1. JPEG 인코더 구현

JPEG의 인코더 과정을 통해 이미지의 정보를 압축해서 표현할 수 있다.

1. level shifting

먼저 블록을 8x8 단위로 나눈 후 128만큼 뺀다. 블록의 단위가 커지면 필요한 메모리의 크기가 커지며 압축 화질도 나빠진다. 반면, 블록 크기가 작아지면 전체 코딩 시간이 늘어난다. (DCT, 양자화, entropy coding등의 과정에서) 이 trade-off를 적절히 타협한 크기가 8x8 크기이다.

128만큼 빼는 이유는 0~255 범위를 가지는 화소 값의 범위를 -128~127 범위로 맞추기 위함이다. 일반적인 영상들의 화소 값들은 대부분 중간 값을 가지는데 이러한 값들이 0 근처에 있을 수 있게 level shifting을 하는 것이다.

이 과정을 수행함으로 DCT 변환 시 평균 밝기 값을 낮출 수 있고, 압축 효율을 높일 수 있다.

다음 아래는 직접 level shifting을 구현한 결과이다.

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

왼쪽 값이 원본 값, 오른쪽 값이 level shifting을 한 후의 값이다.

각 항들이 딱 128만큼 차이나는 것을 확인할 수 있다.

1. DCT 변환

DCT 변환은 시간(혹은 공간)영역에 있는 데이터들을 주파수 영역으로 옮기기 위해 실행한다. 주파수 영역으로 공간을 옮기는 이유는 고주파 영역은 사람이 쉽게 인지하지 못하므로 해당 영역의 데이터는 손실시키고, 사람이 잘 인지하는 저주파 영역을 살림으로 압축하기 위함이다.

주파수 변환에 DCT를 채택한 이유는 푸리에 변환의 경우 이산 값 적용에 어려움이 있고 이를 대체한 DFT(Discrete Fourier Transform)은 실수 항과 허수 항 두 부분이 존재해 연산량이 급격히 늘어나는 성향을 보인다. 따라서 이 경우를 보완하는 DCT를 사용한다. 이는 많은 표준 기술에서 사용한다.

아래는 DCT를 수행한 결과인데, level shifting 여부에 따라 값들이 어떻게 변하는지 살펴보자.

텍스트, 칠판이(가) 표시된 사진

자동 생성된 설명텍스트, 칠판이(가) 표시된 사진

자동 생성된 설명

왼쪽이 원본 이미지에 DCT한 결과, 오른쪽은 level shifting을 한 결과에 DCT를 한 결과이다. DC계수인 첫 성분이 눈에 띄게 값이 줄어들었다는 것을 알 수 있다. 이렇게 level shifting을 함으로써 DC 계수가 0에 더 가까워졌기 때문에 양자화 에러를 줄일 수 있고 압축률도 높일 수 있다.

한가지 신기한 사실은 DC 계수를 제외한 다른 계수들은 변화가 없다는 점이다.

1. 양자화

DCT를 해서 얻은 결과를 특정 값들로 나누어 주는 과정을 양자화라고 한다. 양자화를 한 결과로 우리는 입력했던 블록을 특정 값들로 나눈 몫을 얻는다. 아래는 JPEG에서 밝기 신호에 사용하는 양자화 테이블이다.

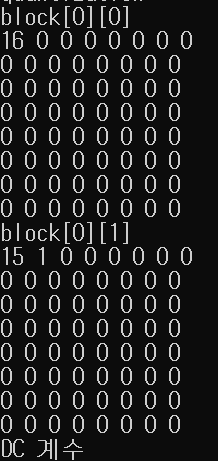
테이블이(가) 표시된 사진

자동 생성된 설명

이 때 한 블록에서 위치마다 나누는 값들이 달라진다. 주로 저주파 성분 (왼쪽 상단 영역)은 작은 값들로 나누어 값을 최대한 보존하고 고주파 성분은 높은 값으로 나누어 정보를 많이 잃게 된다.

그 결과, 왼쪽 상단은 0이 아닌 값들이 존재하지만, 고주파 영역은 대부분 0이 차지하게 된다.

아래는 DCT한 블록을 양자화 한 결과이다.



이렇듯 거의 0이 대부분 차지하게 된다. 또한 몫만 챙기므로 정수 값이 출력된다.

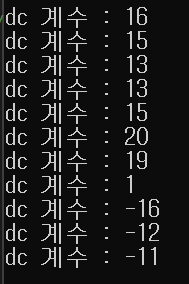
1. DPCM

양자화까지 끝마치면 남은 단계들은 양자화한 결과를 어떻게 최대한 압축해서, 효율적으로 저장할 수 있을까?에 관한 과정들이다.

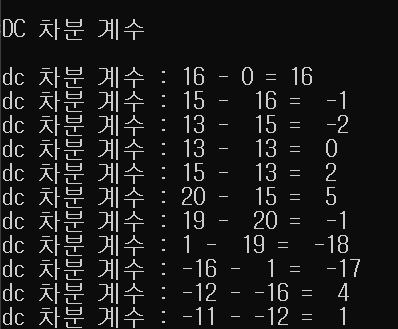
먼저 양자화한 결과에서 각 블록의 [0][0]인 DC계수를 따로 저장한다. 이 때, DPCM이라는 방법을 사용하는데, 맨 처음 저장하는 DC계수는 그대로 저장을 하고, 그 다음 값부터는 자기 자신 – 이전 DC 계수 의 계산을 수행해서 그 값을 저장한다.

이런 과정을 거치면 작은 값들을 정보를 부호화함으로써 더 효과적으로 압축을 수행할 수 있다.

아래는 직접 프로그래밍 한 과정이다.



DC 계수의 일부를 순서대로 프린트하면 위와 같다. 우리는 이 값들의 차분 값을 저장한다.



모든 값들이 한 자리수로 표현되는, 0과 가까운 값은 아니지만, 비교적 작은 값들을 이용해 DC 계수를 저장할 수 있게 되었다.

1. zigzag scan

DC 계수를 제외한 나머지 AC 계수들을 저장하기 위해서는 zigzag scan을 거쳐야 한다.

zigzag scan을 사용하는 이유는 고주파 영역에서 반복되는 0을 효과적으로 저장하기 위함이다. 반복되는 0을 run length code를 이용해 부호화 하면 더 적은 bit를 사용해 저장할 수 있을 것이다.

zigzag scan은 다음과 같은 순서로 block 내 data를 가져오며, 그 결과를 1차원 배열로 이해하면 쉽다.

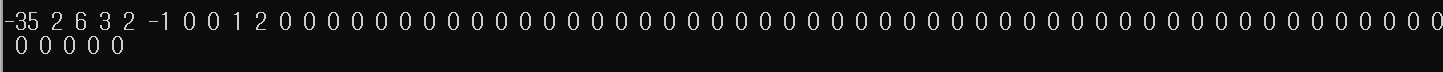
텍스트, 키보드이(가) 표시된 사진

자동 생성된 설명

직접 coding한 결과는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위 블록을 zigzag scan을 해서 아래의 1차원 데이터로 저장했다. 0이 매우 반복되는 양상이기에 run length code를 통해 부호화하기에 매우 적당해졌다.

1. run length code

dc 계수를 DPCM을 통해 효율적으로 저장했다면 ac 계수는 run length code를 통해 효율적으로 저장할 수 있다.

저장 형태는 (건너 뛸 0의 수, 0이 아닌 값)으로 이루어지며, (0, 0)이 해당 블록의 마지막을 나타낸다. 주의할 점은 DC 계수는 저장하지 않는 것이다.

위의 block[32][32]를 부호화하면 다음과 같을 것이다.

(0, 2) (0, 6) (0, 3), (0, 2), (0, -1) (2, 1), (0, 2), (0, 0)

1. JPEG 디코더 구현

디코더는 인코더의 역순이며 우리가 용량을 줄이기 위해 인코더를 통해 압축해 놓은 정보들을 해석해서 이미지로 표현하는 단계이다.

1. run length code 해석

run length code를 사용해 ac 계수를 저장하기 위해 사용한 (건너 뛸 0의 수, 0이 아닌 값)의 정의를 이용해 해석한다.

(0, 0)은 블록의 끝을 나타내므로 만약 블록 내부에 채워야 하는 공간이 남아 있다면 모두 0으로 채우면 된다.

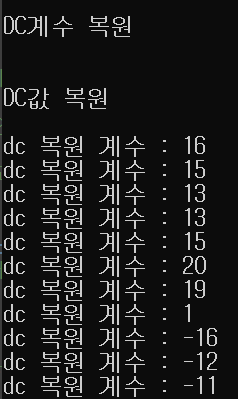
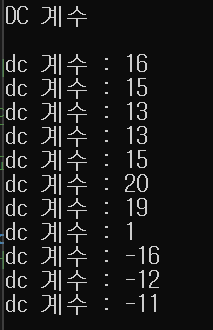
해당 부분은 계속된 에러의 반복으로 구현하지 못했다.

1. DC 계수 복원

DPCM을 통해 저장한 DC 계수도 복원해준다.

저장된 값 = 현재 DC 계수 – 이전 DC 계수이고, 우리가 구하고자 하는 DC 값은 현재 DC 계수이므로 현재 DC 계수 = 저장된 값 + 이전 DC 계수를 계산하면 구할 수 있다. 이 때, DPCM을 통해 저장된 DC 계수의 첫 항은 원본 값이므로 위의 계산이 가능하다.

직접 구현한 결과는 아래와 같다.



왼쪽 이미지는 인코더 과정에서 프린트한 DC 계수이고 오른쪽 이미지는 디코더 과정에서 복원한 DC 계수이다. DPCM의 과정 자체가 어렵지 않고, 이를 반대로만 수행하면 되는 과정이라 어렵지 않다.

1. DC 계수와 AC 계수를 block 이미지로 전환

위의 과정을 통해 우리는 DC 계수와 AC 정보를 따로 따로 가지고 있다. DC 계수 뒤에 AC 계수 데이터 63개를 이어주는 작업을 반복하면 인코더 과정에서 우리가 구한 zigzag scan의 결과이다.

zigzag scan을 원래 블록 데이터의 배열로 바꾸는 법 역시 인코더 과정의 역순이다. 반복문은 그대로 사용하고, 저장할 값, 저장하는 값의 위치만 바꾸어 주면 손쉽게 구현이 가능하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위 이미지는 직접 구현한 결과를 print한 결과로, 이 block들은 인코더에서 우리가 진행한 양자화 후 얻은 행렬과 같아진다.

1. IDCT

다음은 IDCT를 수행한다. 하지만 우리는 인코더 과정에서 DCT를 수행하기 전에 양자화 과정에서 특정 값들로 나누기 연산을 수행했다.

따라서 IDCT 과정에서 그 값들을 다시 곱하는 곱하기 연산을 수행한다.

이렇게 되면, 전체적으로 보았을 때 우리가 인코더 과정에서 양자화를 함으로써 양자화 과정에서 나누어 준 나머지들을 잃게 된다. 이 과정에서 데이터의 손실이 일어나게 되며, 이는 복원할 수 없는 값들이다.

전체적으로 디코더를 완벽하게 인코더의 역순으로 구현하려면 양자화의 역 연산을 먼저 한 후에 IDCT를 하는 것이 맞지만, 이전에 구현한 함수를 다시 재사용하기 위해 이를 한 번에 수행했다.

한 편, 이 과정을 거쳐 나온 결과는 인코더에서 DCT를 하기 전의 block과는 값이 다르다. 이유는 앞서 말했듯이 나머지 정보를 잃어버렸기 때문이다.

텍스트이(가) 표시된 사진

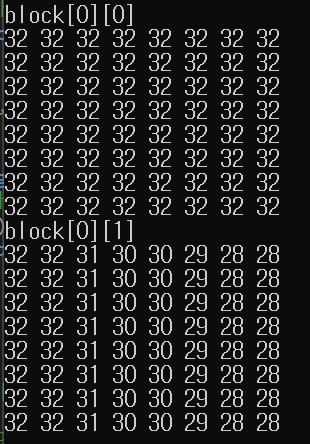
자동 생성된 설명테이블이(가) 표시된 사진

자동 생성된 설명

왼쪽은 인코더에서 DCT를 하기 전 행렬, 오른쪽은 디코더 과정에서 IDCT를 한 행렬이다. 양자화로 인해 정보가 손실되었기에 서로 값이 다르고, 완벽하게 복원되지 않았다.

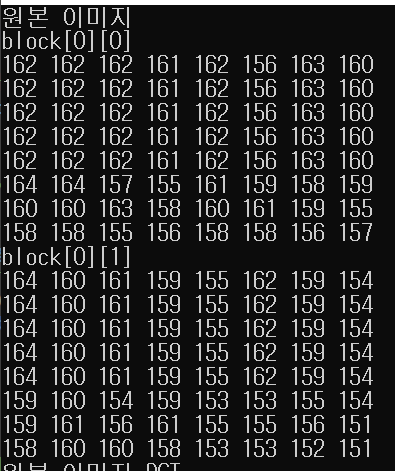
1. level shifting

마지막으로 level shiting이다. 인코더 과정에서 시작하면서 128을 뺐으므로 디코더에서는 마지막을 128로 더하면서 마무리한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

각각 level shifting을 하기 전 후 행렬이다. 이로써 복원이 완료되었다. 실제 원본 이미지 데이터 값은 아래와 같다.



1. 결과

사람, 여자, 모자이(가) 표시된 사진

자동 생성된 설명사람, 여자, 모자, 하얀색이(가) 표시된 사진

자동 생성된 설명

왼쪽이 원본, 오른쪽이 복원 이미지이다. 비교적 날카로웠던 모자의 털 부분이 부드러워졌고, 밝기도 어느정도 어두워진 느낌이다.

한 편, 이렇게 JPEG의 손실압축 과정을 거친 이미지의 MSE와 PSNR은 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. github에서 jpeg codec 프로그램 다운받아서 실행

먼저, 해당 코드 제작자가 예시로 든 이미지를 실행해 보았다.

개, 잔디, 앉아있는, 포유류이(가) 표시된 사진

자동 생성된 설명앉아있는, 개, 포유류, 실내이(가) 표시된 사진

자동 생성된 설명

왼쪽이 원본, 오른쪽이 jpeg의 압축 과정을 거쳐 복원된 이미지이다. 눈으로 보기에도 비교적 원본이 많이 훼손되었음을 알 수 있다. 아마 양자화 과정에서 저주파 영역도 큰 값으로 나누어 주었을 것 같다. 내 생각이 사실인지 확인하기 위해 코드를 살펴보았는데 그 값은 책에서 제시한 값과 같았다.

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

아마 사진을 보여주는 프로그램이 해석하는 과정에서 차이를 일으켰을 수도 있다고 생각한다. 왼쪽은 jpg확장자인 반면 오른쪽은 BMP 확장자이다.

또, 다시 생각해보면 우리가 입력으로 넣어준 이미지는 jpg로 이미 압축과정에서 손실을 겪었다. 이를 다시 jpeg의 인코더 디코더 과정을 거치므로 더 많은 정보들이 손실되어 빛이 바랜 이미지가 나온 것이 아닐까 예상이 된다.



여기서 에러는 MSE이다. 작성자가 원본과 예측 값의 차를 제곱하지 않은 부분을 수정했다.

내가 직접 사진을 찍어서 이 프로그램에 적용하고 싶었지만 1024정도의 가로 길이/세로 길이가 넘어가면 에러가 일어나는 것 같았다.

따라서 직접 찍은 사진을 아이폰에서 제공하는 메일 첨부 시 중간 용량으로 압축해서 보내주는 기능을 사용해 얻은 이미지를 사용해보았다. (즉, 이미 해상도 변경으로 인한 압축이 한 번 진행된 이미지..)

텍스트, 실내이(가) 표시된 사진

자동 생성된 설명텍스트, 실내이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

전체적으로 및이 많이 바랜 모습을 볼 수 있다. 개인적으로 더 초록빛이 많이 도는 것 처럼 느껴졌다.

작성자가 예시로 둔 이미지 몇 개를 더 돌려보았다.

포유류, 영장류, 원숭이, 밝은이(가) 표시된 사진

자동 생성된 설명원숭이, 영장류, 눈이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 의류, 여자이(가) 표시된 사진

자동 생성된 설명텍스트, 사람, 여자, 머리카락이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트, 패브릭이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

이미지의 크기가 클수록 MSE가 높게 측정되었으며 원본 이미지에 비해 예측 이미지가 살짝 어두운 느낌과 빛이 바랜 느낌이 들었다.

흑백 이미지에서는 압축 과정에서 색차 신호가 어떻게 변하는지 느낄 수 없었지만, github에서 다운받은 프로그램으로 색차 신호가 어떻게 변하는지에 대한 부분까지 볼 수 있었다.

github 주소 : https://github.com/camilanovaes/jpeg-codec

1. Discussion

이번 과제는 마치 이전에 했던 과제들이 왜 필요한지 알려주는 과제였다.

DCT, 양자화 과정을 통해 불필요한 정보는 날리고, 그 데이터들을 RLC, 허프만 코드 등을 통해 효과적으로 표현할 수 있고, 이 과정은 jpeg에서 사용하는 방법이며, 그동안 배웠던 내용들이 모두 적용되어 있었다.

한 편, 블록의 크기가 8이 된 이유가 메모리 사용량/해상도와 연산량의 trade-off 때문인 것, level shifting을 통해 이미지의 값들을 더 0에 가깝게 배치함으로 entropy coding 과정에서 더 압축률이 높아지도록 하는 것, DPCM, zigzag scan 등에 대해 더 배울 수 있었다.