Emoji Mosaic Effect on the Selfie Background

Chaeun Hong (hong.930), Yuting Yang (yang.5493)

1. Introduction

In the project, we extracted backgrounds from selfies and applied an emoji mosaic effect using emojis based on edge detection, background subtraction, template matching, Euclidean distance, color histogram comparison, and covariance matrix algorithm. Our design enables users to experience a digital cyber world using a simple artistic filter.

2. Data

There are two kinds of data we used for the project - selfie image and collection of emoji. Selfie images are from googling with the keywords ‘iPhone portrait image’ and ‘selfie’. The emoji collection is from the GitHub repository (https://github.com/ericandrewlewis/emoji-mosaic).

3. Process and Techniques

a. Background extraction

For background extraction, we did simple preprocessing for each image. If each pixel is brighter than (150,150,150) based on RGB color, then lower the pixel brightness to 50 than the original brightness. Then, we used edge detection and the principle of background subtraction. We conducted edge detection with Sobel, Gaussian, Canny, and Laplacian of Gaussian. The most important one in edge detection is not to detect all edges in the image. Instead, it is important to get edges between the foreground (object) and the background and ignore detailed edges on the background. Using Sobel, and Canny, there are noises in the background (Figure 1, 2). It was hard to remove them using the post-processing techniques that we used for the project. Therefore, we used the Gaussian filter for smoothing first, then used the Laplacian of Gaussian filter to detect appropriate edges (Figure 3, 4). The best results are shown when the mask size is 5-by-5 for Gaussian smoothing and 3-by-3 for Laplacian of Gaussian filter, the sigma value for Gaussian smoother is 0.6 and for Laplacian of Gaussian is 0.20.

Figure 1. Edge detection with sobel filter

Figure 2. Edge detection with Canny

Figure 3. Gaussian filter

Figure 4. Laplacian of Gaussian filter

Then, we did noise removal using median filtering (Figure 5). Compared to Figure 4, you can see there are fewer white parts (noises) in the image. After removing noises, we used dilation and fill the region inside the edges (Figure 6). There are still background pixels, so we removed lower-brightness pixels except for the filled part. Then use dilation and fill the region again (Figure 7). For the tight edges between foreground and background, we did erosion finally.

Figure 5. Noise removal

Figure 6. 1st dilation & fill

Figure 7. 2nd dilation & fill

Figure 8. Erosion

Finally, we separated foreground and background using the principle of background subtraction. There are final examples below.

Figure 9. Example results

b. Emoji mosaic effect

Six approaches were experimented on to generate an emoji mosaic effect on the photo, including three template matching approaches (SAD, SSD, and NCC), modified covariance tracking, Bhattacharyya Coefficient, and Euclidean distances between 3x1 RGB vectors. 846 Apple emojis were read and resized into 10x10 RGB images, and stored in a 4D matrix (10x10x3x846). Except for the last approach, all the other approaches were applied based on the preprocessing of segmenting the original background into 10x10 patches and using the techniques above to find the best match for each patch among the candidate emoji images.

Inspired by SLIC Superpixel with extremely high compactness, in the last approach, the original is preprocessed by "Pixelization", which is a process to segment the image into many 10x10 regions and fulfill the region with its mean color. Compared with the result of SLIC Superpixel, Pixelization can generate completely square pixel clusters (Figure 10). Then the emoji mosaic is matched for each amplified pixel by simply computing the Euclidean distances between their mean colors.

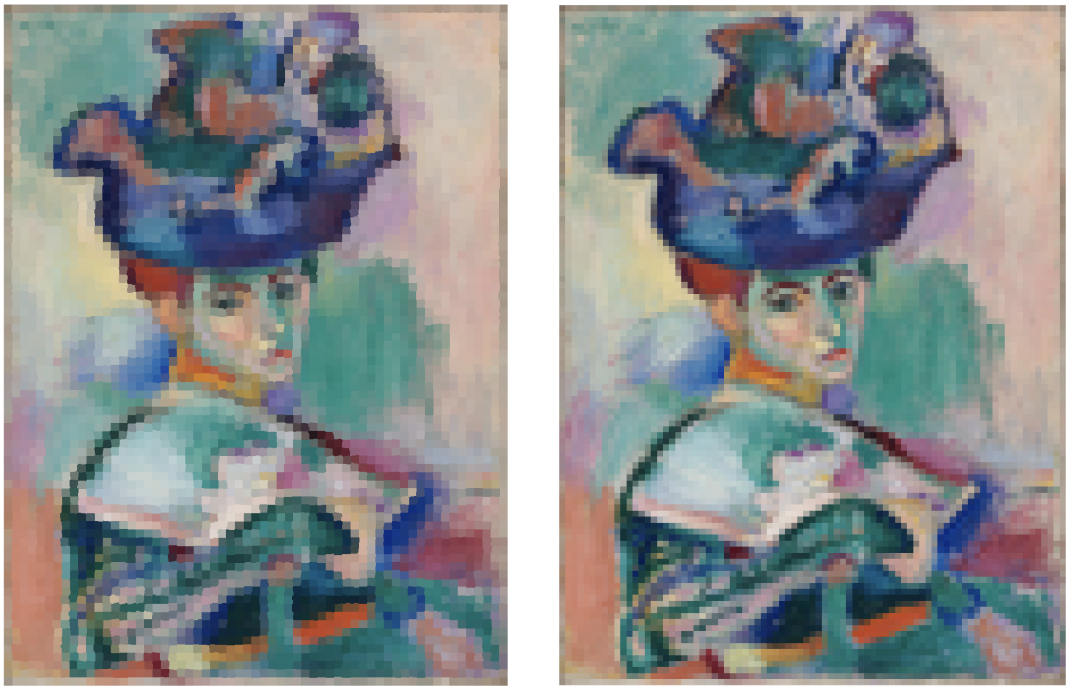


Figure 10. Pixel clusters generated by the SLIC Superpixel algorithm (left) and "Pixelization" (right)

Since NCC can not generate recognizable results, only the other five approaches' results are shown and compared in *Figure* 11-14.

Figure 11. Original image (1). Result image of SAD (2). Result image of SSD (3). Result image of Modified Covariance Tracking (4).

Figure 12. Result image of Pixelization + Euclidean distance between RGB vectors (left) Result image of evaluating the similarity between color histograms using Bhattacharyya Coefficient (right).

Figure 13. Original image (1). Result image of SAD (2). Result image of SSD (3). Result image of Modified Covariance Tracking (4).

Figure 14. Result image of Pixelization + Euclidean distance between RGB vectors (left) Result image of evaluating the similarity between color histograms using Bhattacharyya Coefficient (right).

4. Evaluation and Problems Encountered

Based on the results above, the background extraction works pretty well. The accuracy was calculated by comparing the truth result (created by the background removal function in the PowerPoint slide) and our result. When we calculated the accuracy using the worst result, the accuracy was about to 86 percent. However, there are still limitations to our algorithm for background extraction. It works only for limited images. For example, it does not work well when part of the background is brighter than the overall brightness of the image. Another example is that the image has a strong horizontal or vertical line in the background. Therefore, they need to be solved in the future.

The emoji mosaic effect was first tested on single images without background extraction. The results show that the effect of mosaic using template matching approaches are not satisfied as expected (Figure 16-18). A potential reason may be the impact of the background color of emoji images - most emoji images are only partially covered by RGB pixels. Since template-matching approaches care only the pixel-to-pixel color similarity, it is easily influenced by the background pixels that have zero values in emoji images. Moreover, as NCC removes variations from illumination conditions, the color information was lost but only the image pattern was kept. Therefore, the result generated by NCC is hardly recognized and only part of the portrait’s silhouette was observed.

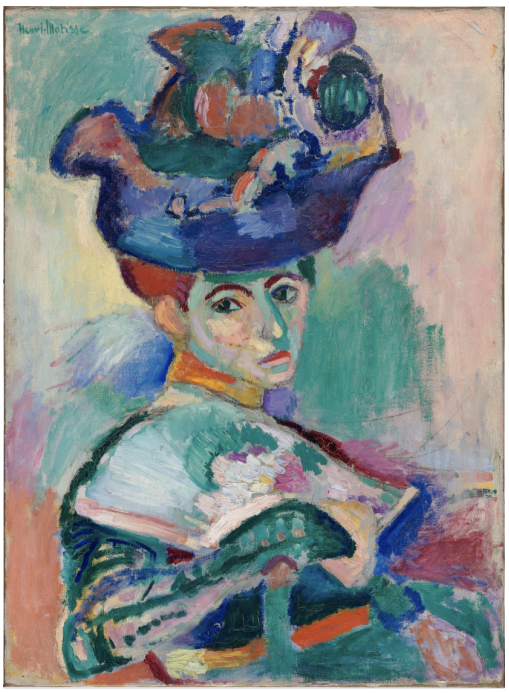


Figure 15. Original image

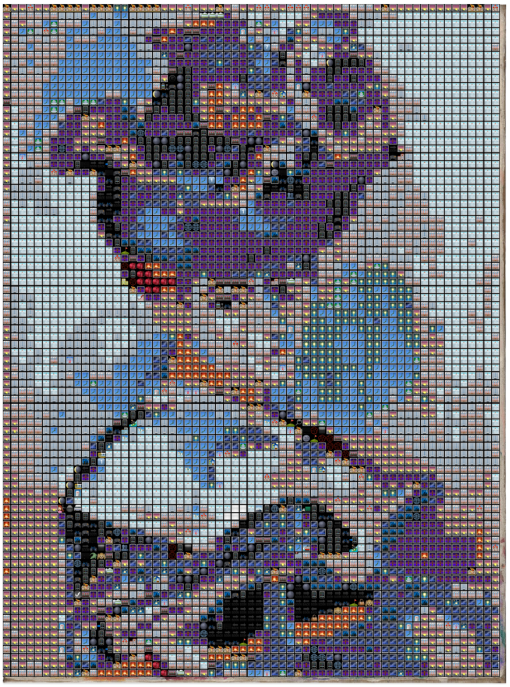


Figure 16. Emoji mosaic effect generated by using Sum-of-Absolute Differences (SAD)

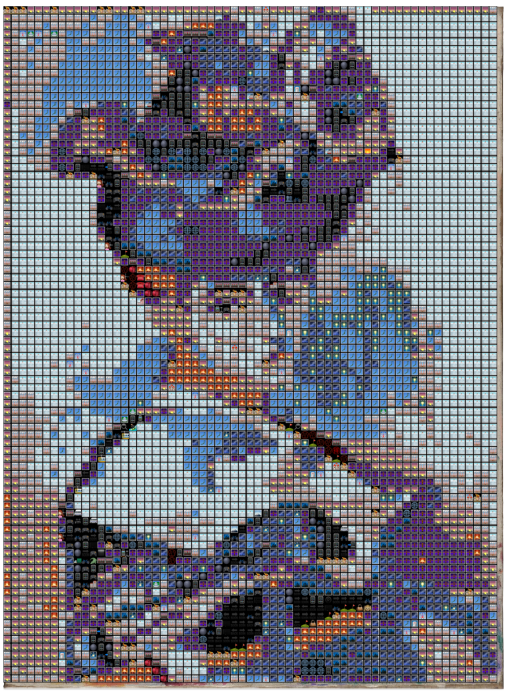


Figure 17. Emoji mosaic effect generated by using Sum-of-Squared Differences (SSD)

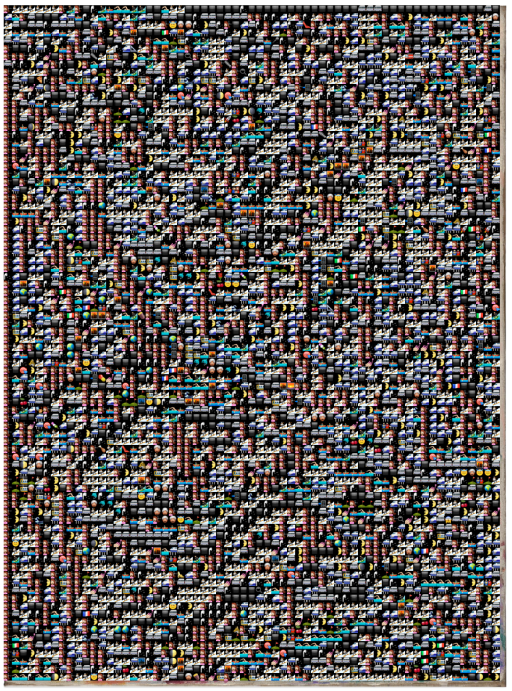


Figure 18. Emoji mosaic effect generated by using Normalized Cross-Correlation (NCC)

To retain color information only, covariance tracking is modified by removing x and y from the feature vector, making the 3x1 RGB vector into a feature vector. Modified covariance tracking is based on a 3x3 covariance matrix for the model (each patch in the original image) and candidates (emoji image). However, the result is not ideal either (Figure 19).

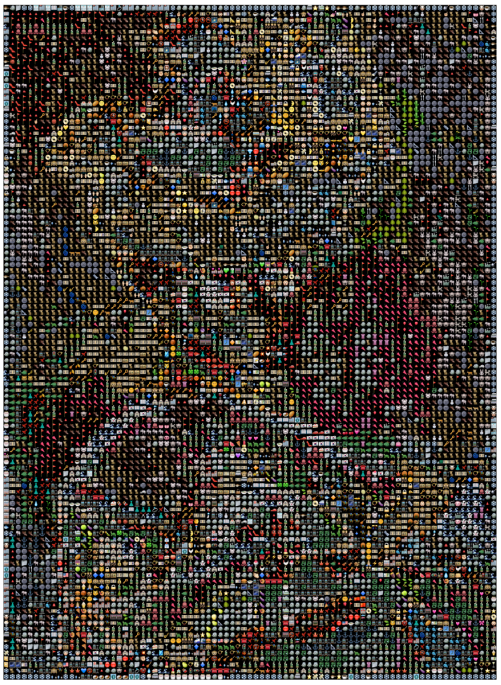


Figure 19. Emoji mosaic effect generated by using Modified Covariance Tracking

The last two approaches generate the best emoji mosaic effect. The first approach is by evaluating the similarities between the color histograms of the model and the candidate using the Bhattacharyya Coefficient (Figure 20). 3D Color Histogram served as a way to quantize the color, maximizing the effect of the mosaic. However, computing color histograms for all patches and emojis is time-consuming. The last approach solved this problem through Image Segmentation. The result indicates that this method can generate the most satisfying emoji mosaic effect while greatly reducing running time (Figure 21).

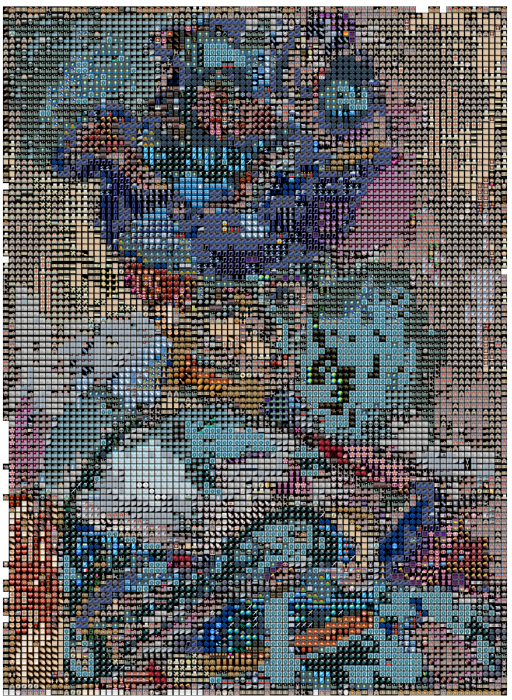


Figure 20. Emoji mosaic effect generated by maximizing similarity between color histograms using Bhattacharyya Coefficient.

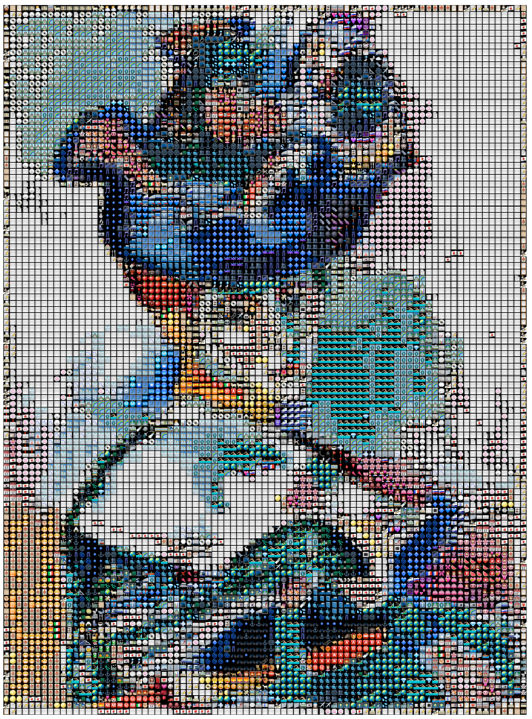


Figure 21. Emoji mosaic effect generated by minimizing Euclidean distances between mean colors of amplified pixel and candidate emojis

5. Conclusion

Through this project, we learned that we do not need to detect all edges for the background extraction and how the sigma values & filter size affect the edge detection. In terms of generating emoji mosaic effects, our original plan is using NCC, but it turns out that NCC generated the worst result. The algorithms of motion tracking and template matching introduced in the lectures capture not only color properties but also spatial properties like location, while color information is the most important feature of mosaic. Therefore, it cannot be directly applied to matching mosaics without further modification. If more time is allowed for the project, further processes of the project such as higher accurate object extraction are likely to be able to improve the mosaic effect. In addition, image segmentation helps to greatly save the running time of pixel-by-pixel analysis. We also aim to turn this design into a video filter and apply it to dynamic visual sources for future improvement.

6. Appendix

[Background Extraction] - Code and images used for evaluation

image_preprocessing.m

```
function img = image_preprocessing(img)
    [row, col, ~] = size(img);
    for i=1:row
        for j=1:col
            if img(row, col, 1) < 150 && img(row, col, 2) < 150 && img(row,
col, 3) < 150
                img(row,col,:) = img(row, col, :);
            else
                img(row, col, :) = img(row, col, :) - 50;
            end
```

```
        end
    end
end
```

LoG_edge_detect.m

```
function [result, x, y] = LoG_edge_detect(img)
    img = double(rgb2gray(img));
    [row, col, ~] = size(img);
    result = zeros(size(img));

    % 1. Gaussian smoother
    filter_size = [5 5];
    sigma = 0.6;
    mask_size = ceil(3*sigma)*2+1;
    g=zeros(mask_size, mask_size);
    y = -1*ceil(3*sigma);
    for i=1:mask_size
        x=-1*ceil(3*sigma);
        for j=1:mask_size
            g(i,j) = 1/(2*pi*sigma^2)*(exp(-(x.^2+y.^2)/(2*sigma^2)));
            x=x+1;
        end
        y=y+1;
    end
    normalized_g = (g - mean(g, 'all', 'omitnan'));
    for i = 1:row - filter_size(1,1)
        for j = 1:col - filter_size(1,2)
            cropped_image = img(i:i+filter_size(1,1)-1,
j:j+filter_size(1,2)-1);
            result(i,j) = sum(normalized_g .* cropped_image,'all'); %
convolution
        end
    end
    figure; imshow(result);
    % 2. Laplacian of Gaussian
    mask_size = (ceil(sigma)*filter_size(1,1)-1)/2;
    [x, y] = meshgrid(-mask_size:mask_size, -mask_size:mask_size);
    sigma = 0.20;
    % for comparsion - sobel filter
    %{
    sobel_filter = fspecial('sobel');
    sobel_x = imfilter(img, sobel_filter');
    sobel_y = imfilter(img, sobel_filter);
    result = sqrt(sobel_x.^2 + sobel_y.^2);
    result = uint8(result);
    result = result > mean(mean(result)) * 2 ;
    %}

    % for comparsion - canny edge detection
    %canny_result = edge(result, 'canny');
```

```
    LoG =
((x.^2+y.^2-2*sigma^2)).*(exp(-(x.^2+y.^2)/(2*sigma^2)))/(sum(exp(-(x.^2+y.^2
)/(2*sigma^2)),'all')*(sigma^4));
    normalized_LoG = (LoG - mean(LoG, 'all', 'omitnan'));

    for i = 1:row - filter_size(1,1)-1
        for j = 1:col - filter_size(1,2)-1
            cropped_image = result(i:i+filter_size(1,1)-1,
j:j+filter_size(1,2)-1);
            result(i,j) = sum(normalized_LoG .* cropped_image,'all'); %
convolution
        end
    end
    result = uint8(result);
end
```

morphological_improvement.m

```
function result=morphological_improvement(input_image)
    [row, col, ~] = size(input_image);
    input_image = medfilt2(input_image,[2 2]); % noise removal
    % show the process - noise removal
    figure; imshow(input_image);
    se = strel('disk',10);
    input_image = imdilate(input_image,se);
    before_fill = input_image;
    [input_image] = imfill(input_image, 'holes');
    after_fill = input_image;
    % show the process - dilation
    figure; imshow(input_image);

    difference = after_fill - before_fill;

    % remove lower brightness
    for i=1:row
        for j=1:col
            if input_image(i,j) < 150
                input_image(i,j)=0;
            elseif (difference(i,j)==0) && (input_image(i,j) <
max(max(input_image)) - 70)
                input_image(i,j)=input_image(i,j)-150;
            end
        end
    end

    se = strel('disk',20);
    input_image = imdilate(input_image,se);
    result = imfill(input_image, 'holes');
    % show the process - dilation
    figure; imshow(result);
    se2 = strel('disk', 20);
    result = imerode(result, se2);
    % show the process - erosion
    figure; imshow(result);
```

```
end
```

foreground_background_segmentation.m

```
function [background, foreground] =
foreground_background_segmentation(result, original_image)
   [row, col, dim] = size(original_image);

   background = zeros(row, col, dim);
   foreground = zeros(row, col, dim);

   for i=1:row
       for j=1:col
           if result(i, j)==0
               background(i, j, :) = original_image(i, j, :);
           else
               foreground(i,j,:) = original_image(i,j,:);
           end
       end
   end

end
```

background_extraction.m (main)

```
% load images
img = imread('selfie16.png');
RGBImg = imread('selfie16.png');
% image preprocessing - lower brightness to bright part
img = image_preprocessing(img);
% find edges with Laplacian of Gaussian
result = LoG_edge_detect(img);
% show the process - edge detection
figure; imshow(result);
% using dilation & imfill function to get the region of foreground
result = morphological_improvement(result);
% segmentation of foreground and background
[background, foreground] = foreground_background_segmentation(result,
RGBImg);
imwrite(uint8(background),"background.png");
imwrite(uint8(foreground),"foreground.png");
figure; imshow(uint8(background));
figure; imshow(uint8(foreground));
```

evaluation.m

```
%% evaluation for foreground/background segmenation part
ground_truth = imread("ground_truth.png");
result = imread("foreground.png");
[row, col, dim] = size(ground_truth);
count = 0;
total = 0;
for i=1:row
   for j=1:col
       if ground_truth(i,j,:) == result(i,j,:)
           count = count + 1;
```

```
        end
        total = total + 1;
    end
end
accuracy = count / total * 100;
```

Figure 22. Ground truth image (left) and our result (right)

[Emoji Mosaic]

colorHistogram.m

```
function [hist] = colorHistogram(image, bins)
hist = zeros(bins, bins, bins);
[numRows, numCols,~] = size(image);
binSize = round(256/bins);
for i = 1:numRows
    for j = 1:numCols
        R = double(image(i,j,1));
        G = double(image(i,j,2));
        B = double(image(i,j,3));
        binR = ceil((R+1)/binSize);
        binG = ceil((G+1)/binSize);
        binB = ceil((B+1)/binSize);
        hist(binR, binG, binB) = hist(binR, binG, binB)+1;
    end
end
% Normalize the histogram/cube so it sums to 1
hist = hist ./ sum(hist(:));
end
```

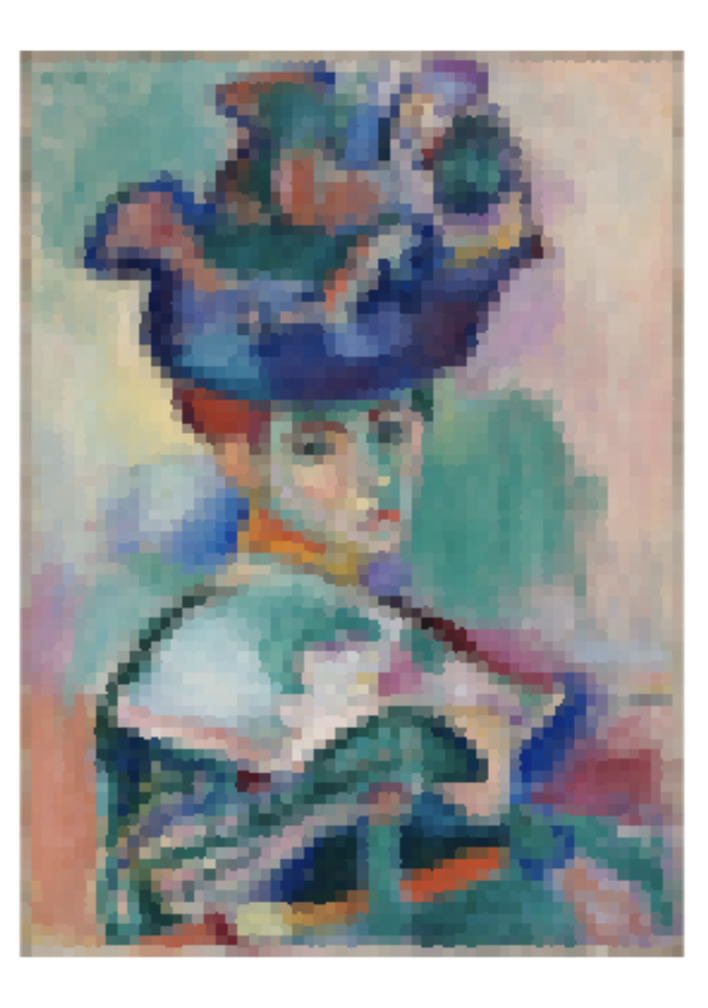
Contents

- Try superpixel
- Approach 1: Pixelization + Euclidean distance
- Approach 2: Modified Covariance Tracking
- Approach 3: NCC
- Approach 4: Color Histogram
- Approach 5: SSD
- Approach 6: SAD


```
%$Yuting Yang
%$CSE5524 - Project
%$11/20/2022
```

Try superpixel

```
testImg = imread('woman_with_a_hat.png');

N = 5000;  % target number of superpixels
C = 1000;  % compactness

[L, NumLabels] = superpixels(testImg, N, "Compactness", C);

outputImage = zeros(size(testImg),'like',testImg);
idx = label2idx(L);

numRows = size(testImg,1);
numCols = size(testImg,2);
for labelVal = 1:NumLabels
    redIdx = idx{labelVal};
    greenIdx = idx{labelVal}+numRows*numCols;
    blueIdx = idx{labelVal}+2*numRows*numCols;
    outputImage(redIdx) = mean(testImg(redIdx));
    outputImage(greenIdx) = mean(testImg(greenIdx));
    outputImage(blueIdx) = mean(testImg(blueIdx));
end

figure
imshow(outputImage,'InitialMagnification',67);
```



Approach 1: Pixelization + Euclidean distance

Initialize variables

```
mosaicSize = 10;  % square mosaic
numEmoji = 846;

originalIm = im2double(imread('background-16.png'));
[m,n,~] = size(originalIm);
originalIm = originalIm(1:floor(m/mosaicSize)*mosaicSize, 1:floor(n/mosaicSize)*mosaicSize, :);

foreIm = im2double(imread('foreground-16.png'));
foreIm = foreIm(1:floor(m/mosaicSize)*mosaicSize, 1:floor(n/mosaicSize)*mosaicSize, :);

[orgR, orgC, ~] = size(originalIm);
numMosaicR = orgR/mosaicSize;
numMosaicC = orgC/mosaicSize;

% Sigmentation
testL = zeros(size(originalIm,1:2));
i = 0;
for c = 1:numMosaicC
    for r = 1:numMosaicR
        i=i+1;
        testL((r-1)*mosaicSize+1:(r-1)*mosaicSize+mosaicSize, (c-1)*mosaicSize+1:(c-1)*mosaicSize+mosaicSize) = i;
    end
end

% Get mean color (Ref: Superpixel)
outputImage = zeros(size(originalIm),'like',originalIm);
idx = label2idx(testL);
NumLabels = numMosaicC*numMosaicR;
numRows = size(originalIm,1);
numCols = size(originalIm,2);
for labelVal = 1:NumLabels
    redIdx = idx{labelVal};
    greenIdx = idx{labelVal}+numRows*numCols;
    blueIdx = idx{labelVal}+2*numRows*numCols;
    outputImage(redIdx) = mean(originalIm(redIdx));
    outputImage(greenIdx) = mean(originalIm(greenIdx));
    outputImage(blueIdx) = mean(originalIm(blueIdx));
end

testIm = outputImage;
copyIm = testIm;

% Read emojis and resize into mosaic size
searchIm = zeros(mosaicSize, mosaicSize, 3, numEmoji);

for i=1:numEmoji
    filename = sprintf('%04d.png', i);
    searchIm(:,:,:,i) = imresize(im2double(imread(filename)), [mosaicSize mosaicSize]);
end

% Calculate mean color for each emoji patch
PList = zeros(mosaicSize,mosaicSize,3,numEmoji);
for n = 1:numEmoji
    P = zeros(mosaicSize,mosaicSize,3);
    for i = 1:mosaicSize
        for j = 1:mosaicSize
            for k = 1:3
                pp = searchIm(:,:,k,n);
                P(i,j,k) = mean(pp(pp>0),"all");  % threshold out black background
            end
        end
    end
    PList(:,:,:,n) = P;
end

% Compare RGB for each segmentation
for r = 1:numMosaicR
    for c = 1:numMosaicC
        tIm= outputImage((1+(r-1)*mosaicSize):r*mosaicSize, (1+(c-1)*mosaicSize):c*mosaicSize, :);
        r1 = tIm(1,1,1);
        g1 = tIm(1,1,2);
        b1 = tIm(1,1,3);

        if r1>0||g1>0||b1>0
            bestD = Inf;
            bestIdx = 0;
            for i = 1:numEmoji  % for every emoji patch
                r2 = PList(1,1,1,i);
                g2 = PList(1,1,2,i);
                b2 = PList(1,1,3,i);
                d=sqrt((r1-r2)^2+(g1-g2)^2+(b1-b2)^2);
                if d<bestD
                    bestD = d;
                    bestIdx = i;
                end
            end
            copyIm((1+(r-1)*mosaicSize):r*mosaicSize,  (1+(c-1)*mosaicSize):c*mosaicSize, :) = searchIm(:,:,:,bestIdx);
        end
    end
end

% Display result
resultIm = copyIm+foreIm;
imshow(resultIm);
```

Approach 2: Modified Covariance Tracking

Initialize variables

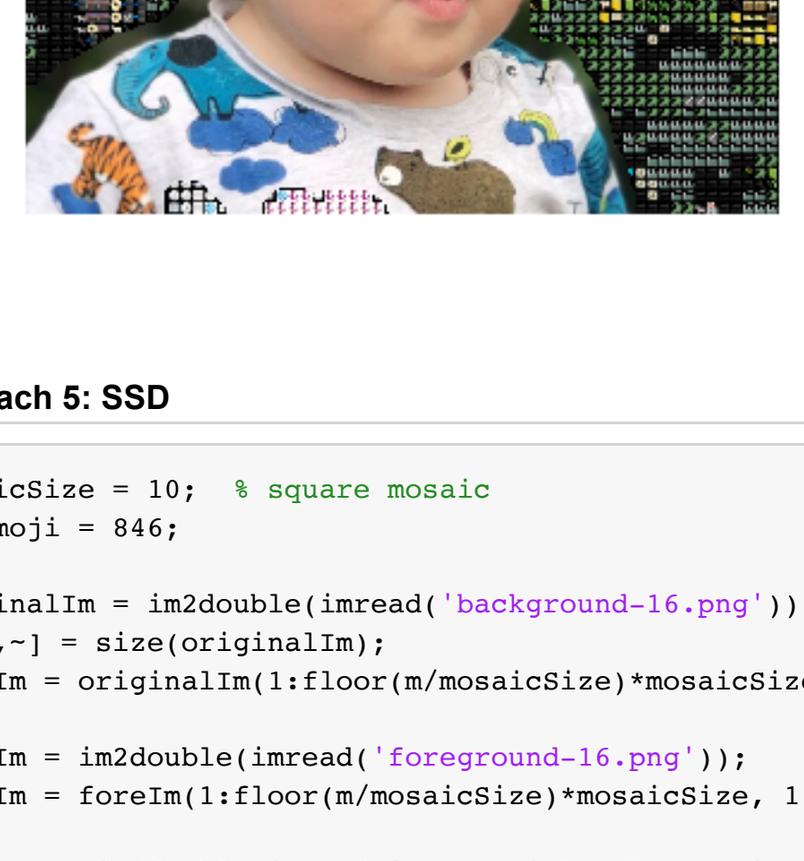
```
mosaicSize = 10;  % square mosaic
numEmoji = 846;

originalIm = im2double(imread('background-16.png'));
[m,n,~] = size(originalIm);
originalIm = originalIm(1:floor(m/mosaicSize)*mosaicSize, 1:floor(n/mosaicSize)*mosaicSize, :);

foreIm = im2double(imread('foreground-16.png'));
foreIm = foreIm(1:floor(m/mosaicSize)*mosaicSize, 1:floor(n/mosaicSize)*mosaicSize, :);

copyIm = originalIm;
[orgR, orgC, ~] = size(originalIm);
numMosaicR = orgR/mosaicSize;
numMosaicC = orgC/mosaicSize;


% Read emojis and resize into mosaic size
searchIm = zeros(mosaicSize, mosaicSize, 3, numEmoji);

for i=1:numEmoji
    filename = sprintf('%04d.png', i);
    searchIm(:,:,:,i) = imresize(im2double(imread(filename)), [mosaicSize mosaicSize]);
end

CModelList = zeros(3,3,numMosaicR,numMosaicC);
for r = 1:numMosaicR
    for c = 1:numMosaicC
        model = originalIm((r-1)*mosaicSize+1:(r-1)*mosaicSize+mosaicSize, (c-1)*mosaicSize+1:(c-1)*mosaicSize+mosaicSize, :);
        miu = reshape(mean(model,1:2), [3 1]);
        sumTemp = zeros(3,3);
        for m = 1:10
            for n = 1:10
                feature = reshape(model(m,n,:),[3 1]);
                temp = feature-miu;
                sumTemp = sumTemp+temp*temp.';
            end
        end
        CModelList(:,:,r,c) = sumTemp./(m*n);
    end
end

CCandidateList = zeros(3,3,i);
miu2 = zeros(3,1);
for i = 1:numEmoji
    candidate = searchIm(:,:,:,i);
    for k = 1:3
        tempC = candidate(:,:,k);
        miu2(k,1) = mean(tempC(tempC>0.1),"all");
    end
    %miu2 = reshape(mean(candidate,1:2), [3 1]);
    sumTemp = zeros(3,3);
    for m = 1:10
        for n = 1:10
            if candidate(m,n,1)>0.5||candidate(m,n,2)>0.5||candidate(m,n,3)>0.5
                feature = reshape(candidate(m,n,:), [3 1]);
                temp = feature-miu2;
                sumTemp = sumTemp+temp*temp.';
            end
        end
    end
    CCandidateList(:,:,i) = sumTemp./(m*n);
end

for r = 1:numMosaicR
    for c = 1:numMosaicC
        CModel = CModelList(:,:,r,c);
        minDistance = 1000;
        for i = 1:numEmoji
            CCandidate = CCandidateList(:,:,i);
            lamda = eig(CModel, CCandidate);
            if lamda > 0
                distance = sqrt(sum(log(lamda).^2));
                if distance < minDistance
                    bestMatch = i;
                    minDistance = distance;
                end
            end
        end
        copyIm((1+(r-1)*mosaicSize):r*mosaicSize,  (1+(c-1)*mosaicSize):c*mosaicSize, :) = searchIm(:,:,:,bestMatch);
    end
end

[m,n,~] = size(originalIm);
for i = 1:m
    for j = 1:n
        if mean(foreIm(i,j,:),"all")~=0
            copyIm(i,j,:) = foreIm(i,j,:);
        end
    end
end
imshow(copyIm);
```

Approach 3: NCC

```
mosaicSize = 10;  % square mosaic
numEmoji = 846;

originalIm = im2double(imread('background-16.png'));
[m,n,~] = size(originalIm);
testIm = originalIm(1:floor(m/mosaicSize)*mosaicSize, 1:floor(n/mosaicSize)*mosaicSize, :);

foreIm = im2double(imread('foreground-16.png'));
foreIm = foreIm(1:floor(m/mosaicSize)*mosaicSize, 1:floor(n/mosaicSize)*mosaicSize, :);

%testIm = im2double(imread('woman_with_a_hat.png'));
[testR, testC, ~] = size(testIm);
copyIm = testIm;
searchIm = zeros(mosaicSize, mosaicSize, 3, numEmoji);

% read emoij patches
for i=1:numEmoji
    filename = sprintf('%04d.png', i);
    searchIm(:,:,:,i) = imresize(im2double(imread(filename)), [mosaicSize mosaicSize]);
end

numPinRow = floor(testR/mosaicSize);
numPinCol = floor(testC/mosaicSize);
for r = 1:numPinRow
    for c = 1:numPinCol
        templateIm= testIm((1+(r-1)*mosaicSize):r*mosaicSize, (1+(c-1)*mosaicSize):c*mosaicSize, :);

        bestNCC = -Inf;
        bestIdx = 1;
        NCClist = zeros(numEmoji);
        for i = 1:numEmoji  % for every emoji patch
            currentNCC = 0;
            for k = 1:3
                P = searchIm(:,:,k,i);
                T = templateIm(:,:,k);
                miu_T = mean(T, "all");
                sigma_T = std(T, 0, "all");
                currentNCC = currentNCC + computeNCC(P, T, miu_T, sigma_T);
            end
            NCClist(i) = currentNCC;
            if currentNCC > bestNCC
                bestNCC = currentNCC;
                bestIdx = i;
            end
        end
        copyIm((1+(r-1)*mosaicSize):r*mosaicSize,  (1+(c-1)*mosaicSize):c*mosaicSize, :) = searchIm(:,:,:,bestIdx);
    end
end

[m,n,~] = size(testIm);
for i = 1:m
    for j = 1:n
        if mean(foreIm(i,j,:),"all")~=0
            copyIm(i,j,:) = foreIm(i,j,:);
        end
    end
end
imshow(copyIm);
```

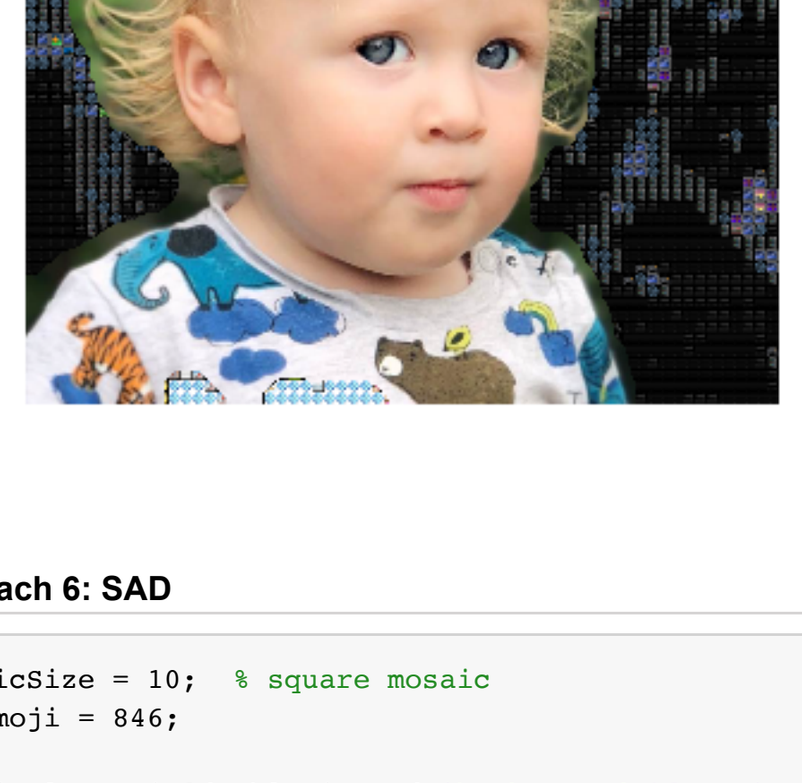
Approach 4: Color Histogram

```
mosaicSize = 10;  % square mosaic
numEmoji = 846;

originalIm = imread('background-16.png');
[m,n,~] = size(originalIm);
testIm = originalIm(1:floor(m/mosaicSize)*mosaicSize, 1:floor(n/mosaicSize)*mosaicSize, :);

foreIm = imread('foreground-16.png');
foreIm = foreIm(1:floor(m/mosaicSize)*mosaicSize, 1:floor(n/mosaicSize)*mosaicSize, :);

%testIm = im2double(imread('woman_with_a_hat.png'));
[testR, testC, ~] = size(testIm);
copyIm = testIm;

% read emoij patches
for i=1:numEmoji
    filename = sprintf('%04d.png', i);
    searchIm(:,:,:,i) = imresize(imread(filename), [mosaicSize mosaicSize]);
end

numPinRow = floor(testR/mosaicSize);
numPinCol = floor(testC/mosaicSize);

% evaluate similarity between color histograms
bins = 16;
for r = 1:numPinRow
    for c = 1:numPinCol
        templateIm= testIm((1+(r-1)*mosaicSize):r*mosaicSize, (1+(c-1)*mosaicSize):c*mosaicSize, :);
        targetHist = colorHistogram(templateIm,bins);

        bestD = -Inf;
        bestIdx = 0;
        for i = 1:numEmoji  % for every emoji patch
            candidateHist = colorHistogram(searchIm(:,:,:,i), bins);
            d = sum(sqrt(targetHist.* candidateHist),'all');
            if d>bestD
                bestD = d;
                bestIdx = i;
            end
        end
        copyIm((1+(r-1)*mosaicSize):r*mosaicSize,  (1+(c-1)*mosaicSize):c*mosaicSize, :) = searchIm(:,:,:,bestIdx);
    end
end

[m,n,~] = size(testIm);
for i = 1:m
    for j = 1:n
        if mean(foreIm(i,j,:),"all")~=0
            copyIm(i,j,:) = foreIm(i,j,:);
        end
    end
end
imshow(copyIm);
```



Approach 5: SSD

```
mosaicSize = 10;  % square mosaic
numEmoji = 846;

originalIm = im2double(imread('background-16.png'));
[m,n,~] = size(originalIm);
testIm = originalIm(1:floor(m/mosaicSize)*mosaicSize, 1:floor(n/mosaicSize)*mosaicSize, :);

foreIm = im2double(imread('foreground-16.png'));
foreIm = foreIm(1:floor(m/mosaicSize)*mosaicSize, 1:floor(n/mosaicSize)*mosaicSize, :);

%testIm = im2double(imread('woman_with_a_hat.png'));
[testR, testC, ~] = size(testIm);
copyIm = testIm;
searchIm = zeros(mosaicSize, mosaicSize, 3, numEmoji);

% read emoij patches
for i=1:numEmoji
    filename = sprintf('%04d.png', i);
    searchIm(:,:,:,i) = imresize(im2double(imread(filename)), [mosaicSize mosaicSize]);
end

numPinRow = floor(testR/mosaicSize);
numPinCol = floor(testC/mosaicSize);
for r = 1:numPinRow
    for c = 1:numPinCol
        templateIm= testIm((1+(r-1)*mosaicSize):r*mosaicSize, (1+(c-1)*mosaicSize):c*mosaicSize, :);

        minSSD = Inf;
        bestIdx = 0;
        for i = 1:numEmoji  % for every emoji patch
            currentSSD = 0;
            for k = 1:3
                P = searchIm(:,:,k,i);
                T = templateIm(:,:,k);
                currentSSD = currentSSD+sum((P-T).*(P-T),"all");
            end
            if currentSSD < minSSD
                minSSD = currentSSD;
                bestIdx = i;
            end
        end
        copyIm((1+(r-1)*mosaicSize):r*mosaicSize,  (1+(c-1)*mosaicSize):c*mosaicSize, :) = searchIm(:,:,:,bestIdx);
    end
end

[m,n,~] = size(testIm);
for i = 1:m
    for j = 1:n
        if mean(foreIm(i,j,:),"all")~=0
            copyIm(i,j,:) = foreIm(i,j,:);
        end
    end
end
imshow(copyIm);
```



Approach 6: SAD

```
mosaicSize = 10;  % square mosaic
numEmoji = 846;

originalIm = im2double(imread('background-16.png'));
[m,n,~] = size(originalIm);
testIm = originalIm(1:floor(m/mosaicSize)*mosaicSize, 1:floor(n/mosaicSize)*mosaicSize, :);

foreIm = im2double(imread('foreground-16.png'));
foreIm = foreIm(1:floor(m/mosaicSize)*mosaicSize, 1:floor(n/mosaicSize)*mosaicSize, :);

%testIm = im2double(imread('woman_with_a_hat.png'));
[testR, testC, ~] = size(testIm);
copyIm = testIm;
searchIm = zeros(mosaicSize, mosaicSize, 3, numEmoji);

% read emoij patches
for i=1:numEmoji
    filename = sprintf('%04d.png', i);
    searchIm(:,:,:,i) = imresize(im2double(imread(filename)), [mosaicSize mosaicSize]);
end

numPinRow = floor(testR/mosaicSize);
numPinCol = floor(testC/mosaicSize);
for r = 1:numPinRow
    for c = 1:numPinCol
        templateIm= testIm((1+(r-1)*mosaicSize):r*mosaicSize, (1+(c-1)*mosaicSize):c*mosaicSize, :);

        minSAD = Inf;
        bestIdx = 0;
        for i = 1:numEmoji  % for every emoji patch
            currentSAD = 0;
            for k = 1:3
                P = searchIm(:,:,k,i);
                T = templateIm(:,:,k);
                currentSAD = currentSAD+sum(abs(P-T),"all");
            end
            if currentSAD < minSAD
                minSAD = currentSAD;
                bestIdx = i;
            end
        end
        copyIm((1+(r-1)*mosaicSize):r*mosaicSize,  (1+(c-1)*mosaicSize):c*mosaicSize, :) = searchIm(:,:,:,bestIdx);
    end
end

[m,n,~] = size(testIm);
for i = 1:m
    for j = 1:n
        if mean(foreIm(i,j,:),"all")~=0
            copyIm(i,j,:) = foreIm(i,j,:);
        end
    end
end
imshow(copyIm);
```