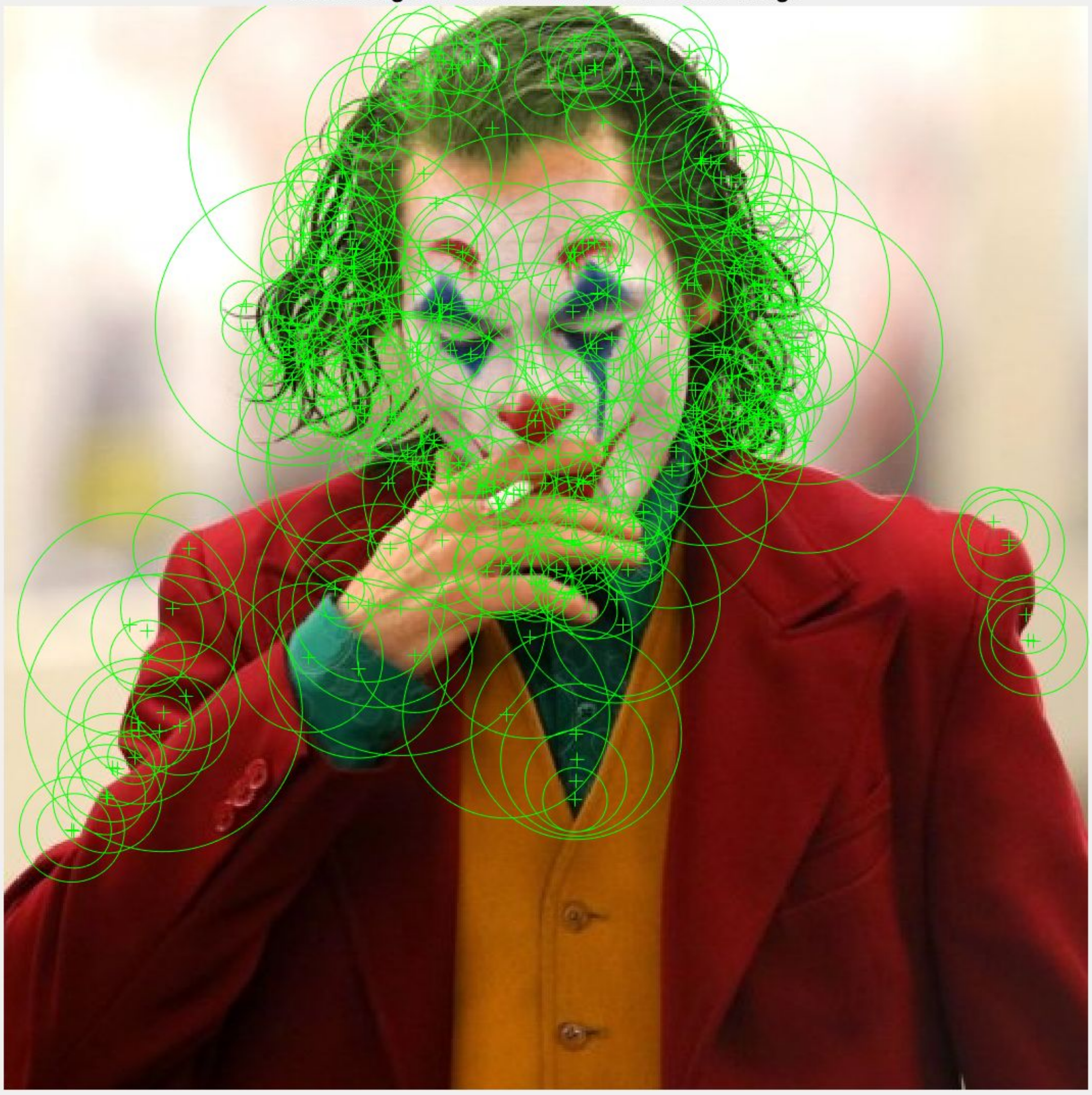
```
%Read the reference image containing the object of interest.
boxImage = imread('cigarette.jpeg');

%Read the target image containing a cluttered scene.
sceneImage = imread('jokerimage.png');
hFig = figure;
hFig.WindowState = 'maximized';
imshow(sceneImage);
title('Image of a Cluttered Scene');

%Detect feature points in both images.
boxPoints = detectSURFFeatures(rgb2gray(boxImage));
scenePoints = detectSURFFeatures(rgb2gray(sceneImage));

%Extract feature descriptors at the interest points in both images.
[boxFeatures, boxPoints] = extractFeatures(rgb2gray(boxImage),
boxPoints);
[sceneFeatures, scenePoints] = extractFeatures(rgb2gray(sceneImage),
scenePoints);
```

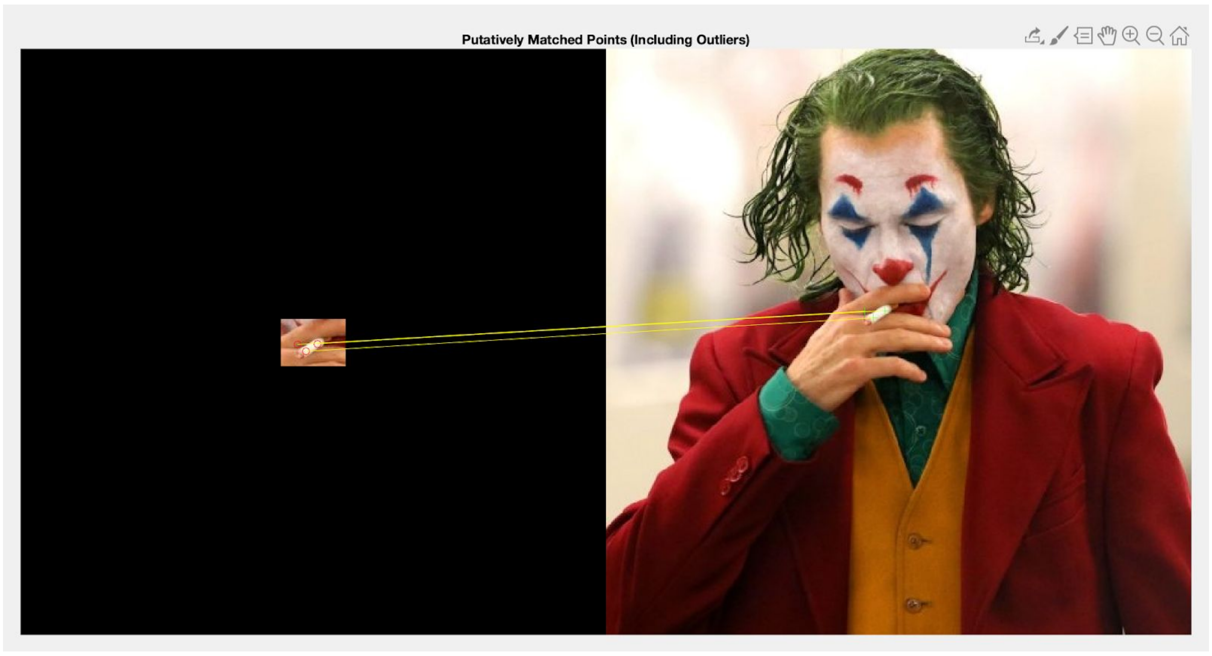
```
%Match the features using their descriptors.
boxPairs = matchFeatures(boxFeatures, sceneFeatures);

%Display putatively matched features.
matchedBoxPoints = boxPoints(boxPairs(:, 1), :);
matchedScenePoints = scenePoints(boxPairs(:, 2), :);


% estimateGeometricTransform calculates the transformation relating
the matched points, while eliminating outliers.
% This transformation allows us to localize the object in the scene.
[tform, inlierBoxPoints, inlierScenePoints] = ...
   estimateGeometricTransform(matchedBoxPoints,
matchedScenePoints, 'affine');

%Get the bounding polygon of the reference image.
boxPolygon = [1, 1;...                          % top-left
        size(boxImage, 2), 1;...                 % top-right
        size(boxImage, 2), size(boxImage, 1);... % bottom-right
        1, size(boxImage, 1);...                 % bottom-left
        1, 1];                   % top-left again to close the polygon

%Transform the polygon into the coordinate system of the target image.
%The transformed polygon indicates the location of the object in the
scene.
newBoxPolygon = transformPointsForward(tform, boxPolygon);


% image(), which permits us to pass data coordinates for lower left and
upper right.
% Note that the coordinates are for the centers of pixels.
% coordinates of left side of detected part of image
x = newBoxPolygon(1);
y = newBoxPolygon(6);
```

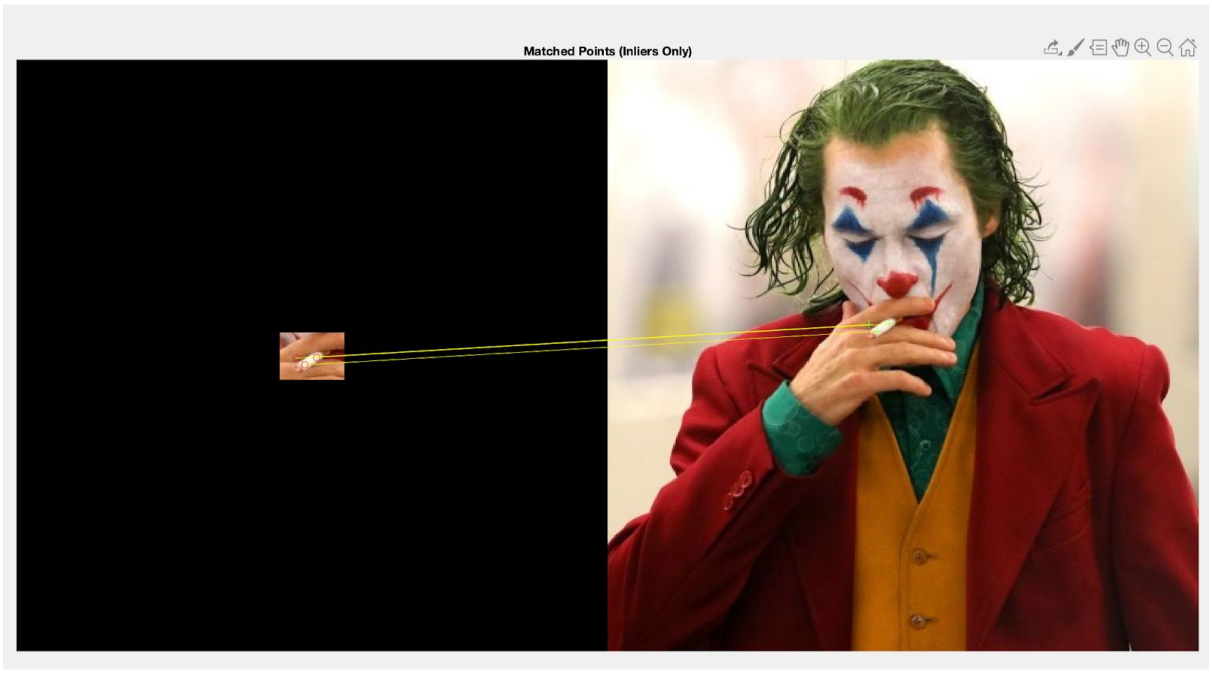
```
% replace our image with the detected part of the original image
papatya = imread('papatya.png');
hold on;
image([x x], [y y], papatya, 'CDataMapping', 'scaled');
colormap(gray(64));
```



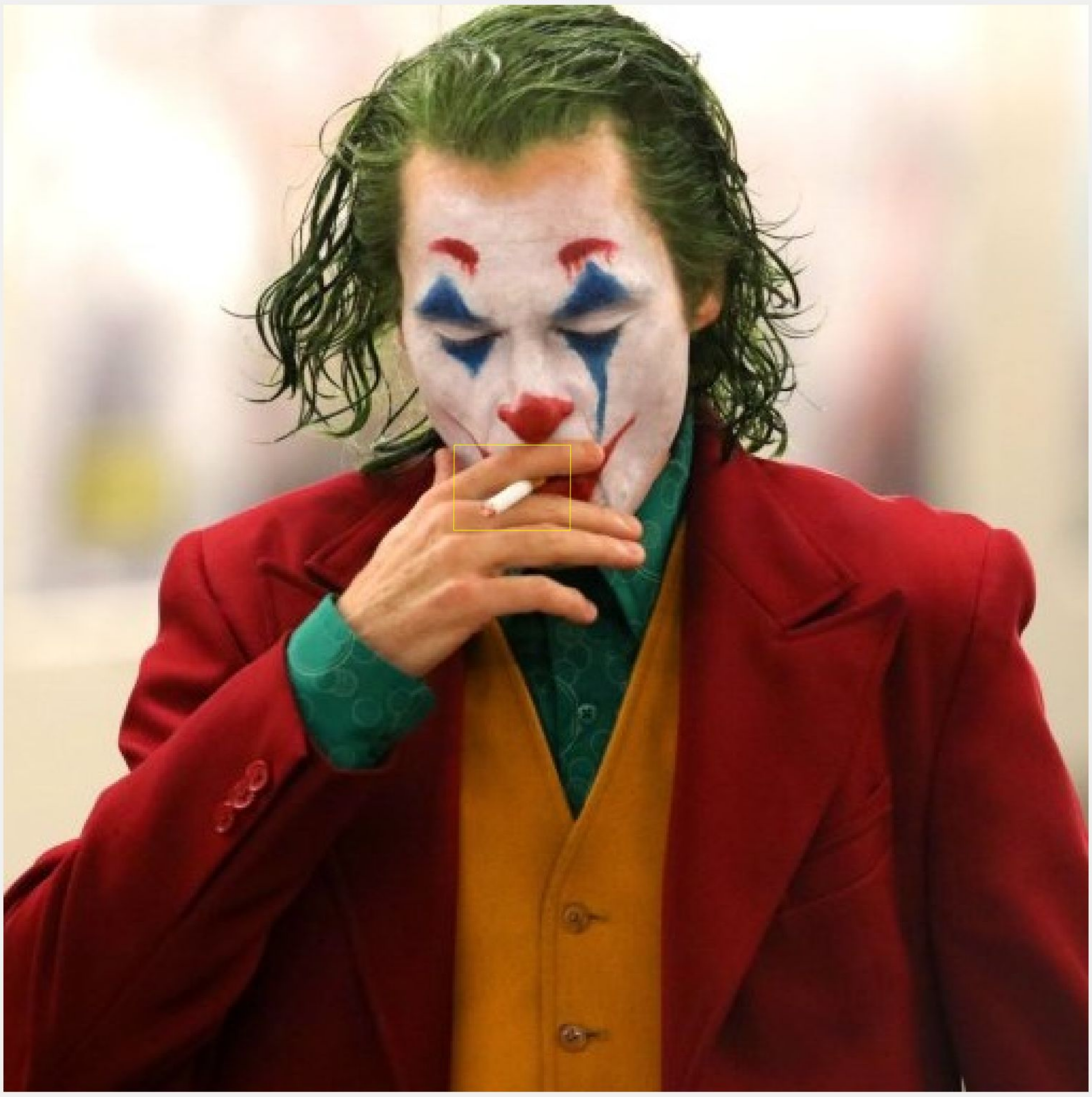
**300 Strongest Feature Points from Scene Image**

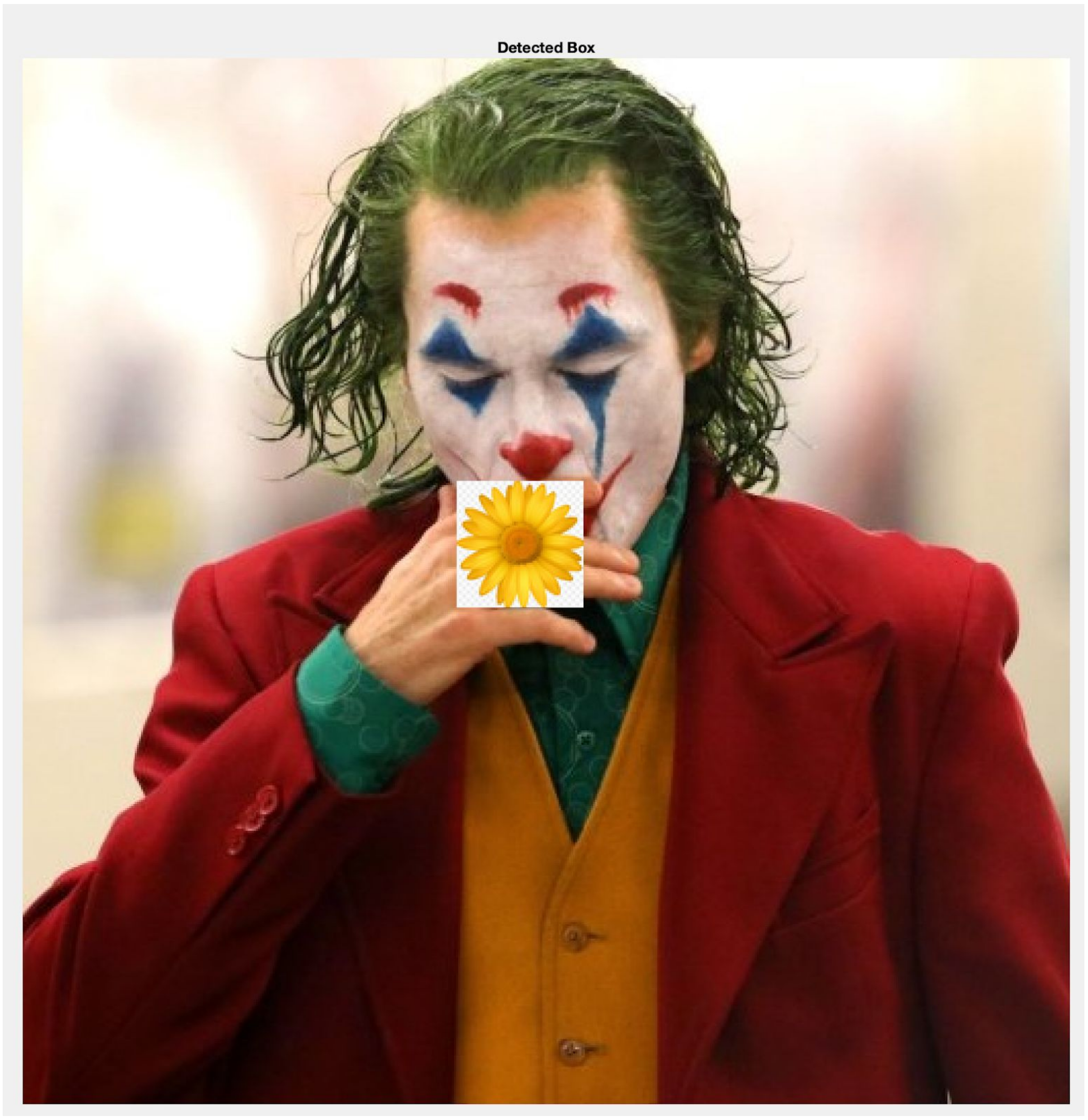



Putatively Matched Points (Including Outliers)


Matched Points (Inliers Only)

Detected Box




Detected Box