Application of Discrete Sine Transform in Image Processing

Nitesh Agarwal
Department of Computer Science
Jodhpur Institute of Engineering & Technology
Jodhpur, India

Rahul Solanki
Department of Mathematics
Jodhpur Institute of Engineering & Technology
Jodhpur, India

Dr. A.M. Khan
Department Of Mathematics
Jodhpur Institute of Engineering & Technology
Jodhpur, India

***Abstract*— Digital devices & computational resources have limited communication & storage capabilities. Because of these limitation digital multimedia data need to compress. For example if there are multiple camera are connected in a network for communication then images captured by camera need to compress before communication to use proper bandwidth & for synchronized communication. Similarly local computer system has limited storage capacity & uncompressed multimedia data required high volume of storage to store. Hence multimedia data like images, videos need to be compress for many purpose. Image compression process use two technique to compress image lossless image compression & lossy image compression. Lossy Image compression needs some transformation like DCT, DFT, KLT, DST etc. Purpose of transformation is to convert the data into a form where compression is easier. This transformation will transform the pixels which are correlated into a representation where they are decorrelated. The new values are usually smaller on average than the original values. The net effect is to reduce the redundancy of representation. In Lossy image compression input image is divided in to 8*8 blocks & then each pixel is converted in to its equivalent frequency value using various transformation like DCT, DST etc. The present paper deals with the study of transformation of an 8 bit (b/w) image into its frequency domain through Discrete Sine Transform.**

Key Words: DCT, DST, KLT, DFT

1. Introduction

An image basically a 2D signal processed by the human visual system. The signal representing images are usually in analog form, but for processing, storage, transmission and computing by computer application analog images are converted from analog to digital form. Discrete Sine Transform (DST) converts this digital information into its equivalent frequency domain by partitioning image pixel matrix into blocks of size N*N, N depends upon the type of image. For example if we used a black & white image of 8 bit then all shading of black & white color can be expressed into 8 bit hence we use N=8, similarly for color image of 24 bit we can use N=24 but using block size N=24, time complexity may increase. Hence we operate DST on individual color component for a color image. Color image consist of 8 bit red + 8 bit green + 8 bit blue hence we apply DST on each color component (Red, Green, Blue) using block size N=8. DST give best result if block size is 8*8 or 16*16 hence block size is used in this way that it fulfill the constraint of DST as well constraint of pixel format of image.

1.1 One-Dimensional DST

If we have one-D sequence of signal value of length N then its equivalent DST can be expressed as

$$s(u)=\alpha(u)\sum_{x=0}^{N-1} f(x)\sin\left[\frac{\pi(2x+1)(u+1)}{2N}\right] \quad (1)$$

for $u = 0,1,2,\ldots,N-1$.

& inverse transformation is defined as

$$f(x)=\sum_{u=0}^{N-1}\alpha(u)s(u)\sin\left[\frac{\pi(2x+1)(u+1)}{2N}\right] \quad (2)$$

Where $f(x)$ is signal value at point x & $\alpha(u)$ is transform coefficient for value u.

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & for\ u = N-1 \\ \sqrt{\frac{2}{N}} & for\ u \neq N-1 \end{cases} \qquad (3)$$

It is clear from (1) for u=0,

$$s(u=0) = 0 \qquad (4)$$

1.2 Two – Dimensional DST

An image is 2-D pixel matrix where each position (i,j) represents a color value for that particular point or position. Hence to transform an image into its equivalent DST matrix we use 2-D DST.

2-D DST can be defined as

$$s(u,v)=\alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\sin\left[\frac{\pi(2x+1)(u+1)}{2N}\right]\sin\left[\frac{\pi(2y+1)(v+1)}{2N}\right] \qquad (5)$$

for $u, v = 0,1,2,\ldots,N-1$.

& inverse transformation is defined as

$$f(x,y)=\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\alpha(u)\alpha(v)s(u,v)\sin\left[\frac{\pi(2x+1)(u+1)}{2N}\right]\sin\left[\frac{\pi(2y+1)(v+1)}{2N}\right] \qquad (6)$$

Where $s(u,v)$ represents frequency value for u, v & $f(x,y)$ represents pixel color value at position (x, y).

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & for\ u = N-1 \\ \sqrt{\frac{2}{N}} & for\ u \neq N-1 \end{cases} \qquad (7)$$

$$\alpha(v) = \begin{cases} \sqrt{\frac{1}{N}} & for\ \ v = N-1 \\ \sqrt{\frac{2}{N}} & for\ v \neq N-1 \end{cases} \qquad (8)$$

2. Main Results

2.1 Implementation of DST

This paper describe how a b/w image is convert into equivalent frequency domain using DST.

Steps involved in this implementation

1. Create pixel matrix of the image & divided it into blocks of size 8*8
2. Apply FDST (Forward Discrete Sine Transform) on each 8*8 block of pixel matrix to get equivalent 8*8 DST blocks.
3. To get Original image we apply IDST (Inverse Discrete Sine Transform) on each 8*8 block DST & get its equivalent 8*8 IDST block.
4. Using 8*8 IDST blocks we create modified pixel matrix to get modified image.
5. Now we Find MSE (Mean Squared Error) & PSNR (Peak Signal To Noise Ratio) to determine quality of image obtain by IDST. MSE & PSNR calculated by following formulas

$$MSE = \frac{1}{H*W}\sum_{x=0}^{H-1}\sum_{y=0}^{W-1}[o(x,y)-m(x,y)]^2 \qquad (9)$$

$$PSNR=20*\log_{10}(MAX) - 10*\log_{10}(MSE) \qquad (10)$$

Where H=Height of Image, W= Width of Image, variable MAX shows max value of a pixel for example if image is 8 bit then MAX=255.

Quality of image obtain by IDST is depend on MSE & PSNR value. If as the MSE value increases PSNR value decreases then we get a bad quality of image by IDST & if as the MSE value decreases PSNR value increases we get a batter quality image hence a best suitable transformation like DCT, DST, DFT is taken on the basis of this MSE & PSNR value.

2.1.1 Algorithm 1

```
Get_8*8_blocks (image)
{
n=8, k=0; width=width
of image;
height=height of image;
for ( i=0;i < width/n; i++)
 {
for ( j=0; j < height/n; j++)
  {
 xpos = i * n;
 ypos = j * n;
   for ( a=0; a < n; a++)
     {
    for ( b=0; b < n; b++)
      { color = color at position(xpos+a, ypos+b);
        block[k][a][b]=color-128;
      } //end of for loop b
```

```
            k=k+1;
        } // end of for loop a }
      // end of for loop j
    } // end of for loop i }// end
of Get_8*8_blocks
```

2.1.2 Algorithm 2

```
FDST (block [] [] [])
{ width=width of image, N=8;
  height=height of image;
  q=(width/8)*(height/8)
  for ( i=0;i < q; i++)
  { for ( u=0; u< N; u++)
    { for ( v=0; v < N; v++) { if (u==N-1) {
          $\alpha(u) = \sqrt{\frac{1}{N}}$
        }
        else{
          $\alpha(u) = \sqrt{\frac{2}{N}}$
        }
        if (v==N-1){
          $\alpha(v) = \sqrt{\frac{1}{N}}$
        }
        else {
          $\alpha(v) = \sqrt{\frac{2}{N}}$
        }
      sum=0;
      for( x=0;x<N;x++)
        { for( y=0;y<N;y++)
          { sum= sum + block[i][x][y]* $\sin\left[\frac{\pi(2x+1)(u+1)}{2N}\right]$* $\sin\left[\frac{\pi(2y+1)(v+1)}{2N}\right]$ ;
          } // end of for loop y } // end of for loop x
      dst[i][j][k]=( $\alpha(u)$ *$\alpha(v)$ *sum);
    } // end of for loop v
  } // end of for loop u
} // end of for loop i
}//end of FDST
```

2.1.3 Algorithm 3

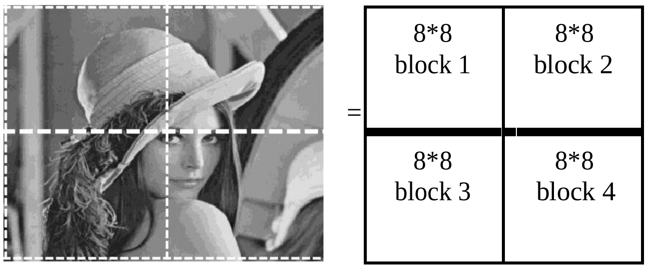
```
IDST(dst [] [] [])
{ width=width of image, N=8;
  height=height of image;
  q=(width/8)*(height/8)
  for ( i=0;i < q; i++)
  { for ( x=0; x< N; x++) {
    for ( y=0; y< N; y++)
      {
      sum=0;
      for( u=0;u<N;u++)
        { for( v=0;b<N;v++)
          { if (u==N-1) {
              $\alpha(u) = \sqrt{\frac{1}{N}}$
            }
          else{
              $\alpha(u) = \sqrt{\frac{2}{N}}$
            }
          if (v==N-1){
              $\alpha(v) = \sqrt{\frac{1}{N}}$
            }
          else {
              $\alpha(v) = \sqrt{\frac{2}{N}}$
            }
sum= sum + $\alpha(u) * \alpha(v)$ *dst[i][u][v] *
$\sin\left[\frac{\pi(2x+1)(u+1)}{2N}\right] * \sin\left[\frac{\pi(2y+1)(v+1)}{2N}\right]$;
            } // end of for loop
          v } // end of for loop u
            idst[i][j][k]=sum;
          } // end of for loop y }
            // end of for loop x
        } // end of for loop i }
    // end of IDST
```

2.1.4 Algorithm 4

```
Get_Image(idst [] []
[]) { k=0;
width=width of image;
height=height of image;
   for ( i=0; i < width; i++) {
      for ( j=0; j < height; j++) {
            xpos = i * n;
            ypos = j * n;
         for (a=0; a < n; a++)
            { for (b=0; b < n; b++)
               {
         color=(int)idst[k][a][b];
         set color at position (xpos+a, ypos+b);
                  }// end of loop b
            } // end of loop a
               k++;
         } // end of loop j
      } // end of loop i
   }// end of Get_Image
```

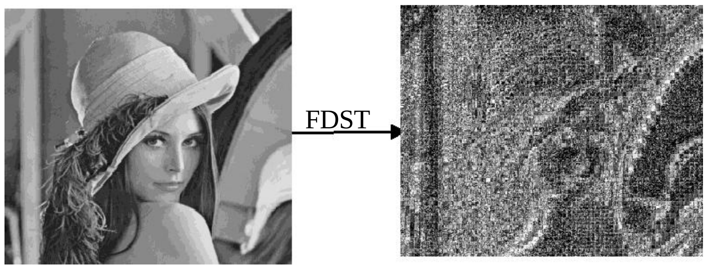
2.2 Outputs

1. Convert pixel matrix into blocks of size 8*8



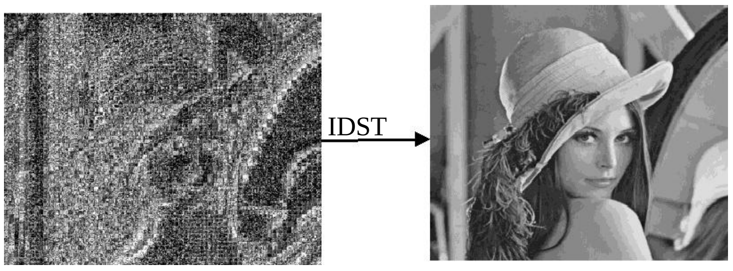
Input Image of size 16*16 Output blocks of size 8*8

2. Transform Input image into equivalent DST image



Input Image of size 16*16 Output DST Image of size 16*16

3. Get modified image from DST image

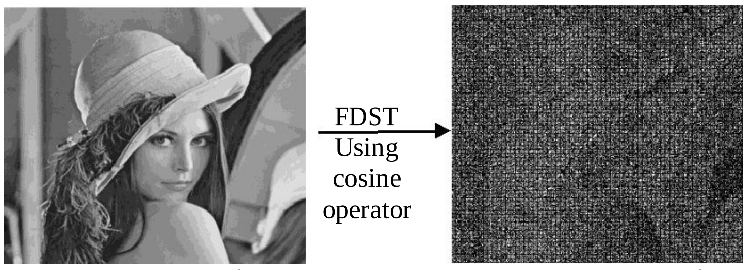


Input DCT Image of size 16*16 Output Image of size 16*16

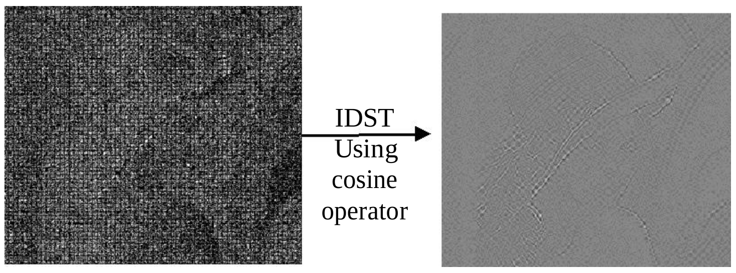
3. Modification in original DST

3.1 Using cosine operator rather than sine

There is a difference of π/2 between sine & cosine operator hence using cosine rather than sine operator in DST may loss pixel data



Input Image of size 16*16 Output DST Image of size 16*16



Input Image of size 16*16 Output Image of size 16*16

3.2 Change in block size

All shading of black & white image can be expressed in 8 bit of blocks hence we use block size 8*8 to perform DST on it. But in color image each color value of a pixel can be expressed into 24 bit of block which contain 8 bit red + 8 bit green + 8 bit blue. To transform a color image into its equivalent DST format we extract each 8 bit color component from 24 bit of block & then perform 8*8 DST on each color component rather than using 24*24 DST for 24 bit block. The main reason is that if use 24*24 DST rather than 8*8 DST the time complexity of DST is increases in a very large amount.

For example

For an image of size 48*48

1. If 8*8 DST used

Total no of blocks q=(48/8)*(48/8)=36 For FDST

```
for ( i=0;i < q; i++) // loop runs 36 times
  { for ( u=0; u< 8; u++) // loop runs 36*8 times
    { for ( v=0; v < 8; v++)// loop runs 36*8*8 times
        {

        }
      for( x=0;x<8;x++)// loop runs 36*8*8*8 times
        {   for( y=0;y<8;y++) // loop runs 36*8*8*8*8
  times
            {
                } // end of for loop
            y } // end of for loop x
            } // end of for loop v
          } // end of for loop u
        } // end of for loop i
```

Total no. of iteration = 36*8*8*8*8= 147456

2. If 24*24 DST used

Total no of blocks q=(48/24)*(48/24)=4 For FDST

```
for ( i=0;i < q; i++) // loop runs 4 times
  { for ( u=0; u< 24; u++) // loop runs 4*24 times
    { for ( v=0; v < 24; v++)// loop runs 4*24*24 times
        {
                  }
      for( x=0;x<24;x++)// loop runs 4*24*24*24 times
        {
      for( y=0;y<24;y++) // loop runs 4*24*24*24*24 times
                  {
                    } // end of for loop
                y } // end of for loop x
                } // end of for loop v
              } // end of for loop u
            } // end of for loop i
```

Total no. of iteration =4*24*24*24*24= 1327104

Hence 24 * 24 DST required 1327104-147456=1179648 extra iteration to preform DST which increases time complexity in large amount hence DST used with block size 8*8.

3.3 MSE & PSNR

Following table gives a comparative analysis of quality of transformed image using DST, DST with cosine operator & original DCT.

	MSE	PSNR
2D DST	0.38	52.38
2DDST with cosine	2239.17	14.63
2D DCT	0.28	53.64

Table 1: MSE & PSNR value of input image after transformation

As the table show 2D DCT has minimum MSE & maximum PSNR value hence it is best transformation technique & we cannot use DST with cosine operator in normal way because it gives an image with high amount of noises in the pixel of image.

4. Conclusion

The result presented in this document shows that

1. It is very easy to implement DST rather than other transformation on image except DCT.

2. If DST used with cosine operator rather than sin pixel data may lose.

3. If DST used with block size 24*24 rather than block size 8*8 then time complexity of DST is increases in very large amount.

REFERENCES

[1] Andrew B. Watson, “Image Compression Using Discrete Cosine Transform”, NASA Ames Research Centre, 4(1), pp. 81-88, 1994.

[2] Anjali Kapoor and Dr. Renu Dhir, “Image Compression Using Fast 2-D DCT Technique”, International Journal on

Computer Science and Engineering (IJCSE), vol. 3 pp. 2415-2419, 6 June 2011.

[3] Harley R. Myler and Authur R. Weeks “The Pocket Handbook of Image Processing Algorithms in C”, ISBN

0-13-642240-3 Prentice Hall P T R Englewood Cliffs, New Jercy 07632

[4] Iain E.G. Richardson “H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia”, ISBN 0470848375, 9780470848371, Wiley,2003.

[5] L.Dhang, W. Dong D.Zhang and G.Shi “Two stage image denoising by principal component analysis with local pixel grouping” Pattern Recognition, Vol.43, pp1531-1549, 2010.

[6] Maneesha Gupta and Dr.Amit Kumar Garg, “Analysis Of Image Compression Algorithm Using DCT” International

Journal of Engineering Research and Applications (IJERA), vol.2, pp. 515-521, Jan-Feb 2012.

[7] N.Ahmed, T.Natatarajan, and K.R. Rao, “Discrete Cosine Transform”, IEEE Transactions on Computers, vol. C-32, pp. 90-93, Jan. 1974.

[8] S. Malini. & R.S. Moni. ”Use of Discrete Sine Transform for A Noval Image Denoising Technique”. International Journal of Image Processing(IJIP), Vol, 8, Issue 4, pp. 204-213, 2014.

[9] Swati Dhamija and Priyanka Jain “Comparative Analysis for Discrete Sine Transform as a suitable method for noise estimation” IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 5, No 3pp. 162-164, September 2011.

[10] V.P.S.Naidu, “Discrete cosine Transform based Image Fusion”, Defence Science Journal, Vol.60, No.1, pp.48-54., Jan.2010.