영상처리 9주차 과제

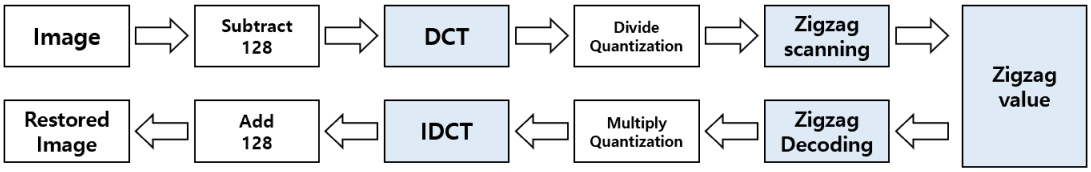
학번: 201404376

이름: 정성욱

1. 과제 내용

목표 JPEG에 대한 decoding구현

Jpeg 압축 과정:



Encoding에서 도출한 zigzag를 복호화하여 배열로 만든후에 인버스 디시티로 값 복원을 한후 128을 더해서 이미지를 복구 시키는 것임

1. **지그제그 스캐닝을 한 리스트를 읽어와서 지그제그 스캐닝을 다시해서 리스트가 되기전의 배열로 복구시킴.**
2. **이 블록 하나하나에 대해 디시티 테이블을 곱해줌**
3. **디시티 테이블을 적용 시킨 블록들에 대해 인버스 디시티를 적용한 한 후에 128을 더한 후 리턴함.**
4. **이 과정을 이미지를 지그제그 스캐닝을 당한 모든 리스트들, 입력 값에 대해 수행한다.**

코드 설명

```
function C=set_C(w,n)
    if(w==0)
        C=sqrt(1/n);
    else
        C=sqrt(2/n);
    end
end
```

set_C는 필터를 만드는데 쓰이는 함수로 매개변수의 값이 0 일경우 sqrt(1/n)를 리턴하고 그외의 경우엔 sqrt(2/n)을 리턴함

$$C(w) = \begin{cases} \sqrt{1/n} & if\ w = 0 \\ \sqrt{2/n} & otherwise \end{cases}$$

```
function res_batch =  layzigzag(i_list)
re_batch=  zeros(8,8);
[lx,ly] = size(i_list);
list = zeros(1,64);
list(1:ly) = i_list;
```

하나의 지그제그 스캐닝 된 리스트를 받아서 8,8 배열로 만들어주는 함수이다.

```
17 -        count = 1;
18 -        idx=1; idy=1;
19 -        mode_change=false;
20 -        while(true)
21 -            if(mode_change~=true)
22 -            if(idx==1&&idy==1)
23 -                    re_batch(idx,idy) = list(count);
24 -                    count = count+1;
25 -                    idy=idy+1;
26 -                    descent = true;
27 -                    accsent = false;
28 -            elseif(descent==true)
29 -                ix=idx;
30 -                for iy = idy:-1:idx
31 -                    re_batch(ix,iy) = list(count);
32 -                    ix=ix+1;
33 -                    count = count+1;
34 -                end
35 -                descent = false;
36 -                accsent = true;
37 -                idx=ix;
```

```
        else
            tmp=idx;
            ix =idx;
            idy=1;
            for iy = idy:idx
                   if(count==37)
                    count+1;
                   mode_change=true;
                    descent = false;
                    accsent = true;
                    idx=8;
                    idy=2;
                    break;
                end

                re_batch(ix,iy) = list(count);
                ix=ix-1;
                count = count+1;
            end
                     if(mode_change) continue
                   end
            descent = true;
            accsent = false;
            tmp=tmp+1;
            idy=tmp;
            idx=1;
        end
```

```
    else
        endw=false;

        if(descent)
        ix=idx;
        idx;
        idy;
        for iy = idy:-1:idx
            re_batch(ix,iy) = list(count);
            count=count+1;
            ix=ix+1;
        end
        descent = false;
        accsent = true;
        tmp = idx;
        idx = idy;
        idy =tmp +1;
    else
        tmp=idx;
        ix =idx;
        for iy = idy:idx
            if(count==64)
                re_batch(ix,iy) = list(count);
                endw=true;
                break;
            end
```

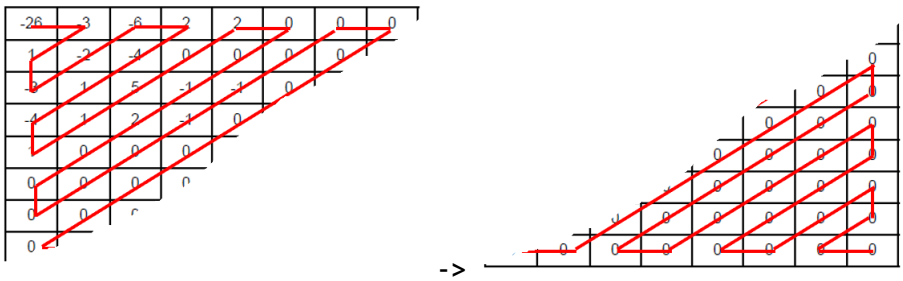
```
91 -                re_batch(ix,iy) = list(count);
92 -                ix=ix-1;
93 -                count = count+1;
94 -            end
95 -            descent = true;
96 -            accsent = false;
97 -            tmp=idy;
98 -            idy =idx;
99 -            idx =tmp+1;
100
101 -            end
102 -        if(endw==true)
103 -            break;
104 -        end
105 -        end
106 -    end
107 -    res_batch = re_batch;
108 -    end
```

여기역시 두가지 모드로 나누고 이 모드안에서 올라가는 모드 내려가는 모드로 나누어두었다.

endw란 조건을 두어 항상 조건 만족시에 탈출 체크를 한다. 이전의 지그제그와의 차이점이라면, 루프를 도는 것이 하나의 리스트를 읽어오고 그 리스트에 대한 처리값을 8x8로 만든 제로들이 가득한 배열에 대해 대입을 한다는 것이다. 이 배열에 대한 인덱싱 처리는

두가지 모드로 나누어서

첫번째일 경우(모드가 false)일 경우는 위



두번째 모드(mode가 true일 경우)는 아래에 따라 정반대적인 시퀀스를 취하고 리스트의 끝부분일 경우 그만둠. 그것을 표시하는 변수가 endw다.

위의 경우는 1x축의 경우 1:2 3:1 1:4 이런식으로 번갈아 가면서 진행하고 1:8일 경우 수행하고

8:2로 초기값을 주고 모드를 변경함. 모드 변경 시 이전과 마찬가지로 2:8 8:3 4:8 이렇게 8:8까지 진행하는데 한가지 차이점이라면 endw가 true면 루프를 중단함

```
function ss = inverse_dct(s_img)
   [px,py] = size(s_img);

   inverse_img = zeros(size(s_img));
   F_uv=zeros(8,8);
   for i = 0:8:px-1
       for j = 0:8:py-1
       F_uv = s_img(i+1:i+8,j+1:j+8);
       this_sum = zeros(8,8);
       for u = 0:7
           for v = 0:7
          this_sum(u+1,v+1)=set_C(u,8)*set_C(v,8)*cos((2*(i)+1)*(u)*pi/(2*8))*cos((2*(j)+1)*(v)*pi/(2*8));
           end
       end
       inverse_img(i+1:i+8,j+1:j+8) =this_sum.*F_uv;
       end
   end


   ss=inverse_img;
end
```

인버스 디시티에 대한 적용이다. 따로 필터가 존재하지않으므로, 위쪽에서 만든 유사필터를 매개변수로 받은 8x8이미지로 바로 곱해서 결과값의 이미지를 순차대로 8의 스트라이드로 떨어지게 해서 채움.

```
DCT_table = ([[16,11,10,16,24,40,51,61];
              [12,12,14,19,26,58,60,55];
              [14,13,16,24,40,57,69,56];
              [14,17,22,29,51,87,80,62];
              [18,22,29,51,87,80,51,62];
              [24,35,55,64,81,104,113,92];
              [49,64,78,87,103,121,120,101];
              [72,92,95,98,112,100,103,99];]);

% Compress Image using portion of JPEG
% zigzag : result of zigzag scanning
% img    : GrayScale Image
[ax,ay] = size(zigzag);
pxy = sqrt(ay/2);
s_img =  zeros(pxy*8,pxy*8);
res = s_img;

for iax = 1:2:ay/2
    tmp = layzigzag(zigzag{iax});
    for ix = 1:8:pxy*8
        for iy= 1:8:pxy*8
           s_img(ix:ix+7,iy:iy+7) = tmp.*DCT_table;
        end
    end
end
```

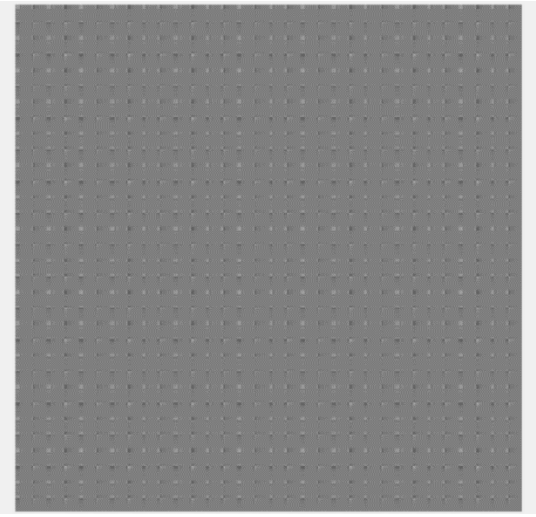
디시티 테이블 정의하고 받은

매개변수 배열에 대해서 배열을 돌면서 tmp값으로 리배치 지그제그 스캐닝으로 리스트에서 지그제그 하기전의 배열을 돌려 받은 다음에, 디시티 테이블을 씌웁니다. 이것을 이미지의 크기만큼 진행하고, 8개의 스텝별로 진행함.

```
end
fxy = inverse_dct(s_img);
    %inverse_dct


% Construct 8x8 blocks using zigzag scanning value

% Multiply Quantization Table


% Apply Inverse DCT


% Add 128

img = fxy+128;
end
```

인버스 디시티를 적용한후 128를 더한 이미지를 반환함.

구현 결과



2. 느낀 점

:: 구현하면서 느낀 점, 어려웠던 점, 혹은 설명이 필요하다고 느낀 부분

디시티역시 실패했기 때문에, 이것 역시 실패하리라 예상하고 있었습니다. 대신 다른 모든 기능은 정상작동 확인 되었습니다.