Special Lecture for Computer Science – COSE490

Digital Image Processing :: Histogram Processing

Due date : 2022-10-02

고려대학교 컴퓨터학과 2017320108

고재영

개발 환경 : Matlab Desktop

과제 만기일 : 2022-10-02

제출 날짜 : 2022-09-29

최종 제출: 2022-09-29

재제출 사유: 9/29 수업 초반부 레포트 작성에 대한 전체 공지를 듣고, AHE 부분에 관련하여 코드 설명을 추가하기로 함.

[0] – 과제 설명

- You need to write a short report explaining the results (test with your own images, try with different tile size for adaptive histogram equalization).

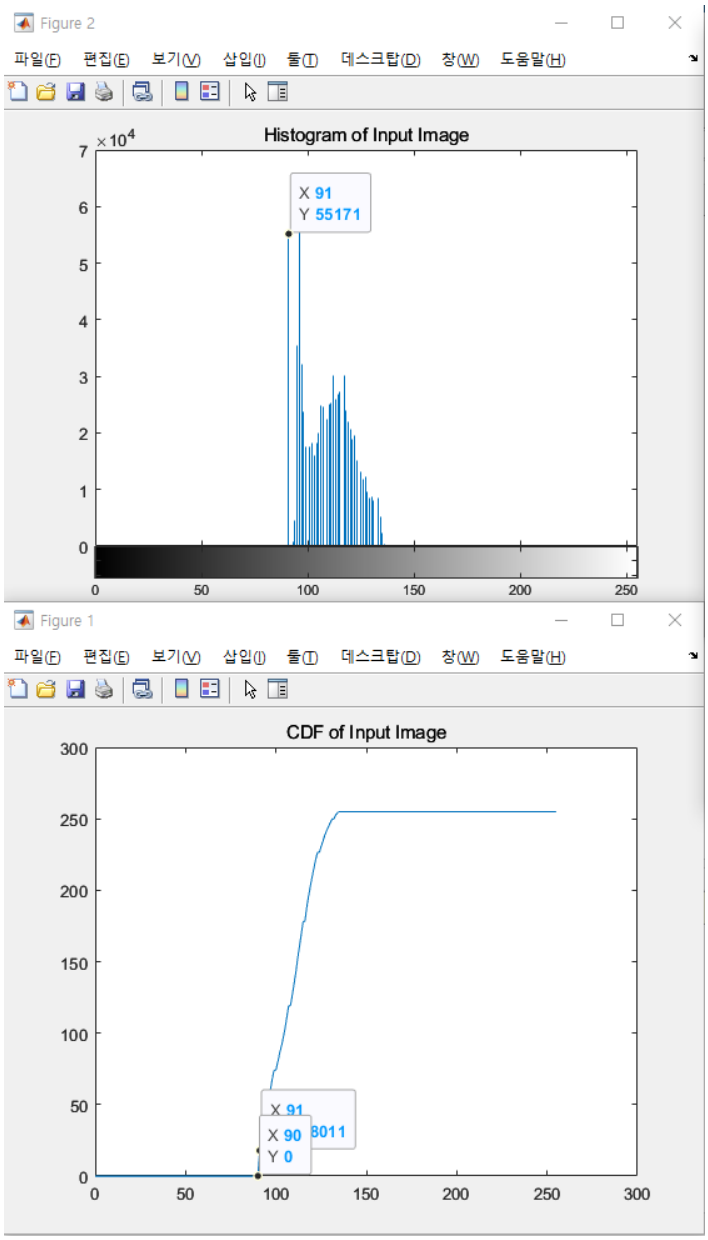
- Histogram Processing에 관련하여 Histogram Equalization에 관하여 matlab 내장함수를 사용하지 않고 원리대로 직접 구현을 한다.

- 주어진 input image에 대해서, 8 bit pixel의 gray-scale image에 관해 다루기 때문에,

0 ~ 2^8 – 1 (= 255)로 총 256가지의 intensity level을 가진다.

[1] – myCDF

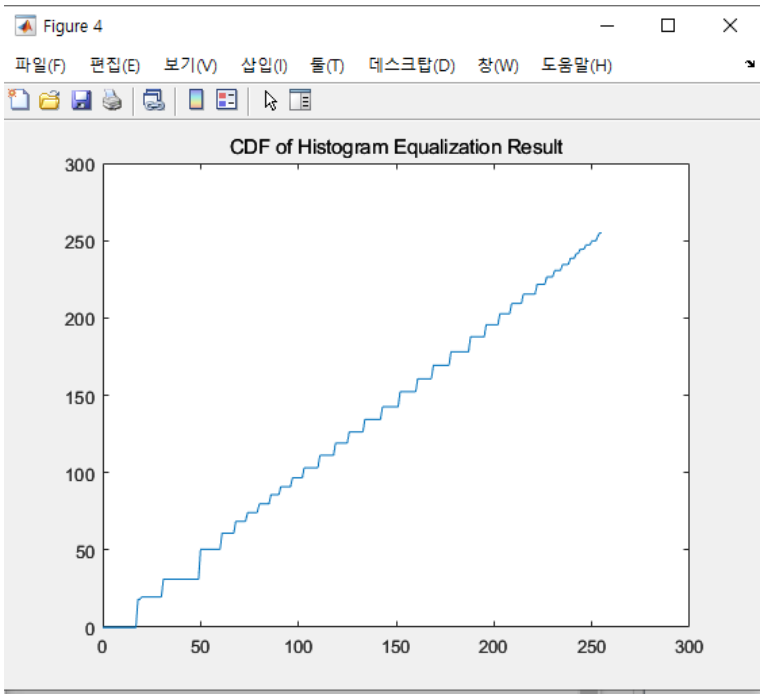
myCDF 함수는 input image를 입력받아서 해당 입력값에 대해서 누적확률분포함수, 즉 cdf를 결과값으로 리턴하는 함수이다. 필자는 구현을 위해서 확률분포함수 pdf를 먼저 구한 후, 각 intensity level이 증가함에 따라 더해져서 누적되는 형태로 구현했다.

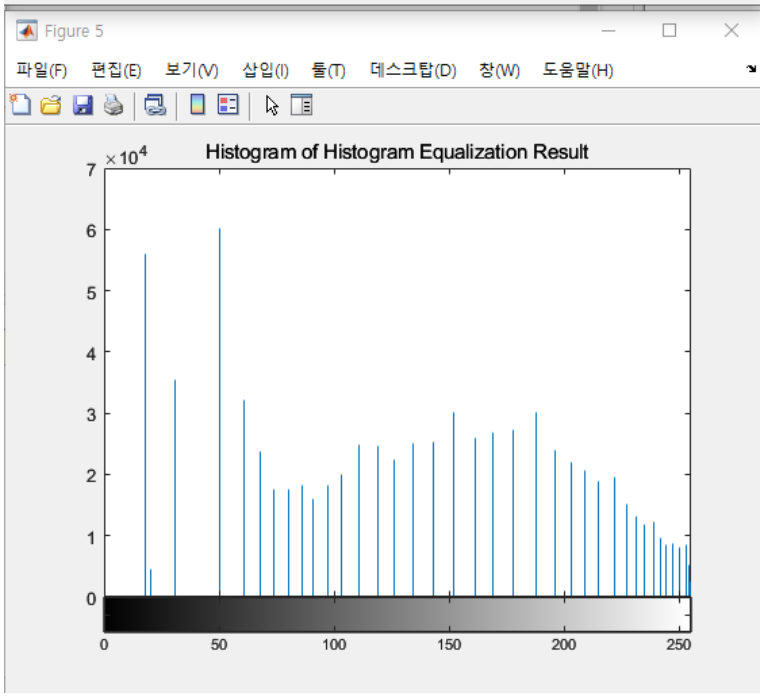


위 이미지를 통해 myPDF로 인해 나타난 figure 1. Figure 2를 보아 제대로 구현되었음을 알 수 있다.

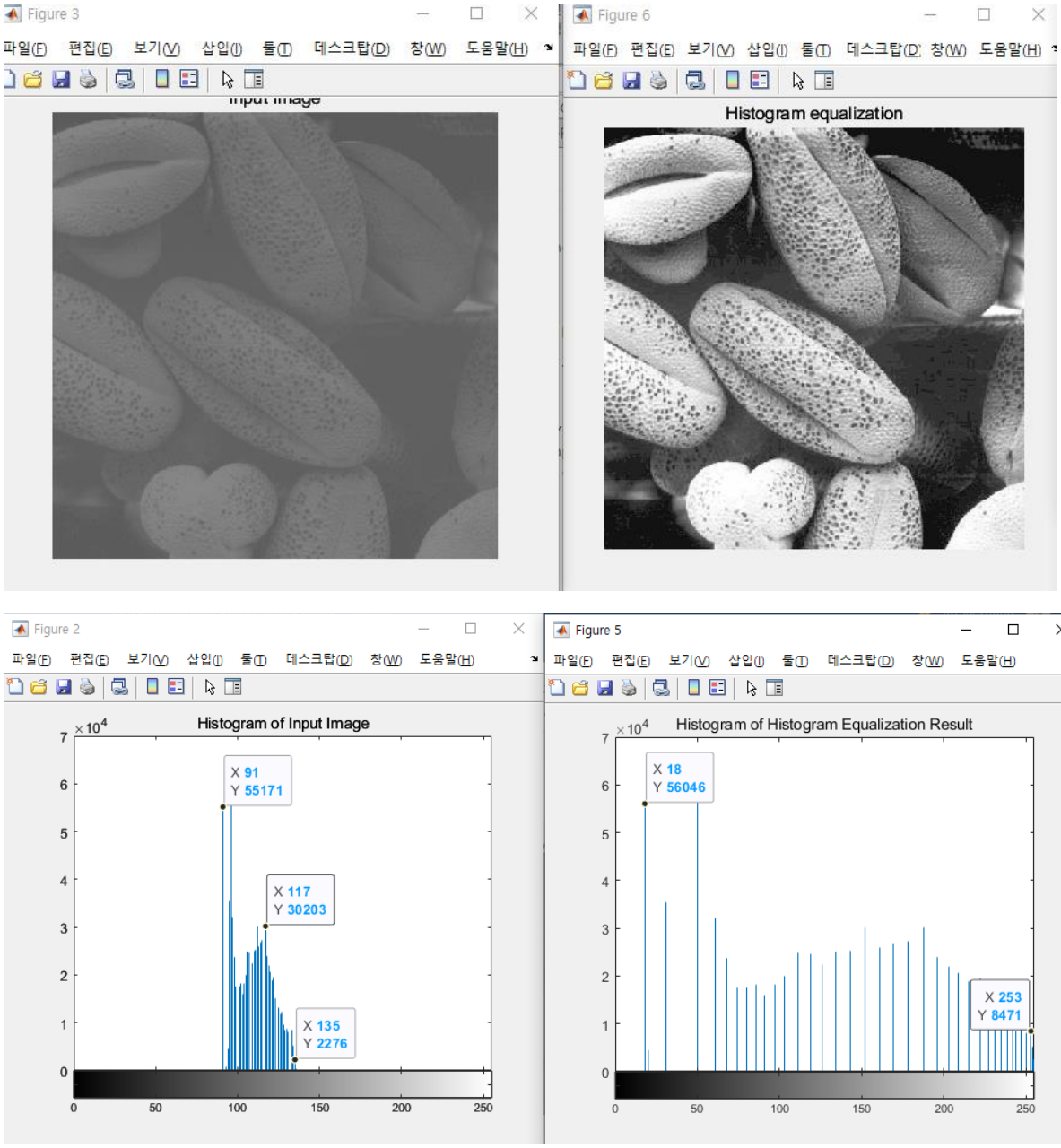
[2] – myHE

두 번째로, myHE 함수는 input image를 입력받아서 해당 이미지에 대해서 histogram equalization을 수행하는 함수이다. 이 때, 기본 histogram equalization처럼 이미지 전역에 대해서 수행하게 된다. 또, histogram equalization을 구현하는 데에, transformation function으로 역할하는 s = T(r) 의 과정에 있어 CDF를 계산할 때에 앞서 구현한 myCDF를 통해 간단히 구할 수 있다.





위 figure 5, 6 이미지는 HE를 통한 결과의 히스토그램과 CDF 이미지이다. 결과가 완전히 straight하게 평탄하지 않은 이유는, input 값 자체가 discrete하기 때문이다.



위 figure 3와 figure 6는 오리지날 input image와 HE 수행 이후 결과 이미지이고, figure 2와 figure 5는 각각에 대한 histogram이다. 기존 original image의 경우는 intensity level이 91~135 정도의 영역에 밀집한 형태였기 때문에 가장 밝은 값과 가장 어두운 값의 차이가 50을 넘지 않아 불분명하고 식별하기 어려운 이미지였다고 해석할 수 있다. 그러나 Histogram Equalization 이후, 결과 histogram은 intensity level이 18~255까지 거의 전반적인 영역을 아우르기 때문에, high contrast 즉 대비가 커져 보다 가시성 있는 이미지를 얻을 수 있다.

[3] – myAHE

세 번째로 myAHE 함수는 histogram equalization을 수행하되, 전체 이미지를 uniform한 크기로 grid처럼 tile로 구역을 나누어 각 구역 마다 HE를 수행한 후 neighbor에 관한 선형 보간(linear interpolation)을 수행한다. 이 때, 이미지 타일 중 가장자리 모서리나 가장자리 면과 같은 경우는 가능한 이웃 타일에 대해서만 선형 보간을 이루거나 해당 타일의 값만 취한다.

<코드 설명>

```
% <1> Split input image into tiles, and setting values for tile
tileNumRow = numtiles(1);       % in this example, 10
tileNumCol = numtiles(2);
tileWidth = ceil(dimX / tileNumRow);       % in this example, 89
tileHeight = ceil(dimY / tileNumCol);
```

input image를 나눌 타일 개수와 각 타일의 가로, 세로 사이즈에 대한 변수값 저장

예시의 경우 889 x 889 이미지를 10 x 10 타일로 쪼개기 때문에 타일의 가로, 세로 사이즈가 정확히 떨어지지 않는데, ceil 함수로 올림을 통해 걸쳐진 타일에 대해서도 포함시키도록 한다.

```
% <2> Compute mapping func CDF for each tile
tileCDF = zeros(tileNumRow, tileNumCol, 256);
for i = 1:tileNumRow
    for j = 1:tileNumCol
        initRow = tileWidth * (i - 1) + 1;
        initCol = tileHeight * (j - 1) + 1;
        currentTile = input(initRow:min(initRow + tileWidth, end),
initCol:min(initCol + tileHeight, end));
        tileCDF(i, j, :) = myCDF(currentTile);
    end
end
```

input image를 타일로 나누고, for문을 이용해 순서대로 탐색하며 해당 타일 정보를 input image로부터 추출하고, tileCDF란 삼차원 배열을 정의해서 각 타일마다 local한 CDF를 저장하도록 한다. 이 때, CDF는 앞서 myCDF로 구현한 함수를 이용해 쉽게 얻는다.

```
% user custom func : find Center pixel's coordinate with knowing tileNo.
function result = TileCenterCoord(tileNumX, tileWidth)
    result = tileWidth * tileNumX - ceil(tileWidth / 2);
end

% user custom func : for Linear Interpolation
function result = LinearInterpolation(target, point1, point2, value1, value2)
    % suppose <point1 - target - point2>
    a = abs(target - point1);
    b = abs(point2 - target);
    result = abs((a / (a + b)) * value2 + (b / (a+b)) * value1);
    result = round(result);
end
```

이제, 각 타일의 CDF를 알고 있기 때문에, 각 픽셀에 대해서 구역에 알맞은 케이스에 따라 Linear interpolation을 진행시키도록 한다. 이 때, 편의를 위해 타일 넘버와 타일의 변 사이즈를 알고 있을 때 타일의 정중앙 픽셀의 좌표를 얻는 함수 TileCenterCoord를 사용자 정의 함수로 구현한다. 그리고 선형 보간에 대해서도 내장함수가 물론 존재하겠지만, 과제의 취지에 맞도록 목표 픽셀의 좌표, 다른 두 점의 좌표, 해당 두 점의 intensity value를 입력받아 보간된 intensity value를 리턴하는 LienarInterpolation 함수도 사용자 정의한다.

```
% <3> Compute four different HE value and Linear Interpolation
% (i,j) as a pixel coordinate for searching
for i = 1:dimX
    for j = 1:dimY
        % want to know which tile (i, j) pixel is located on
        currentTileNumberX = floor(i / tileWidth) + 1;
        currentTileNumberY = floor(j / tileHeight) + 1;
        % the center pixel of the current tile
        cx = TileCenterCoord(currentTileNumberX, tileWidth);
        cy = TileCenterCoord(currentTileNumberY, tileHeight);
```

픽셀을 전부 탐색해가며, 해당 픽셀이 먼저 어떤 타일에 위치하고 있는지, 해당 타일의 중앙 좌표는 무엇인지 변수값 저장한다.

```
% then we can check the possible neighbor tiles :: multiple cases
% Horizontal
if (i < cx) % pixel locates left side of tile center
    if (currentTileNumberX == 1)
        % No Neighbor in left side
        neighborTileNumberX = -1;
    else
        % neighbor in left side
        neighborTileNumberX = currentTileNumberX - 1;
    end
else    % pixel locates right side of tile center
    if (currentTileNumberX == tileNumRow)
        % No Neighbor in right side
        neighborTileNumberX = -1;
    else
        % neighbor in right side
        neighborTileNumberX = currentTileNumberX + 1;
    end
end

% Vertical
if (j < cy) % pixel locates top of tile center
    if (currentTileNumberY == 1)
        % No Neighbor in top side
        neighborTileNumberY = -1;
    else
        % neighbor in top side
        neighborTileNumberY = currentTileNumberY - 1;
    end
else    % pixel locates bottom of tile center
    if (currentTileNumberY == tileNumCol)
        % No Neighbor in bottom side
        neighborTileNumberY = -1;
    else
        % neighbor in bottom side
        neighborTileNumberY = currentTileNumberY + 1;
    end
end
```

해당 변수값을 이용하여 이제 픽셀값과 픽셀이 위치한 타일의 중앙좌표와의 관계에 따라 분류를 시작한다. 예를 들면, 속한 타일의 중앙좌표에 대해서, 오른쪽 하단에 위치할 경우, adjacent할 것으로 예상되는 이웃 타일은, 해당 타일의 바로 오른쪽 타일, 바로 하단 타일, 그리고 대각방향 오른쪽 하단타일이 후보가 될 것이다. 이 때, 픽셀이 속한 타일이 전체 타일 중 최하단 쪽에 위치한다면 이웃이 될 하단 타일과 대각방향 오른쪽 하단 타일이 존재하지 않을 것이기 때문에 예외처리가 필요하다. 이러한 예외 상황일시에 -1이란 값을 배정해서 예외 처리해주도록 한다.

```
% Compute the Interpolation
       % None of neighbors exists :: itself
       if (neighborTileNumberX == -1 && neighborTileNumberY == -1)
           output(i,j) = tileCDF(currentTileNumberX, currentTileNumberY,
input(i,j) + 1);
```

앞선 예외처리를 통해, 그 어떤 가능한 adjacent 후보가 없으면 interpolation 작업 필요없이 해당 타일로부터 CDF에서 값을 읽어오면 된다.

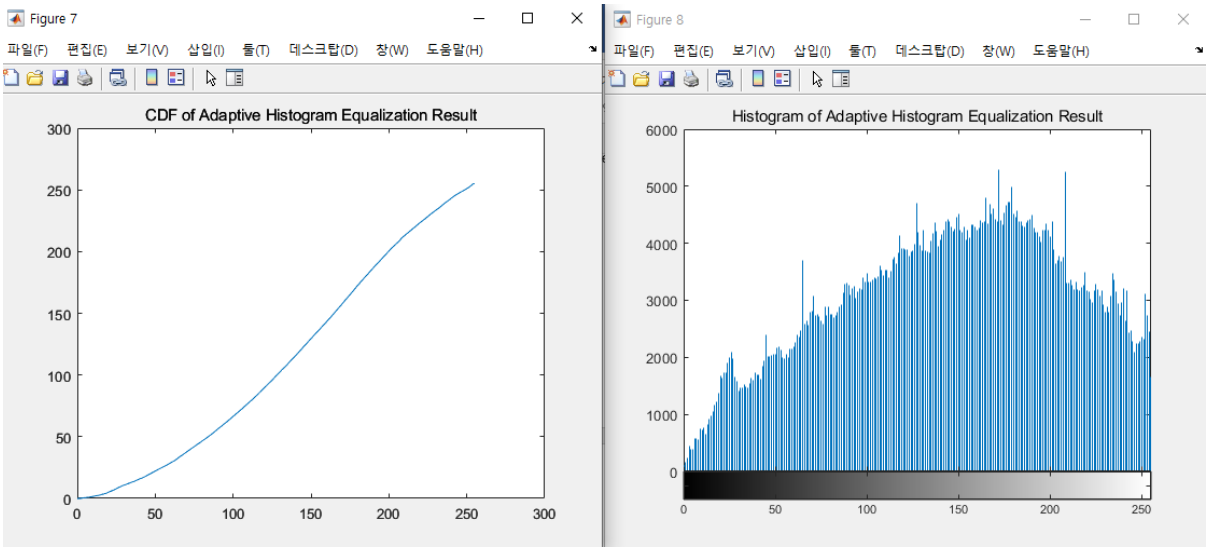
```
elseif (neighborTileNumberX == -1)
           % only Vertical neighbor exists :: y coordinate interpolation
           neighborTileCY = TileCenterCoord(neighborTileNumberY, tileHeight);

           originalInt = tileCDF(currentTileNumberX, currentTileNumberY,
input(i,j) + 1);
           neighborVerticalInt = tileCDF(currentTileNumberX, neighborTileNumberY,
input(i,j) + 1);

           output(i,j) = LinearInterpolation(j, cy, neighborTileCY, originalInt,
neighborVerticalInt);
       elseif (neighborTileNumberY == -1)
           % only Horizontal neighbor exists :: x coordinate interpolation
           neighborTileCX = TileCenterCoord(neighborTileNumberX, tileWidth);

           originalInt = tileCDF(currentTileNumberX, currentTileNumberY,
input(i,j) + 1);
           neighborHorizontalInt = tileCDF(neighborTileNumberX,
currentTileNumberY, input(i,j) + 1);

           output(i,j) = LinearInterpolation(i, cx, neighborTileCX, originalInt,
neighborHorizontalInt);
```

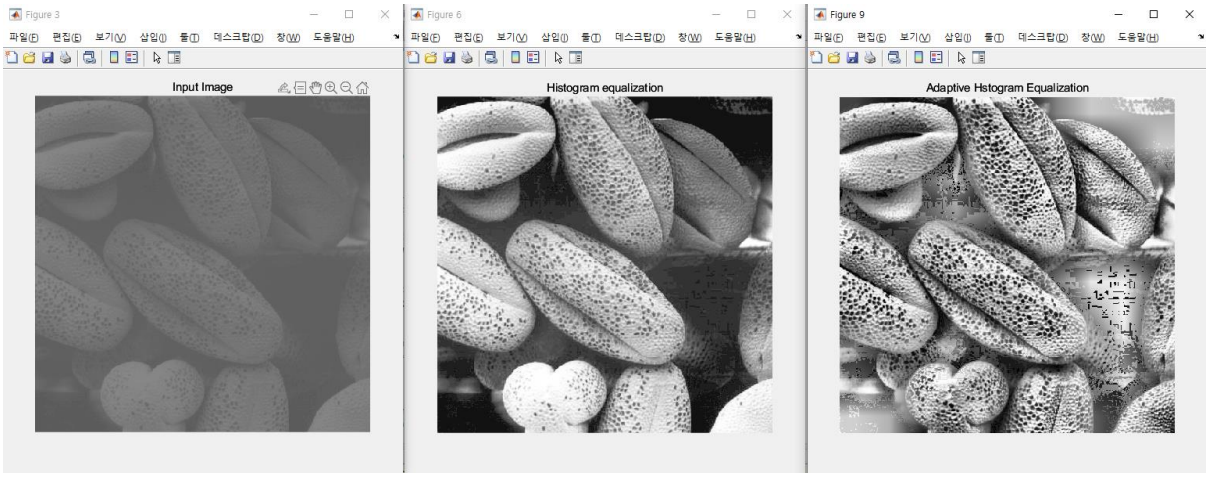
세로 방향 또는 가로 방향에 존재하지 않더라면, 대각도 존재하지 않고 오직 다른 하나만 가능하기에 해당 타일에 대해 Linear Interpolation을 수행하여 얻으면 된다.

```
else
```

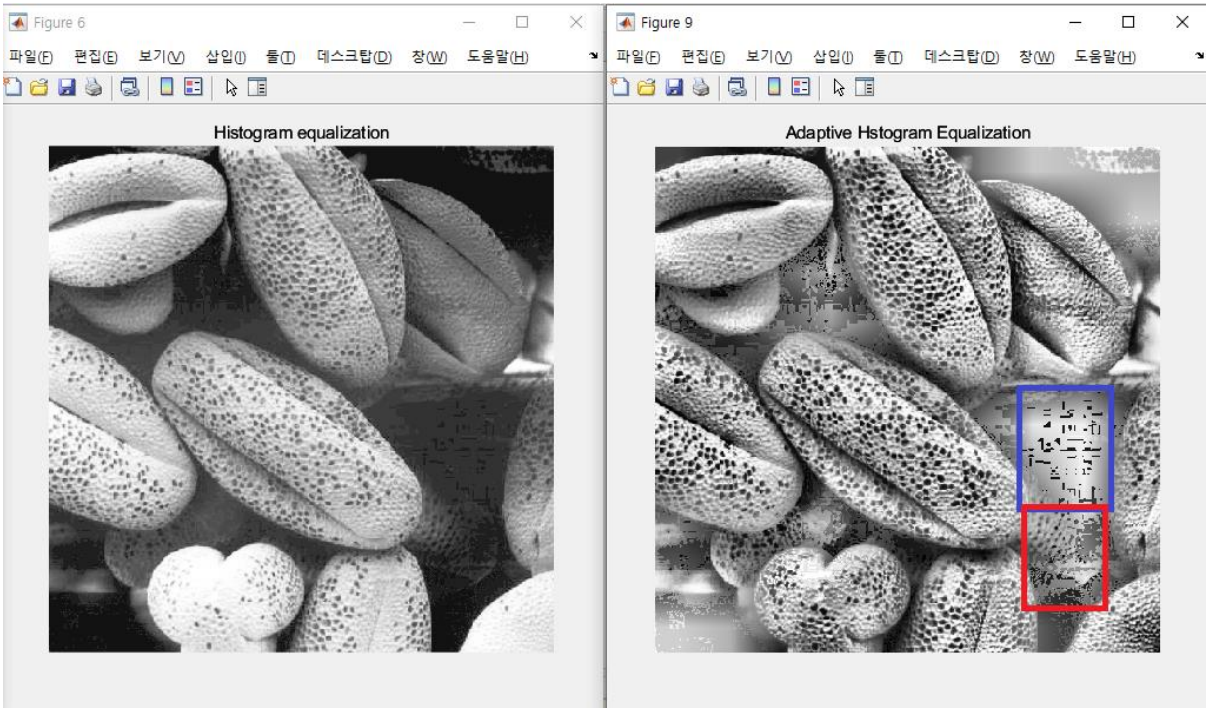
```
            % Bilinear Interpolation
            neighborTileCX = TileCenterCoord(neighborTileNumberX, tileWidth);
            neighborTileCY = TileCenterCoord(neighborTileNumberY, tileHeight);

            originalInt = tileCDF(currentTileNumberX, currentTileNumberY,
input(i,j) + 1);
            neighborVerticalInt = tileCDF(currentTileNumberX, neighborTileNumberY,
input(i,j) + 1);
            neighborHorizontalInt = tileCDF(neighborTileNumberX,
currentTileNumberY, input(i,j) + 1);
            neighborDiagonalInt = tileCDF(neighborTileNumberX, neighborTileNumberY,
input(i,j) + 1);

            resultLerp1 = LinearInterpolation (j, cy, neighborTileCY, originalInt,
neighborVerticalInt);
            resultLerp2 = LinearInterpolation (j, cy, neighborTileCY,
neighborHorizontalInt, neighborDiagonalInt);

            output(i,j) = LinearInterpolation(i, cx, neighborTileCX, resultLerp1,
resultLerp2);
        end
    end
end
```

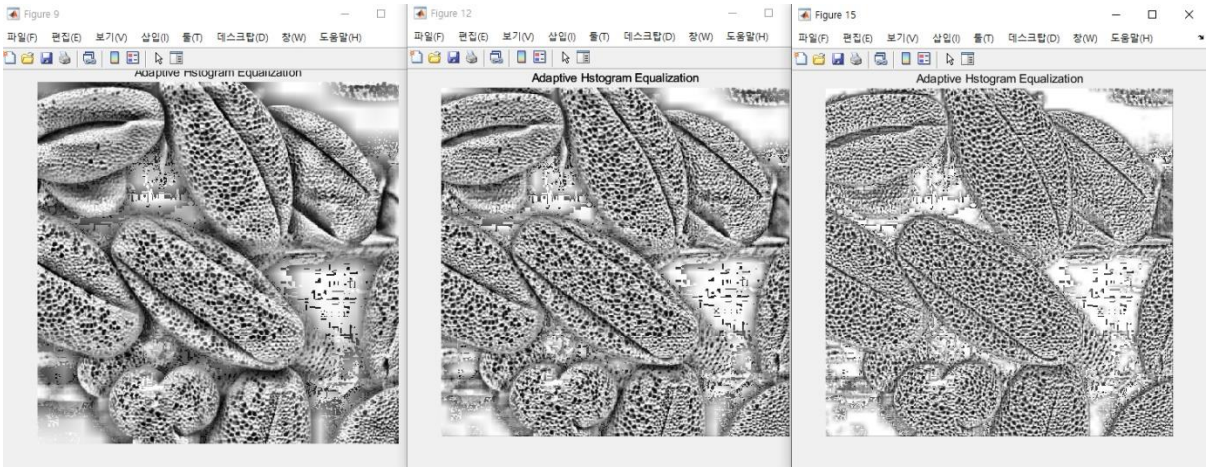
마지막으로, 과제 설명 이미지와 같이 보라색 영역처럼 adjacent 4개 neighbor를 갖는다면, 겹선형보간을 통해 얻도록 한다.



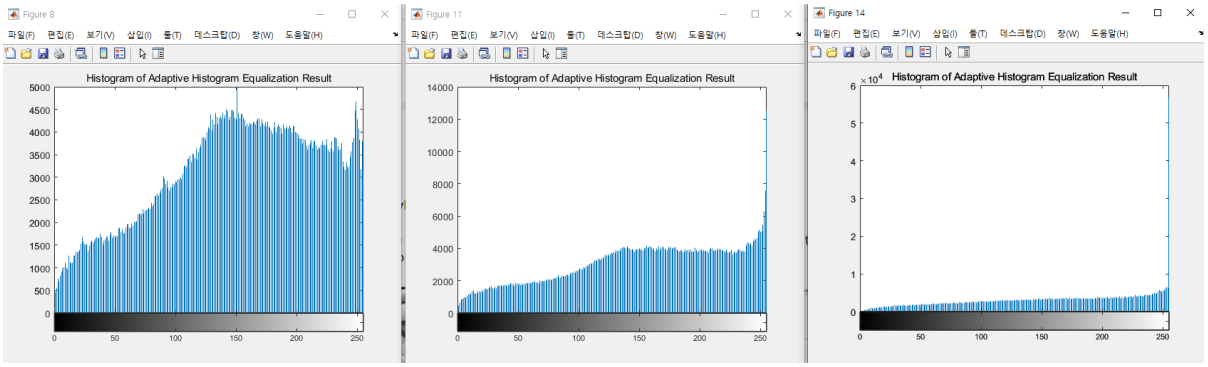


AHE는 adapative 즉 전역에 걸쳐서 HE를 적용하는 게 아니라 local하게 제한된 영역 각각에 HE를 적용한 결과물을 보간하기 때문에, local contrast가 개선된다. HE 결과물과 비교하여 AHE 결과물에서 돌기와 같은 까만 점 요소가 훨씬 두드러지는 점을 볼 수 있다. 또 figure 9에서 빨간 박스 부분을 보면 original image와 HE 결과에서 찾아보기 힘들었던 어두운 부분에 위치한 관찰 요소가 좀 더 식별하기 쉽게 intensity가 개선된 점을 찾을 수 있지만, AHE 과정으로 인해 박스 안의 픽셀처럼 크게 의미없는 요소에 대해서 noise가 강화된 단점도 찾아볼 수 있다.





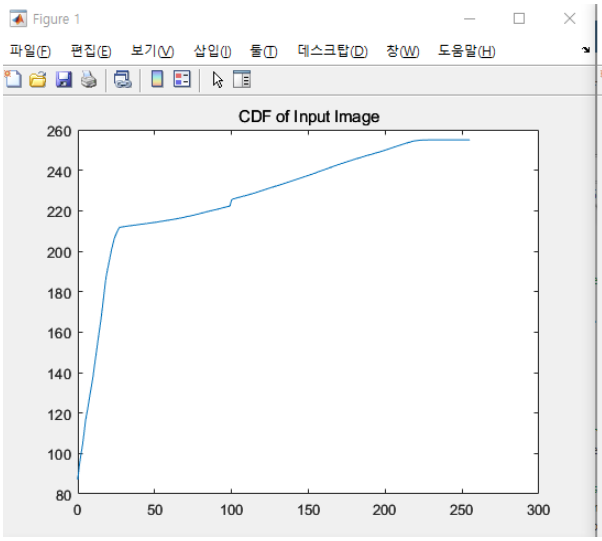
이 세 가지 이미지는 numTile 개수에 대해서 20x20, 40x40, 100x100으로 설정해본 결과이다.

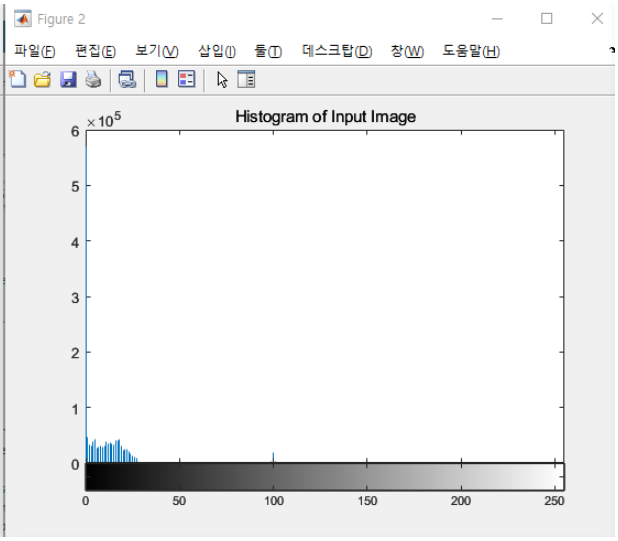


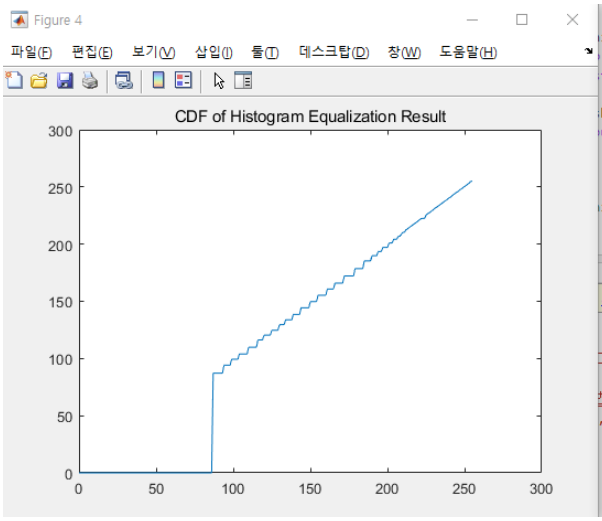
[4] – Custom Image Result

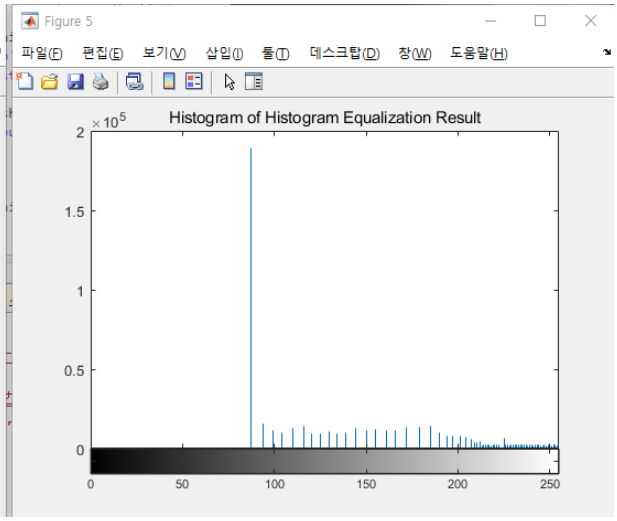
마지막으로 과제에 첨부되어 있지 않은 필자가 직접 새로운 input을 이용한 결과에 대한 것이다. 필자는 유명 축구 클럽 아스날의 전 감독 아르센 벵거의 이미지를 grayscale로 변환하여 입력값으로 사용하였다.

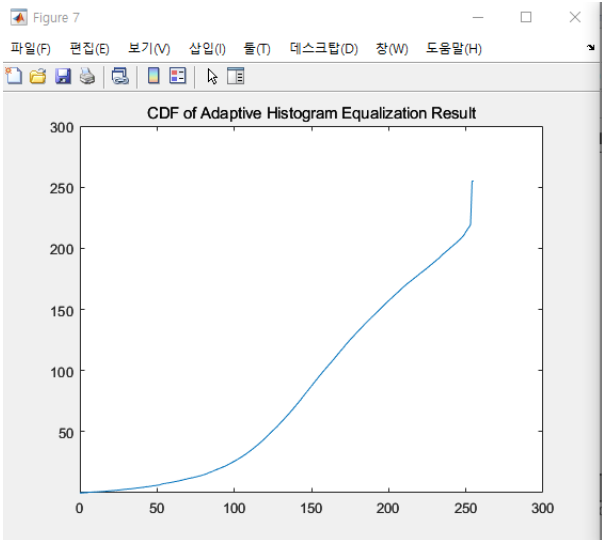
해당 이미지에 대해서 주목할 점은 이후 histogram을 보더라도 알 수 있지만, 전반적으로 매우 어두운 intensity가 대부분이고, 얼굴 부분만 어느 정도 밝은, 매우 low intensity에 치우쳐져 있는 skewed image란 점이다.

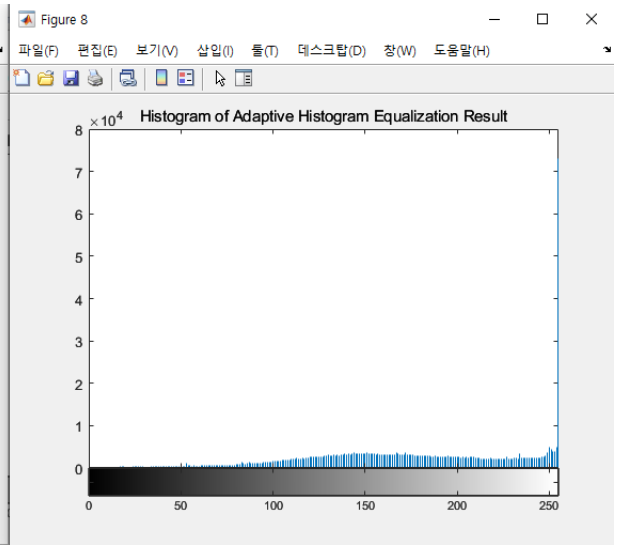












위 이미지는 앞서 설명한 것과 동일한 방식의 코드 실행 결과.

위 이미지는 original, HE, 그리고 AHE에 대해서 numtiles가 각각 10x10, 25x25, 50x50인 결과 이미지이다. HE 결과값을 보면 결국에 전반적인 contrast 대비를 증가시켰지만, original이 low intensity에 skewed 되었었고 특히나 zero-valued intensity가 상당수 비중을 차지했기 때문에 87값의 intensity에 굉장히 쏠려 있음을 알 수 있다. 그리고 원래 밝은 부분인 얼굴 부분에 대해서 전체적인 대비를 키우느라 더 밝아져 overshooting 되듯이 지나치게 밝은 것을 볼 수 있다. 그에 반해, AHE 결과를 본다면 local contrast의 개선으로 얼굴 주름, 머리카락, 상의의 주름 등등의 디테일한 요소들이 훨씬 개선된 점을 찾아볼 수 있다. 특히 numtile의 개수가 커짐에 따라 그러한 점이 더 극단적으로 표현된다. 물론 AHE 단점 그대로 배경 부분에 큰 noise가 증강되는 점도 관

찰할 수 있다.

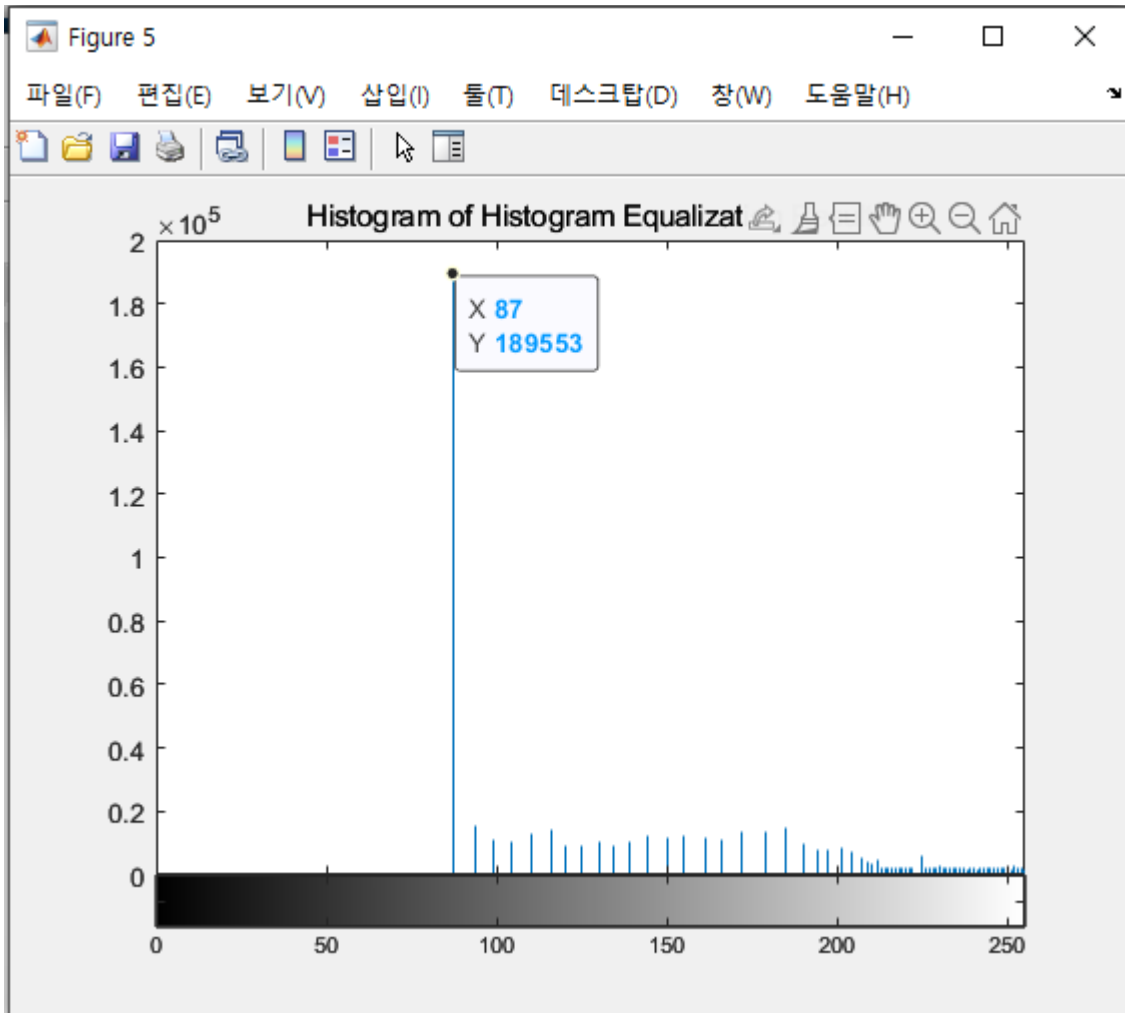


Figure 5
파일(F) 편집(E) 보기(V) 삽입(I) 툴(T) 데스크탑(D) 창(W) 도움말(H)
Histogram of Histogram Equalizat
×10^5
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
X 87
Y 189553
0
50
100
150
200
250