GÖRÜNTÜ İŞLEME - (6.Hafta)

GÖRÜNTÜ NETLEŞTİRME ALGORİTMALARI

NETLEŞTİRME/KESKİNLEŞTİRME FİLTRESİ (Sharpening Filter)

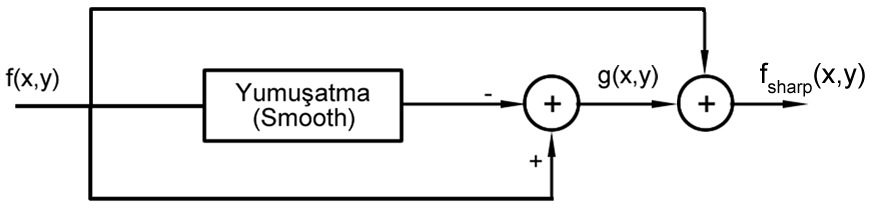
Bu algoritma orjinal görüntüden, görüntünü yumuşatılmış halini çıkararak belirgin kenarların görüntüsünü ortaya çıkarır. Daha sonra orjinal görüntü ile belirginleşmiş kenarların görüntüsünü birleştirerek, kenarları keskinleşimiş görüntüyü (netleşmiş görüntü) elde eder. Görüntünün netleştirilmesi fotogrametri ve baskı endüstrisinde yaygın olarak kullanılır.

g(x,y) çıkış görüntüsü, f(x,y) giriş görüntüsü ve $f_{smooth}(x,y)$ de bu görünün yumuşatılmış hali olmak üzere bu işlem aşağıdaki şekilde gösterilebilir.

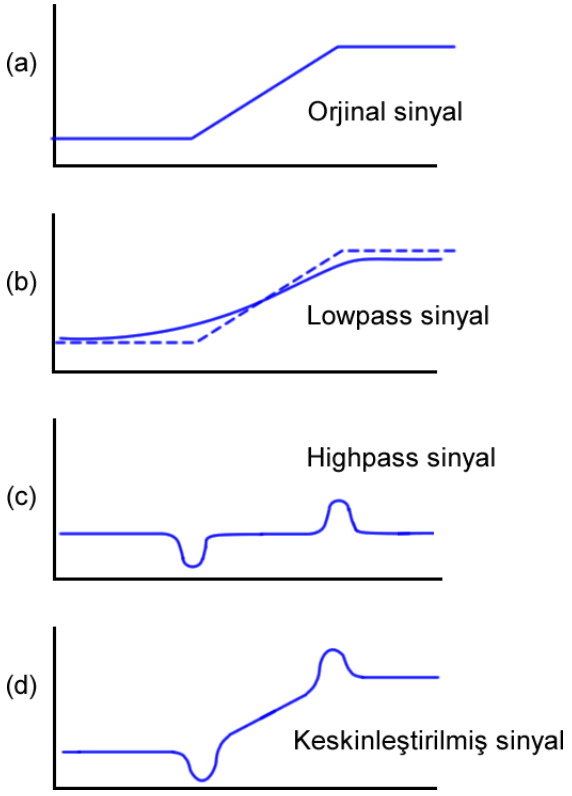
$$g(x,y) = f(x,y) - f_{smooth}(x,y)$$

$$f_{sharp}(x,y) = f(x,y) - k * g(x,y)$$

Burada k bir ölçekleme sabitidir. k için makul değerler 0,2-0,7 arasında değişir. k büyüdükçe keskinleştirme miktarı artar.



Filtrenin frekans cevabını inceleyerek işlemi daha iyi anlayabiliriz. Eğer bizim elimizde aşağdaki gibi bir sinyal varsa (a) bu sinyalden lowpass filtreleme yaparak (kenarlar yumuşatılmış olur) (b) ortaya highpass görüntü (keskin kenarlar) çıkar (c). Bu keskin kenarlar, orjinal görüntü ile birleştirilirse netleştirilmiş (kenarları keskinlşetirilmiş) görüntü oluşur (d).



Şekil. Keskinleştirme filtresi için frekans cevabı ile kenar görüntüsü hesaplama

Keskinleştirme filtresi konvolüsyon işlemi ile yapılır. Yani çekirdek bir matris kullanılarak, resim alanı üzerinde gezdirilir ve gerekli çarpma işlemleri ile yeni resim elde edilir. Görüntünün yumuşatılmış versiyonunu oluşturmak için 3x3 lük Ortalama filtresi (mean filter) kullanılabilir. Ardından yumuşatılmış görüntü, orjinal görüntüden piksel değerleri üzerinden çıkarılır. Bu esnada ortaya çıkan değerler negatif yada 255 i geçebilir. Bu durumda değerleri normalize etmek gerekir.

Değerleri normalize etme: Matematiksel işlemler sonunda görüntüdeki R,G,B değeğerleri 0-255 değerlerinin dışına çıkarsa tüm resmi aynı orantıya tabi tutarak değerleri yeniden normalize etmek gerekir.

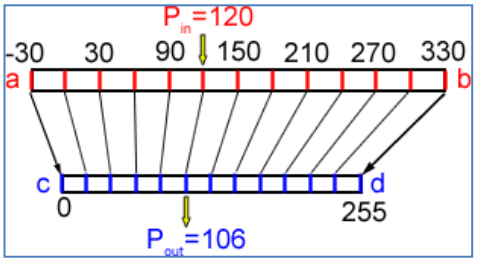
Resim içerisindeki limit değerleri a,b ile, bu değerlerin olması gereken aralık olan 0,255 sayılarını da c,d ile, normalize edilecek değeri Pin, normalize olmuş halinide Pout ile gösterirsek, bu işlemi aşağıdaki şekilde formülüze edebiliriz. Dikkat edilirse bu formül verilen aralıklarda değerleri orantı kurarak bir gerdirme işlemidir.

$$\frac{(b-a)}{(d-c)} = \frac{(P_{in}-a)}{(P_{out}-c)} \rightarrow \boxed{P_{out} = (P_{in}-a)\left(\frac{d-c}{b-a}\right)+c}$$

Örneklersek;

$$P_{out} = (120-(-30))\left(\frac{255-0}{330-(-30)}\right)+0 = 106$$

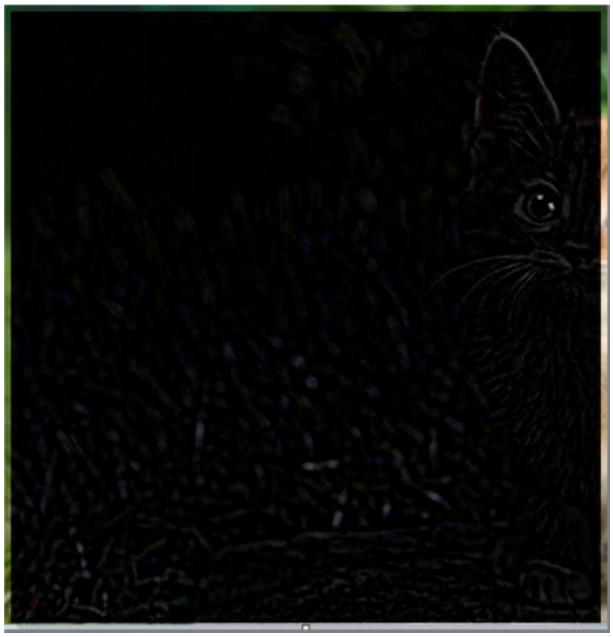
Not: Buradaki Üst ve alt sınırlar 0 ve 255 değerlerini aşmaz ise d ve c sayıları 0 ve 255 alınamaz. Daha içte kalan mevcut değerler (b yada a) alınmalıdır. Diğer türlü resim hiç beyaz yada siyah bölge yokken bu bölgeler ortaya çıkmaya başlar.



Programlama (Görüntü Netleştirme)

Orjinal Resim

Mean 9x9 matris ile Bulanıklaştırma



Kenar Görüntüsü

Mean 9x9 matris ile netleşmiş Görüntü

Orjinal Resim

Gauss (5x5) matrisi ile netleştirilmiş görüntü

Burada orjinal resim daha başlangıçta bulanık ise, kullanılan matrisin boyutu artırılmalıdır. Gauss işlemi daha hassas bulanıklaştırma yaptığından etkisi, ince resimlerde daha iyi gözükecektir.

Not: Aşağıdaki kodlarda normalleştirme limit değerleri kullanarak yapılmadı. Basit düzeyde sınırı geçen değerler sınıra çekildi (0-255).

```
    //NETLEŞTİRME-------------------------------
     private void netlestirmeToolStripMenuItem_Click_1(object sender, EventArgs e)
     {
        Bitmap OrjinalResim = new Bitmap(pictureBox1.Image);

        Bitmap BulanikResim = MeanFiltresi();
        //Bitmap BulanikResim = GaussFiltresi();

        Bitmap KenarGoruntusu = OrjianalResimdenBulanikResmiCikarma(OrjinalResim,
BulanikResim);
        Bitmap NetlesmisResim = KenarGoruntusuIleOrjinalResmiBirlestir(OrjinalResim,
KenarGoruntusu);

        pictureBox2.Image = NetlesmisResim;
     }


     public Bitmap OrjianalResimdenBulanikResmiCikarma(Bitmap OrjinalResim, Bitmap
BulanikResim)
     {

        Color OkunanRenk1, OkunanRenk2, DonusenRenk;
        Bitmap CikisResmi;

        int ResimGenisligi = OrjinalResim.Width;
        int ResimYuksekligi = OrjinalResim.Height;

        CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

        int R, G, B;
```

```
            double Olcekleme = 1.7; //0.2-0.7 arası uygundur.
            for (int x = 0; x < ResimGenisligi; x++)
            {
                for (int y = 0; y < ResimYuksekligi; y++)
                {
                    OkunanRenk1 = OrjinalResim.GetPixel(x, y);
                    OkunanRenk2 = BulanikResim.GetPixel(x, y);

                    R =Convert.ToInt16(Olcekleme * (OkunanRenk1.R - OkunanRenk2.R));
                    G = Convert.ToInt16(Olcekleme * (OkunanRenk1.G - OkunanRenk2.G));
                    B = Convert.ToInt16(Olcekleme * (OkunanRenk1.B - OkunanRenk2.B));

                    //============================================================
                    //Renkler sınırların dışına çıktıysa, sınır değer alınacak.
                    if (R > 255) R = 255;
                    if (G > 255) G = 255;
                    if (B > 255) B = 255;

                    if (R < 0) R = 0;
                    if (G < 0) G = 0;
                    if (B < 0) B = 0;
                    //============================================================
                    DonusenRenk = Color.FromArgb(R, G, B);
                    CikisResmi.SetPixel(x, y, DonusenRenk);

                }
            }

            return CikisResmi;
        }

        public Bitmap KenarGoruntusuIleOrjinalResmiBirlestir(Bitmap OrjinalResim, Bitmap
KenarGoruntusu)
        {

            Color OkunanRenk1, OkunanRenk2, DonusenRenk;
            Bitmap CikisResmi;

            int ResimGenisligi = OrjinalResim.Width;
            int ResimYuksekligi = OrjinalResim.Height;

            CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);


            int R, G, B;

            for (int x = 0; x < ResimGenisligi; x++)
            {
                for (int y = 0; y < ResimYuksekligi; y++)
                {
                    OkunanRenk1 = OrjinalResim.GetPixel(x, y);
                    OkunanRenk2 = KenarGoruntusu.GetPixel(x, y);

                    R = OkunanRenk1.R + OkunanRenk2.R;
                    G = OkunanRenk1.G + OkunanRenk2.G;
                    B = OkunanRenk1.B + OkunanRenk2.B;
```

```
            //================================================================
            //Renkler sınırların dışına çıktıysa, sınır değer alınacak.
            if (R > 255) R = 255;
            if (G > 255) G = 255;
            if (B > 255) B = 255;

            if (R < 0) R = 0;
            if (G < 0) G = 0;
            if (B < 0) B = 0;
            //================================================================

            DonusenRenk = Color.FromArgb(R, G, B);
            CikisResmi.SetPixel(x, y, DonusenRenk);

        }
    }

    return CikisResmi;
}
```

İKİNCİ YÖNTEM (Konvolüsyon yöntemi-çekirdek matris ile netleştirme)

Aşağıdaki çekirdek matris kullanarakda Netleştirme yapılabilir.

0	-2	0
-2	11	-2
0	-2	0

```
private void netlestirme2ToolStripMenuItem_Click(object sender, EventArgs e)
{
    Color OkunanRenk;
```

```
        Bitmap GirisResmi, CikisResmi;
        GirisResmi = new Bitmap(pictureBox1.Image);

        int ResimGenisligi = GirisResmi.Width;
        int ResimYuksekligi = GirisResmi.Height;

        CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

        int SablonBoyutu = 3;
        int ElemanSayisi = SablonBoyutu * SablonBoyutu;


        int x, y, i, j, toplamR, toplamG, toplamB, ortalamaR, ortalamaG, ortalamaB;

        int R, G, B;
        int[] Matris = { 0, -2, 0, -2, 11, -2, 0,-2,0};
        int MatrisToplami = 0 + -2 + 0 + -2 + 11 + -2 + 0 + -2 +0;

        for (x = (SablonBoyutu - 1) / 2; x < ResimGenisligi - (SablonBoyutu - 1) / 2; x++) //Resmi
taramaya şablonun yarısı kadar dış kenarlardan içeride başlayacak ve bitirecek.
        {
           for (y = (SablonBoyutu - 1) / 2; y < ResimYuksekligi - (SablonBoyutu - 1) / 2; y++)
           {
              toplamR = 0;
              toplamG = 0;
              toplamB = 0;

              //Şablon bölgesi (çekirdek matris) içindeki pikselleri tarıyor.
              int k = 0; //matris içindeki elemanları sırayla okurken kullanılacak.
              for (i = -((SablonBoyutu - 1) / 2); i <= (SablonBoyutu - 1) / 2; i++)
              {
                 for (j = -((SablonBoyutu - 1) / 2); j <= (SablonBoyutu - 1) / 2; j++)
                 {
                    OkunanRenk = GirisResmi.GetPixel(x + i, y + j);

                    toplamR = toplamR + OkunanRenk.R * Matris[k];
                    toplamG = toplamG + OkunanRenk.G * Matris[k];
                    toplamB = toplamB + OkunanRenk.B * Matris[k];

                    R = toplamR / MatrisToplami;
                    G = toplamG / MatrisToplami;
                    B = toplamB / MatrisToplami;

                    //=========================================================
                    //Renkler sınırların dışına çıktıysa, sınır değer alınacak.
                    if (R > 255) R = 255;
                    if (G > 255) G = 255;
                    if (B > 255) B = 255;

                    if (R < 0) R = 0;
                    if (G < 0) G = 0;
                    if (B < 0) B = 0;
                     //=========================================================

                    CikisResmi.SetPixel(x, y, Color.FromArgb(R, G, B));

                    k++;
```

```
                }

            }

        }
    }
    pictureBox2.Image =  CikisResmi;

}
```

Aşağıdaki matrisleride deneyiniz.

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

1	-2	1
-2	4	-2
1	-2	1