GÖRÜNTÜ İŞLEME - (5.Hafta)

RESİM YUMUŞATMA (BULANIKLAŞTIRMA-BLURRING) FİLTRELERİ

Görüntü işlemede, filtreler görüntüyü yumuşatmak yada kenarları belirginleştirmek için dijital filtreler kullanılır. Bu bölümde resim yumuşatma ele alınacaktır. Bu amaçla aşağıdaki filtreler incelenecektir.

- Ortalama Filtre (Mean)
- Orta Değer Filtresi (Median)
- Gauss düzleştirme Filtresi (Gaussian Smoothing)
- Konservativ yumuşatma (Conservative Soomthing)
- Crimmins Parlaklık Giderme (Crimmins Speckle Removal)
- Frekans Filtreleri (Frequency Filters)

Filtrelemede, girdi görüntüsü f(i,j) filtre fonksiyonu ile h(i,j) ile konvolüsyon yapılarak işlem yapılır. Bu ifade matematiksel olarak şu şekilde gösterilebilir.

$$g(i,j) = h(i,j) \odot f(i,j)$$

Konvolüsyon işlemi; bir çekirdek şablonun (matrisin/kernel) resim üzerindeki piksellerle 'kaydırma ve çarpma' işlemi olarak tanımlanabilir. Bu işlemde çekirdek şablon resim üzerinde kaydırılır ve değeri resim üzerindeki uygun piksellerle çarpılır.

Belli özel uygulamalar için çeşitli standart şablonlar vardır. Burada şablonun büyüklüğü ve şekli, işlemin özelliklerini belirler. Örneğin, ortalama (mean) ve Laplace operatörünün şablonları (çekirdek matrisleri) aşağıdaki gibidir.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Mean

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

Laplacian

ORTALAMA FİLTRESİ (Mean Filter -Box Blur)

Yaygın isimleri; Mean filtering (ortalama filtresi), Smoothing (yumuşatma), Averaging (ortalama), Box filtering (kutu filtreleme).

Ortalama filtresi, görüntüleri yumuşatmanın basit ve uygulanması kolay bir yöntemidir. Diğer bir deyişle, bir piksel ile diğerleri arasındaki değişim miktarını azaltmaktır. Genellikle görüntülerdeki gürültüyü azaltmak için kullanılır.

Ortalama filtresi, bir görüntünün her bir piksel değerini komşularının ve kendisinin dahil olduğu ortalama değer ile değiştirmektir. Bu durum, çevresindekileri temsil etmeyen piksel değerlerinin ortadan kalkmasına yol açar. Ortalama filtresi bir konvolüsyon filtresidir. Konvolüsyon filtreleri çekirdek şablon (kernel) temeline dayanır. Şekilde gösterildiği gibi çoğunlukla 3×3 kare çekirdek şablon kullanılır. Bazı yumuşatma işlemlerinde daha büyük şablonlar (5×5, 7x7 gibi) kullanılabilir. Büyük şablonun tek bir taramadaki etkisine benzer bir etki, küçük şablonun birden fazla geçişi ile de sağlanabilir.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Mean (Ortalama)

Şekil. Ortalama filtresinde sıklıkla kullanılan 3×3 ortalama şablonu.

Ortalama filtresi, bir görüntüdeki gürültüyü azaltmak için kullanılan en basit yöntemdir. Ancak gürültü daha az belirgin hale getirilirken, görüntüde yumuşatılmış olmaktadır. Kullanılan çekirdek şablonun (matrisin) boyutu artırılırsa yumuşatma daha da artacaktır.

Ortalama filtrelemeyle ilgili iki ana sorun bulunmaktadır:

a) Resmi çok iyi temsil etmeyen değere sahip bir piksel, yakın bölgedeki tüm piksellerin ortalama değerini önemli ölçüde etkiler. Buda resmin değişmesine sebep olur.
b) Filtre (şablon) bir kenar üzerinden geçerken, kenarın her iki tarafındaki pikseller için yeni değerler üretecektir ve bu durum kenarın bulanıklaşmasına sebep olacaktır. Eğer keskin kenarların kaybolması istenmiyorsa bu bir sorun olabilir.

Bu iki problemi gidermek için Ortalama filtresi (mean) yerine, Medyan filtresi (Median Filter) geliştirilmiştir. Fakat bu filtrenin de hesaplama süresi uzun sürmektedir. Resmi yumuşatma için yaygın kullanılan filtrelerden biri de "Gauss yumuşatma filtresi" (Gaussian smoothing filter) dir.

Programlama (Ortalama Filtresi-Mean Filter)

Normal resim

3x3 Şablon

5x5 Şablon

7x7 Şablon

```
        public void meanFiltresi ()
        {
            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 = Convert.ToInt32(textBox1.Text);  //şablon boyutu 3 den büyük tek rakam olmalıdır (3,5,7 gibi).
            int x, y, i, j, toplamR, toplamG, toplamB, ortalamaR, ortalamaG, ortalamaB;

            for (x = (SablonBoyutu - 1) / 2; x < ResimGenisligi - (SablonBoyutu - 1) / 2; x++)
            {
                for (y = (SablonBoyutu - 1) / 2; y < ResimYuksekligi - (SablonBoyutu - 1) / 2; y++)
                {
                    toplamR = 0;
                    toplamG = 0;
                    toplamB = 0;

                    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;
                            toplamG = toplamG + OkunanRenk.G;
                            toplamB = toplamB + OkunanRenk.B;

                        }
                    }

                    ortalamaR = toplamR / (SablonBoyutu * SablonBoyutu);
                    ortalamaG = toplamG / (SablonBoyutu * SablonBoyutu);
                    ortalamaB = toplamB / (SablonBoyutu * SablonBoyutu);
```

```
                CikisResmi.SetPixel(x, y, Color.FromArgb(ortalamaR, ortalamaG, ortalamaB));
            }
        }

        pictureBox2.Image = CikisResmi;
}
```

Yukarıda verilen örnek resim filtrenin özelliklerini görmek için yeterli olmaz. Yumuşatma filtreleri daha çok resim üzerindeki gürültüyü azaltmak için kullanılır. Aşağıdaki resimler Mean filtresinin gürültülü bir resim üzerindeki etkisini göstermektedir. Sırayla normal gürültülü resim, 3x3, 5x5 ve 7x7 şablon (matris) kullanılan resimlerdir.

Normal resim

3x3 Şablon

5x5 Şablon

7x7 Şablon

Aynı resim üzerinde 3 defa 3x3 şablonu ile Mean filtresini uygularsak aşağıdaki gibi sonuç alırız. Bu sonucun 5x5 şablonuna yakın bir sonuç verdiğini görebiliriz. Her iki uygulamada da işlem gören piksel sayısı yakın bir değer olmuş olur. (3x3=9)x3=27 piksel. 5x5=25 piksel.

5x5 Şablon

3 defa 3x3 şablonu uygulanmış resim

Araştırma: Çekirdek matris resim üzerinde ilerlerken her seferinde eski resmin piksellerine bakmaktadır. Oysa rengi değişmiş olan piksellerin rengi de kullanılabilir. Böyle uygulama durumunda resim de nasıl bir değişiklik görürüz. Deneyerek bulmaya çalışın.

MEDYAN FİLTRESİ (ORTA DEĞER FİLTRESİ-MEDİAN FİLTER)

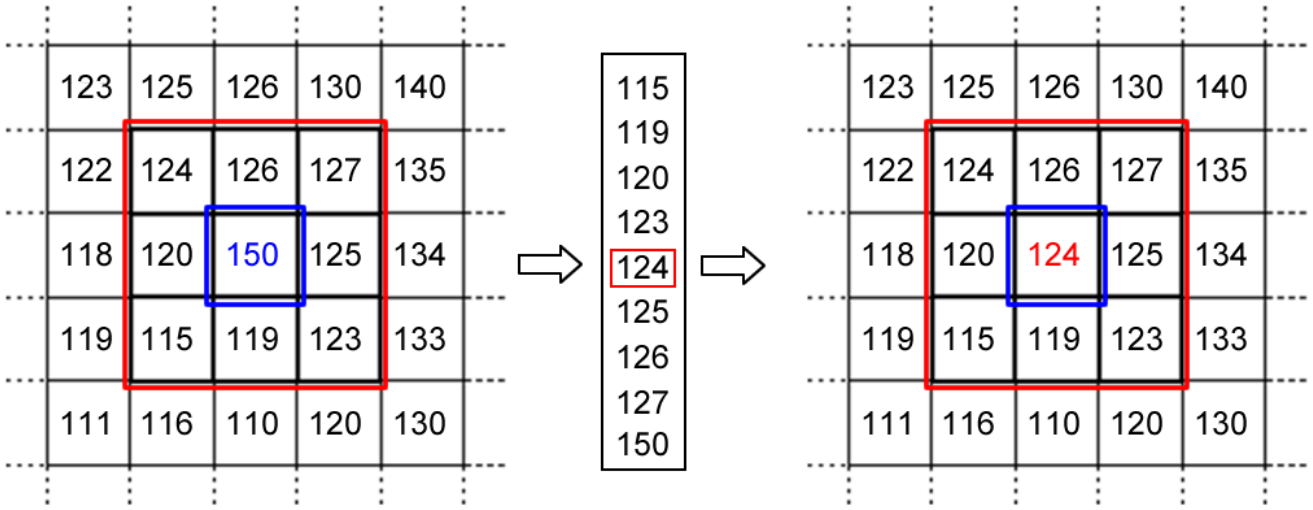
Bu filtrenin yaygın kullanılan isimleri; Medyan filtresi (Orta Değer filtresi) (Median filtering), Sıralama filtresi (Rank filtering)

Medyan filtresi, normal olarak mean filtresi gibi bir resimdeki gürültüyü azaltmak için kullanılır. Ancak resim üzerindeki detayların kaybolmaması noktasında mean filtresinden çok daha iyi sonuç verir.

Medyan filtre de mean filtresi gibi her pikselin değerini hesaplamak için yakınındaki komşularına bakar. Medyan filtresinde piksel değeri komşu piksel değerlerinin ortalaması ile değiştirmek yerine (mean filtresi), komşu pikselleri sıralayıp sıranın ortasındaki değeri alır. Eğer incelenen bölge (şablonun içerisi) çift sayıda piksel varsa, orta değer olarak, ortada bulunan iki pikselin ortalaması kullanılır.

Aşağıdaki şekilde ortadaki piksele göre işlemlerimizi yaparsak bu pikselin değeri olan 150 sayısı çevresindeki pikselleri iyi bir şekilde temsil etmediğini görebiliriz. Bu pikselin değerini değiştirirken öncelikle çevresindeki piksellerin değerini bir sıraya dizelim. Bunlar (115, 119, 120, 123, 124, 125, 126, 127,150) değerlerinden oluşur. Bu değerlerin en ortasında ise 124 sayısı vardır. Buna göre 150 rakamı 124 sayısı ile değiştirilir. Burada 124 sayısı medyan sayısı (orta değer) olmuş olur.

Burada kullanılan şablon 3x3 piksel boyutlarındadır. Daha büyük şablonların kullanılması daha fazla pürüzsüzleştirme (yumuşatma) etkisi üretir.



Medyan filtrenin Mean filtresine nazaran iki avantajı vardır.

- Orta değeri (median) kullanmak, ortalama değeri kullanmaktan (mean) daha güçlü olarak şablonu temsil eder. Temsil yeteneği uzak bir piksel sıralanan dizinin uçlarında kalacağından (hiç bir zaman ortada bulunmayacaktır) oradaki komşuların genel temsilini etkilemesi imkansız hale gelmiş olur.
- Medyan değer (orta değer), komşu piksellerin birinin değeri olması gerektiği için, kenar boyunca hareket ettiğinde gerçekçi olmayan piksel değerleri oluşturmaz. Bu nedenle, medyan filtre, keskin kenarları ortalama filtreden (mean) daha iyi korur. Örnekleyecek olursa, siyah ve beyazdan oluşan bir sınırda ortadaki değer ya siyah olur yada beyaz olur. İkisinin ortalaması olan gri olmayacaktır. Böylece kenar üzerindeki keskinlik kaybolmamış olacaktır.

Programlama (Medyan Filtresi)

Normal resim

3x3 Şablon

5x5 Şablon

7x7 Şablon

```
public void medianFiltresi ()
{
    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 = 0;
    try
    {
      SablonBoyutu = Convert.ToInt32(textBox1.Text);
    }
      catch
    {
       SablonBoyutu = 3;
    }
   int ElemanSayisi = SablonBoyutu * SablonBoyutu;

    int[] R = new int[ElemanSayisi];
    int[] G = new int[ElemanSayisi];
    int[] B = new int[ElemanSayisi];
    int[] Gri = new int[ElemanSayisi];

    int x, y, i, j;

    for (x = (SablonBoyutu - 1) / 2; x < ResimGenisligi - (SablonBoyutu - 1) / 2; x++)
    {
       for (y = (SablonBoyutu - 1) / 2; y < ResimYuksekligi - (SablonBoyutu - 1) / 2; y++)
       {
          //Şablon bölgesi (çekirdek matris) içindeki pikselleri tarıyor.
          int k = 0;
          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);
```

```
                    R[k] = OkunanRenk.R;
                    G[k] = OkunanRenk.G;
                    B[k] = OkunanRenk.B;

                    Gri[k] = Convert.ToInt16(R[k] * 0.299 + G[k] * 0.587 + B[k] * 0.114); //Gri ton formülü

                    k++;
                }
            }

            //Gri tona göre sıralama yapıyor. Aynı anda üç rengide değiştiriyor.
            int GeciciSayi = 0;

            for (i = 0; i < ElemanSayisi; i++)
            {
                for (j = i + 1; j < ElemanSayisi; j++)
                {
                    if (Gri[j] < Gri[i])
                    {
                        GeciciSayi = Gri[i];
                        Gri[i] = Gri[j];
                        Gri[j] = GeciciSayi;

                        GeciciSayi = R[i];
                        R[i] = R[j];
                        R[j] = GeciciSayi;

                        GeciciSayi = G[i];
                        G[i] = G[j];
                        G[j] = GeciciSayi;

                        GeciciSayi = B[i];
                        B[i] = B[j];
                        B[j] = GeciciSayi;
                    }
                }
            }

            //Sıralama sonrası ortadaki değeri çıkış resminin piksel değeri olarak atıyor.
            CikisResmi.SetPixel(x, y, Color.FromArgb(R[(ElemanSayisi - 1) / 2], G[(ElemanSayisi - 1) /
2], B[(ElemanSayisi - 1) / 2]));
        }
    }

    pictureBox2.Image = CikisResmi;

}
```

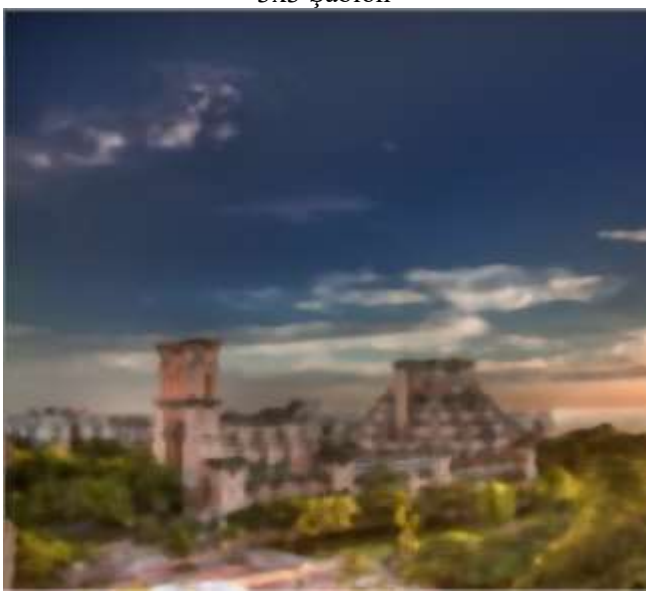
Medyan filtresinin dezavantajı olarak şundan bahsedilebilir. Resim üzerinde büyük boyutlarda gürültü varsa, Medyan filtresi küçük matrislerle (şablonla) gürültüyü tam olarak kaldıramaz. Bu gibi durumlarda daha büyük matrisler kullanmak gerekir yad a bir kaç kez aynı filtreden geçirmek gerekebilir.

Bir diğer dezevantajı ise hesaplama zamanın uzun olmasıdır. Çünkü komşu piksellerin değerlerini alıp bunları sıralamaya tabi tutar. Bazı akıllı algoritmalar ile sıralama hızı artırılabilir. Örneğin şablon resim üzerinde gezerken bir birçok pikselin değeri bir adımdan diğerine aynı kalmaktadır.

Normal resim

3x3 Şablon

5x5 Şablon



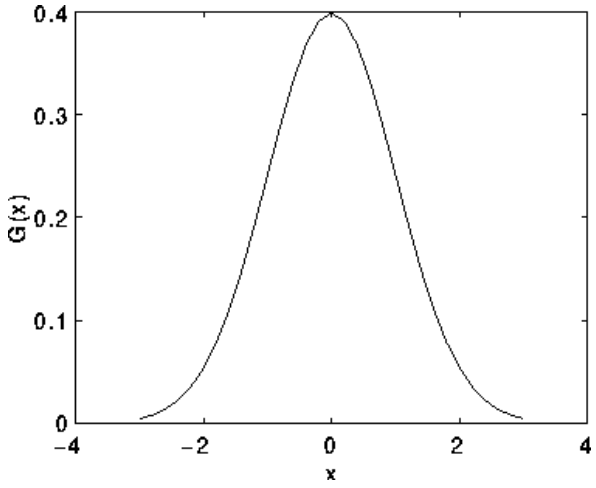
7x7 Şablon

GAUSS BULANIKLAŞTIRMA/YUMUŞATMA FİLTRESİ (Gaussian Smoothing- Gaussian blur)

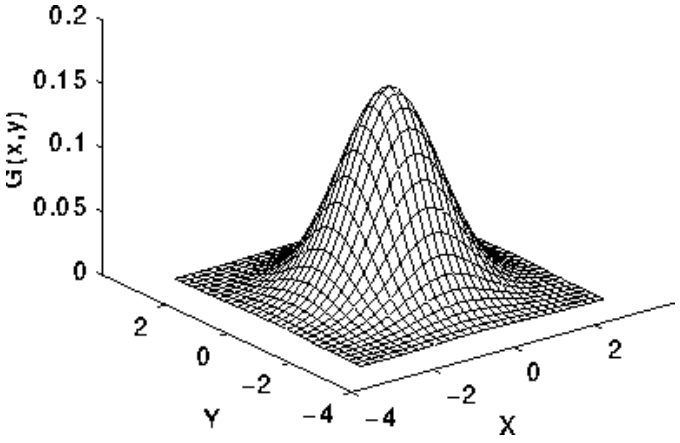
Gauss yumuşatma operatörü, görüntüleri 'bulanıklaştırmak', ayrıntı ve gürültüyü ortadan kaldırmak için kullanılan 2 boyutlu konvolüsyon (çekirdek matris ile resim üzerindeki piksellerin çarpımı işlemi) operatörüdür. Bu anlamda, ortalama (Mean) filtreye benzer. Ancak Gauss aşağıda resmi verilen "çan şeklindeki" grafikle temsil edilebilecek farklı bir çekirdek şablon (matris) kullanır.

Gauss'un çan şeklindeki grafiğini veren formül 2 boyutlu düzlem ve 3 boyutlu uzay için yazılırsa aşağıdaki şekilde gösterilebilir.

$$G(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{x^2}{2\sigma}}$$



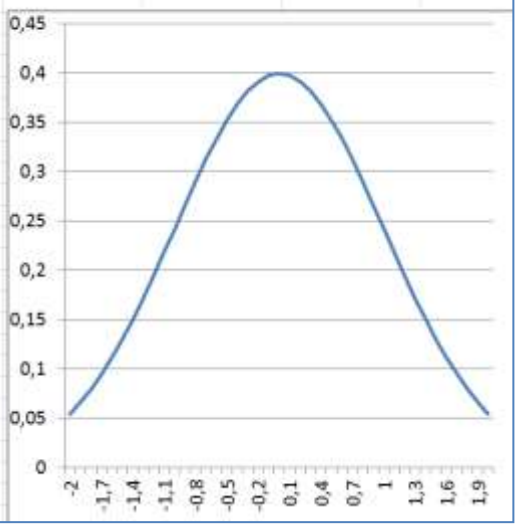
$$G(x) = \frac{1}{\sqrt{2\pi}\,\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



Burada σ dağılımın standart sapmasıdır. Ayrıca, dağılımın ortalamasının sıfır olduğu varsayılmıştır (yani, x = 0 çizgisine ortalanmıştır). 2 boyutlu ve 3 boyutlu grafik eksenel olarak simetrik değerlere sahiptir.

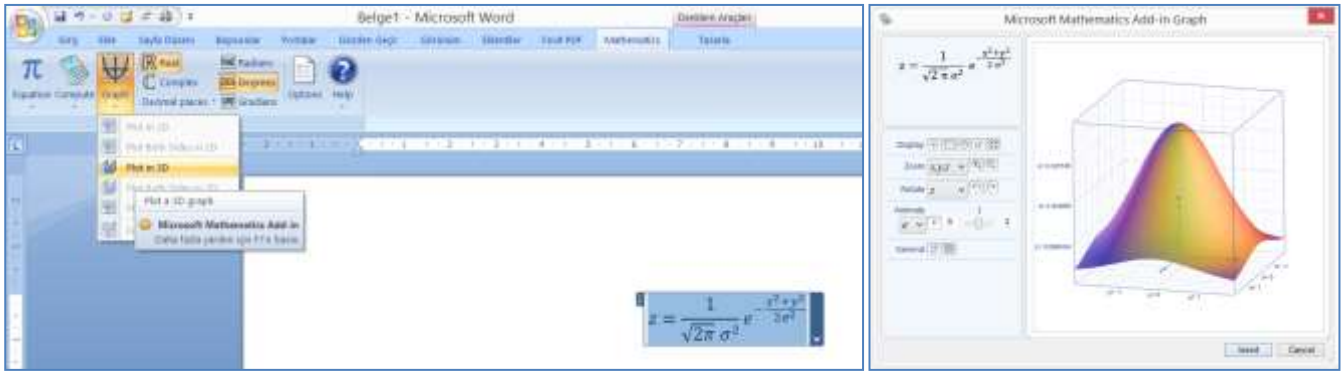
İki boyutlu grafiği Standart sapma σ=1 için x=∓2 aralığında excel de çizecek olursak aşağıdaki grafiği elde ederiz.

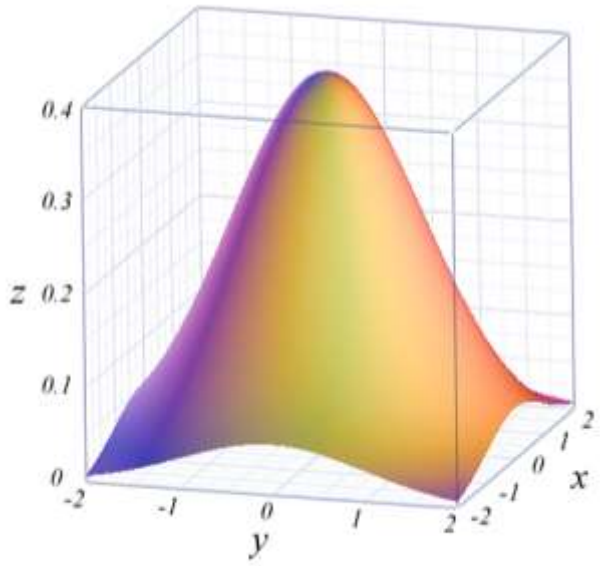
x	σ	$\frac{1}{\sqrt{2\pi}\,\sigma}$	$e^{-\frac{x^2}{2\sigma}}$	$z = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{x^2}{2\sigma}}$
-2	1	0,398943084	0,135335283	0,053991075
-1,9	1	0,398943084	0,164474457	0,065615947
-1,8	1	0,398943084	0,197898699	0,078950317
-1,7	1	0,398943084	0,235746077	0,094049267
-1,6	1	0,398943084	0,2780373	0,110921058
-1,5	1	0,398943084	0,324652467	0,129517856
-1,4	1	0,398943084	0,375311099	0,149727767
-1,3	1	0,398943084	0,429557358	0,171368937
-1,2	1	0,398943084	0,486752256	0,194186446
-1,1	1	0,398943084	0,546074427	0,217852616
-1	1	0,398943084	0,60653066	0,241971212
-0,9	1	0,398943084	0,666976811	0,266085786
-0,8	1	0,398943084	0,726149037	0,289692136
-0,7	1	0,398943084	0,782704538	0,312254562
-0,6	1	0,398943084	0,835270211	0,333225274
-0,5	1	0,398943084	0,882496903	0,352066036
-0,4	1	0,398943084	0,923116346	0,368270882
-0,3	1	0,398943084	0,955997482	0,381388584
-0,2	1	0,398943084	0,980198673	0,391043481
-0,1	1	0,398943084	0,995012479	0,396953347
0	1	0,398943084	1	0,398943084
0,1	1	0,398943084	0,995012479	0,396953347
0,2	1	0,398943084	0,980198673	0,391043481
0,3	1	0,398943084	0,955997482	0,381388584



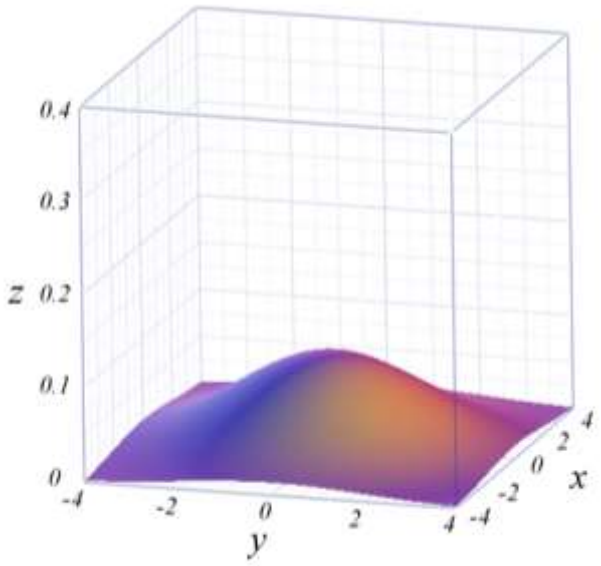
3 boyutluğu grafiği çizmek için Word içerisine yine microsoft'un bir eklentisi olan Mathematics programını (https://www.microsoft.com/en-us/download/confirmation.aspx?id=17786) adresinden indirip kuralım. Word'ü kapatıp açtığımızda ilgili sekme gözükecektir. Ardından denklemimizi sayfada yazıp seçersek

menüden Plot in 3D seçip grafiği çizdirebiliriz. Grafik üzerinde bazı ayarları (çizim aralıklarını vs seçip istenilen grafiğin durumunu görebiliriz.





a) Standart sapma σ=1 için çizilen grafik



b) Standart sapma σ=2 için çizilen grafik

Dikkat edilirse standart sapma değeri arttıkça grafik daha yayvan hale gelmektedir. z değerinin 0 yakın bir değerden başlaması için x ve y değerlerinin daha büyük değer aralığını almasını gerektirir (Burada σ=2 için -4,+4 aralığı alınmıştır) . Dolayısı ile standart sapma değeri (σ) arttıkça onu temsil eden matrisleride daha büyük seçmek uygun olacaktır.

Standart sapma σ=1 için matris değerlerimizi hesaplayalım. Bunun için 5x5 bir matris oluşturalım. Dolayısıyla x ve y değer aralığımızı -2,-1,0,1,2 olarak belirleyelim.

x	y	σ	$\frac{1}{\sqrt{2\pi}\,\sigma^2}$	$e^{-\frac{x^2+y^2}{2\sigma^2}}$	$G(x)=\frac{1}{\sqrt{2\pi}\,\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$	Ölçek: 1/0,007306897= 136,856991
-2	-2	1	0,398943084	0,018315639	0,007306897	1
-1	-2	1	0,398943084	0,082084999	0,032747242	4,48168907
0	-2	1	0,398943084	0,135335283	0,053991075	7,389056099
1	-2	1	0,398943084	0,082084999	0,032747242	4,48168907
2	-2	1	0,398943084	0,018315639	0,007306897	1
-2	-1	1	0,398943084	0,082084999	0,032747242	4,48168907
-1	-1	1	0,398943084	0,367879441	0,146762959	20,08553692
0	-1	1	0,398943084	0,60653066	0,241971212	33,11545196
1	-1	1	0,398943084	0,367879441	0,146762959	20,08553692
2	-1	1	0,398943084	0,082084999	0,032747242	4,48168907

-2	0	1	0,398943084	0,135335283	0,053991075	7,389056099
-1	0	1	0,398943084	0,60653066	0,241971212	33,11545196
0	0	1	0,398943084	1	0,398943084	54,59815003
1	0	1	0,398943084	0,60653066	0,241971212	33,11545196
2	0	1	0,398943084	0,135335283	0,053991075	7,389056099
-2	1	1	0,398943084	0,082084999	0,032747242	4,48168907
-1	1	1	0,398943084	0,367879441	0,146762959	20,08553692
0	1	1	0,398943084	0,60653066	0,241971212	33,11545196
1	1	1	0,398943084	0,367879441	0,146762959	20,08553692
2	1	1	0,398943084	0,082084999	0,032747242	4,48168907
-2	2	1	0,398943084	0,018315639	0,007306897	1
-1	2	1	0,398943084	0,082084999	0,032747242	4,48168907
0	2	1	0,398943084	0,135335283	0,053991075	7,389056099
1	2	1	0,398943084	0,082084999	0,032747242	4,48168907
2	2	1	0,398943084	0,018315639	0,007306897	1

Bu değerler matris formatına dönüştürülürse aşağıdaki Gauss yumuşatma operatörünün çekirdek matrisi elde edilmiş olur.

Y \ X	-2	-1	0	1	2
-2	1	4	7	4	1
-1	4	20	33	20	4
0	7	33	55	33	7
1	4	20	33	20	4
2	1	4	7	4	1

Diğer sayılarda ona göre en yakın tam sayıya yuvarlandığında bu tablo elde edilir. Gauss formülündeki katsayılar bu mantıkla değiştirilerek farklı değerlerde matrislerde elde edilebilir. Tablo yanındaki çarpım değerleri matristeki elemanların toplamını gösterir. Burada olduğu gibi uygun bir Gauss çekirdek matrisi oluşturulduktan sonra ortadaki piksel, resim üzerindeki hedef piksel üzerinde olmak üzere, komşu pikseller de matristeki komşu değerler ile çarpılır ve sol taraftaki toplam değere (273, 112, 16 şeklinde verilen değerler) bölünür.

$\frac{1}{273}$ x

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

$\frac{1}{112}$ x

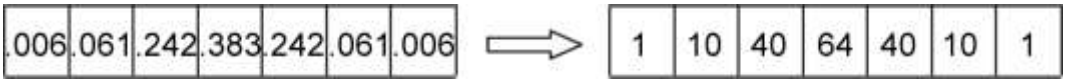
1	2	4	2	1
2	6	9	6	2
4	9	16	9	4
2	6	9	6	2
1	2	4	2	1

$\frac{1}{16}$ x

1	2	1
2	4	2
1	2	1

Burada olduğu gibi konvolüsyon işlemi 3 boyutlu grafikten türetilen 2 boyutlu matrisle (kare matris) yapılabileceği gibi, 2 boyutlu grafikten türetilen 1-boyutlu vektör matrisle, iki ayrı işlemle yapılabilir. 1-boyutlu vektör matrise örnek aşağıda verilmiştir (1x7 matris). Buradaki 1x7 lik matris 2 boyutlu gauss

grafiğinden elde edilmiştir. Matristeki değerler en uçtaki değer 1 olacak şekilde ölçeklenirse yanında verildiği şekilde çekirdek matris elde edilir. Bu matrisin en ortadaki elemanı işlem yapılan piksel üzerinde olmak üzere önce x ekseninde yatay olarak tüm resimdeki piksellerle çarpılabilir, ardından dikeyde y ekseni üzerindeki tüm piksellerle çarpılarak konvolüsyon işlemi yapılabilir.

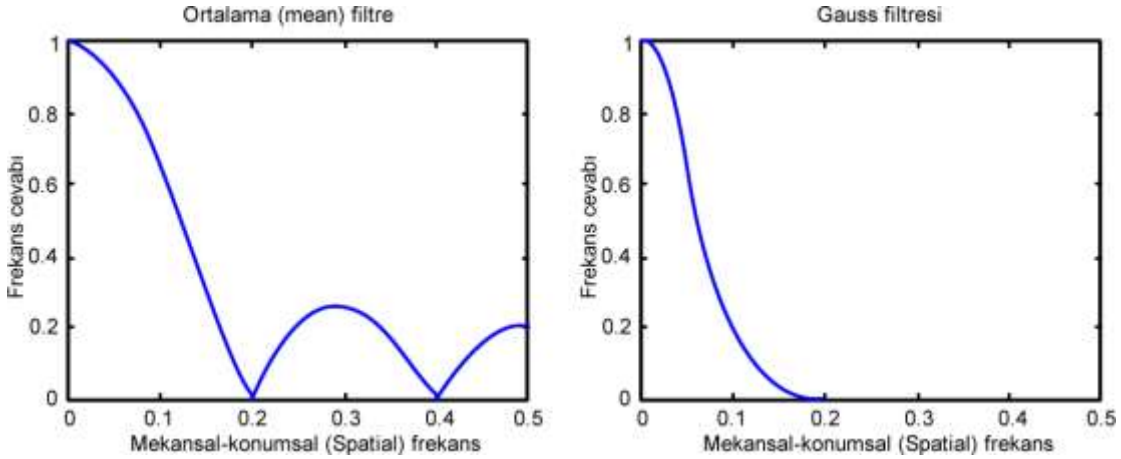


Büyük çekirdek matris kullanarak hesaplama yapmak yerine, daha küçük matrisle resim üzerinden geçmek (konvolüsyon yapmak) de mümkündür. Bu tür bir işlem paralel donanımlar kurarak hesaplama için daha uygun olur.

Gauss'un resim düzgünleştirme etkisi, bir görüntüyü ortalama filtreye benzer şekilde bulanıklaştırmaktır. Düzeltme derecesi Gaussian'ın standart sapması ile belirlenir. Daha büyük standart sapma değeri, ortaya çıkan grafiği daha geniş hale getirir. Bundan türetilen çekirdek matrisinde doğru bir temsil yapabilmesi için daha büyük boyutta olmasını gerektirir.

Gauss, her piksel bölgesinin ağırlıklı ortalamasını çıkarır. Merkez piksel değerine doğru yaklaştıkça ağırlıklandırma artar. Bu durum, ortalama filtrenin (Mean) (her yeri eşit ağırlıklandırma yapar) aksine daha ince bir düzeltme sağlar, kenarları benzer büyüklükteki bir ortalama filtreden daha iyi korur.

Gauss yumuşatma filtresinin kullanmanın gerekçelerinden biri de, resim üzerindeki uzaysal-mekansal frekans bileşenleri korumaktır (yani çok sık aralıkla resim üzerindeki değişimleri korumasıdır). Çoğu diğer filtre alçak geçirgen filtre gibi davranır (yap hep ya hiç kuralı gibi). Her iki filtrede yüksek frekansları zayıflatır (örn: keskin kenarları). Ancak Ortalama filtresi frekans tepkimesi olarak salınımlar gösterir. Gauss filtresi ise salınım göstermez. Aşağıdaki şekiller Ortalama filtre (5x5 boyutunda) ve Gauss filtrenin (standart sapması σ=3) frekans tepkilerini göstermektedir.



Gauss filtrenin daha büyük matrislerle (standart sapmalarla) pürüzsüzleştirme etkisini görmek için şu örnekleri inceleyin.

Gauss Filtresi Programlama

Aşağıdaki program 5x5 lik yukarıda şekli verilen { 1, 4, 7, 4, 1, 4, 20, 33, 20, 4, 7, 33, 55, 33, 7, 4, 20, 33, 20, 4, 1, 4, 7, 4, 1 } matrisini kullanarak uygulanmıştır. Soldaki orjinal resimdir.




```
        private void gaussFiltresiToolStripMenuItem_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 = 5; //Çekirdek matrisin boyutu
            int ElemanSayisi = SablonBoyutu * SablonBoyutu;


            int x, y, i, j, toplamR, toplamG, toplamB, ortalamaR, ortalamaG, ortalamaB;
            int[] Matris = { 1, 4, 7, 4, 1, 4, 20, 33, 20, 4, 7, 33, 55, 33, 7, 4, 20, 33, 20, 4, 1, 4, 7, 4, 1 };
            int MatrisToplami = 1 + 4 + 7 + 4 + 1 + 4 + 20 + 33 + 20 + 4 + 7 + 33 + 55 + 33 + 7 + 4 + 20 +
33 + 20 + 4 + 1 + 4 + 7 + 4 + 1;
```

```
            //int[] Matris = { 1, 2, 4, 2, 1, 2, 6, 9, 6, 2, 4, 9, 16, 9, 4, 2, 6, 9, 6, 2, 1,
2, 4, 2, 1 };
            //int MatrisToplami = 1 + 2 + 4 + 2 + 1 + 2 + 6 + 9 + 6 + 2 + 4 + 9 + 16 + 9 + 4 + 2
+ 6 + 9 + 6 + 2 + 1 + 2 + 4 + 2 + 1;
```

```
        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];


                        k++;
                    }

                }
                    ortalamaR = toplamR / MatrisToplami;
                    ortalamaG = toplamG / MatrisToplami;
                    ortalamaB = toplamB / MatrisToplami;

                   CikisResmi.SetPixel(x, y, Color.FromArgb(ortalamaR, ortalamaG, ortalamaB));

            }
        }
        pictureBox2.Image = CikisResmi;

    }
```

Görüldüğü gibi Gauss filtresi kenarları korumada daha başarılıdır. Fakat Tuz-biber ekilmiş gürültülü resimlerde çok da başarılı sayılmaz (Bu tip resimlerde Medyan filtresinin başarılı olduğunu hatırlayalım). Gauss filtresinin 9x9, 15x15 örneklerini de siz deneyin. Ayrıca aynı resim için Mean ve Medyan filtresi ile karşılaştırmalar yapınız.

Ödev 1: Örnek 4 tane resim tespit edin. Bir tanesi insan yüzü, saç ve kirpiklerin olduğu bir resim olsun. Bir tanesi üzeride tekrar eden çizgilerin olduğu bir resim olsun (Izgara benzeri şeyler olsun). Bir taneside bir manzara resmi olsun. Bir tanede üzerinde tuz-biber noktaları oluşturulmuş bir resim olsun. 3 farklı Algoritmayı (Mean, Median, Gauss) algoritmayı 3x3, 5x5, 7x7 ve 9x9 matrisleri kullanarak bu 4 resim üzerinde deneyin. Ortaya çıkan sonuçları yorumlayın. En iyi hangi algoritmalar hangi matriste sonuç verdi bulun.

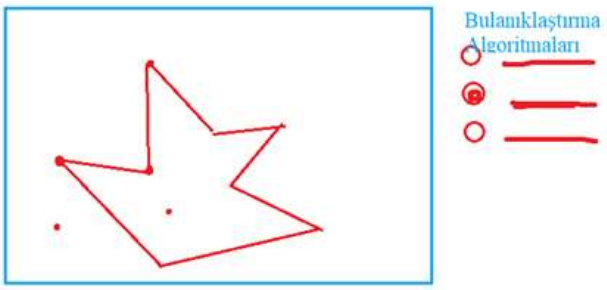
Ödev 2: İnternetten bulanıklaştırma ile ilgili başka bir algoritma bulun. Yada tamamen kendi mantığınıza göre bir algoritma geliştirin (ders notlarında anlatılmayan bir yöntem olması gerekir). Bu algoritmanın performansını Mean, Median ve Gauss ile karşılaştırın. Örnek başka blur algoritmaları: Konservativ yumuşatma (Conservative Soomthing), Crimmins Parlaklık Giderme (Crimmins Speckle Removal), Frekans Filtreleri (Frequency Filters) vs.. daha başka vardır araştırın.

Ödev 3: İçerisinde reklam olan bir resmin köşelerine tıklayın. Sadece o bölgeyi buzlu cama dönüştürsün (bulanıklaştırsın) ve reklamı gizlesin.

Ödev 4: Gauss algoritmasında standart sapmanın etkisinin ne olduğunu programlayarak farklı resimler üzerinde gösterin ve yorumlayın.

Ödev 5: İçerisinde bina olan bir resmi açın. Resim üzerinde binanın dış kenarların mouse ile tıklayarak bir çerçeve çizin. Çok sayıda nokta olacaktır. Ardından Ortalama (mean) algoritmasını çalıştırın. Tüm resmi tararken binanın dışında kalan kısımları bulanıklaştırsın. Binanın içerisindeki bölgeye gelince resim net olarak kalsın. Böylelikle baktığımız nesneyi net olarak, arka planı ise bulanık görmüş oluruz.

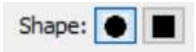
Çoklu nokta şeklinde yapamayanlar, dikdörtgen çerçeve şeklinde bölgeyi seçsinler. Bu şekilde yapanlar daha düşük puan alır.



Ödev 6: Blur (bulanıklaştırma) aracı kullanarak bir resim üzerinde mouse takip eden bulur işlemi yaptırın. Bunun için aşağıdaki adımları takip edin.

a) Mouse ın etrafında büyüklüğü ayarlanabilir belli bir bölge olsun. Örneğin açılır menüden 50 piksel seçilirse mouse etrafında 50x50 lik kare bir şekilde dolaşsın. Bu alanın içi bulanıklaşsın.

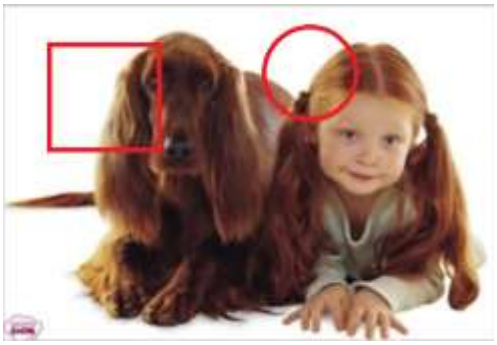
b) Kişi ister bu şekli Kare yada Daire olarak değiştirebilsin. Bu bölge Kare yada Daire olarak seçilebilsin. Bu bölgenin içerisi mouse hareket ettikçe, bulanıklaşsın.

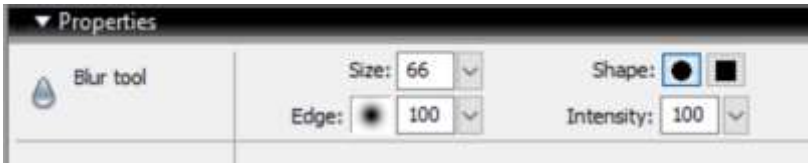


c) Ayrıca bu aracın kenarları üzerinde geçişli bir bulanık oluşturmak için bir ayar daha eklensin. Örneğin burada %100 seçilirse Dairenin merkezinden kenara kadar yavaş yavaş bulur azalarak gitsin. %50 seçilirse darenin yarısından sonra yavaş yavaş bulur azalsın.

d) Bulurun yoğunluğunun ayarlanabildiği bir seçenek daha ekleyin. Aşağıda Intensity olarak gösterilmiş. Bu ayarı kullanmak için kullanılan çekirdek matrsi boyutunu artırarak gerçekleştirebilirsiniz. 7x7 lik matris kullanımı 3x3 lük matristen daha fazla yoğunluk bulur yapar.

e) Kullanılan algoritma açılır menüden, yada radyo buton listesinden seçilebilsin. Örneğin; Mean algoritması, Medyan algoritması, Gauss algoritması gibi .





Ödev 7: Bulanıklaştırma işlemi yaparken resim üzerinde gezdirilen matrisin içindeki renk değerlerini tarayın. Bu renk değerlerinden en fazla hangi renk var ise o rengi yandaki resim üzerine atasın. Algoritmanın kodları Medyan algoritmasına benzerdir. O kodları kullanın. Algoritmada kullanılan matrisin büyüklüğü dışarıdan girilebilsin. 3x3, 5x5, 7x7, 9x9 olmak üzere çeşitli büyüklükte matrisler kullanarak performansını ölçün.

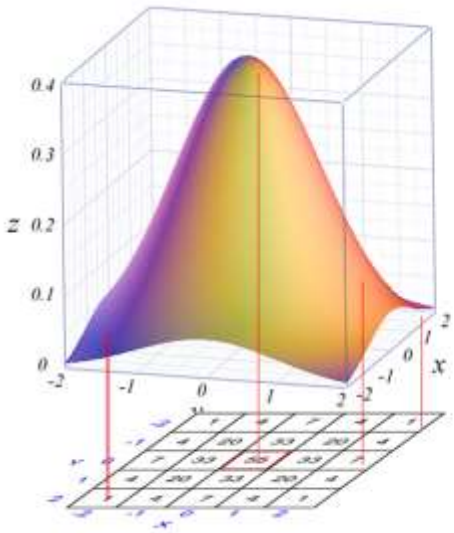
Bu şekilde yapılan bulanıklaştırma resmini Ortalama (Mean) ve Orda Değer (Median) algoritmaları ile karşılaştırın. (Bu iki algoritmanın kodları zaten notlarda var). Farklı resimler üzerinde deneyerek sizin geliştirdiğiniz algoritmanın hangi resimlerde daha mantıklı bulanıklaştırma yaptığını bulun.

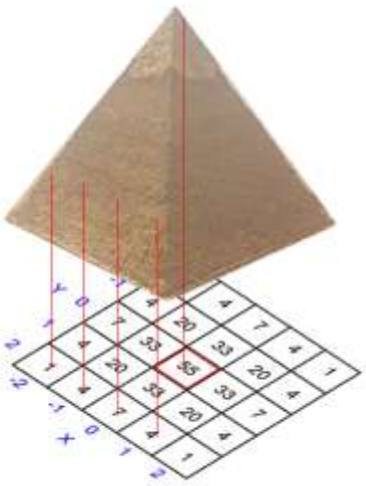
Örnek: Aşağıda 5x5 bir matrisin içindeki sayılara baktığımızda. Tabii renkli resimde 3 kanal var. Siz Medyan filtresinde olduğu gibi Gri renk üzerinden sıralama yaparak en fazla renk sayısı hangisi ise onu atayabilirsiniz. Sonuçların mantıklı çıkıp çıkmayacağını görün bakalım.

[1, 1, 1, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 7, 7, 8, 8, 8, 8 9, 9, 9, 9] =(En fazla değer 3 tür. Piksele 3 atanır)

Ödev 8: Gauss algoritmasında kullanılan matrisin değerlerini oluştururken bir tepe şeklinde geometriden değerler alınıyordu. Bu şekilde oluşan yükseltilere göre tabandaki matris oluşturuluyordu. Bu sefer Tepe şeklinde değilde Pramit şeklinde (Mısır piramitlerini göz önüne getirin) bir yükseltiden tabandaki Matrisin değerlerini oluşturun. Ona göre resim üzerinde bir bulanıklaştırma yapın.

Bu uygulamada matrisde iç içe halkaların hepsi aynı sayıları alır. En dış halka 1 ise daha iç halka diyelim 3 onunda içindeki 5, ortadaki de 7 gibi değerler alır. Yani sayılar lineer (doğrusaldır).





Ek Açıklama: Standart Sapma Nedir?

Her ne kadar görüntü işleme ile ilgisi bulunmasa da burada geçtiği için standart sapmanın ne olduğunu anlamaya çalışalım.

Tanımı: Sayılardan oluşan bir dizideki her bir elemanın, o sayıların aritmetik ortalaması ile farkının karelerinin, dizideki eleman sayısının bir eksiğine bölümünün karekökünе standart sapma denir. Formülüze edersek aşağıdaki şekilde gösterebiliriz.

Dizideki elemanları x_1, x_2, ..., x_n ile gösterirsek bu elemanların ortalaması ve standart sapması aşağıdaki şekilde olur.

$$Ortalama(O) = \frac{x_1 + x_2 + x_3 + \cdots x_n}{n}$$

$$Standart\ Sapma = \sqrt{\frac{(x_1 - O)^2 + (x_2 - O)^2 + (x_3 - O)^2 + \cdots (x_n - O)^2}{n - 1}}$$

Standart sapmanın anlamı nedir? Elimizde iki farklı veri grubu olsun. Bunların her ikisininde aritmetik ortalaması aynı olsun. Bu verilerin ortalamaları aynı ise bunları karşılaştırmak için neye bakabiliriz. Bunun için merkezi yayılma durumuna bakabiliriz. Yani verilerin ortalamadan ne kadar uzakta yada yakında toplandığı bize bir fikir verebilir. Buna standart sapma denir.

- Standart sapma verilerin aritmetik ortalamaya göre nasıl bir yayılım gösterdiğini anlatır.
- Aritmetik ortalamaları aynı olan farklı veri gruplarından; açıklığı küçük olanın standart sapması küçük, açıklığı büyük olanın standart sapması büyüktür.
- Standart sapması küçük olması bir veri grubundaki değerlerin birbirine yakın olduğunu gösterir, büyük olması ise veri grubundaki değerlerin birbirinden uzak olduğunu gösterir.
- Standart sapmanın küçük olması, ortalamadan sapmanın az olduğunu gösterir buda riskin az olduğu yönünde yorumlanabilir. büyük olması durumunda ise tam tersidir.

Örnek: İki öğrencinin bir hafta içinde okudukları kitap sayfa sayısı 5 gün içinde şu şekilde gerçekleşmiş olsun. A={10,25,15,20,10} B={15,10,35,5,15}. Bu iki öğrencinin günlük kitap okuma ortalamalarına bakacak olursak her ikisi de aynı ortalamaya sahip (ortalama 16 sayfa kitap okumuşlardır). Peki bu iki öğrenci aynı performansa mı sahiptir? Ortalamaları aynı ise, aynı performansa sahip diyebiliriz fakat durum öyle değildir. En düzenli ve istikrarlı okuyan hangisi ise onu daha performanslı saymak gerekir. Bunun içinde günlük okumalarının ortalamadan ne kadar saptığını ölçmemiz gerekir. Bu amaçla Standart sapmasını hesaplamalıyız. Hesaplayalım.

Dizimizdeki eleman sayısı n=5 dir. 5 gün boyunca okunan değerler ölçülmüştür.

A nın Ortalaması: (10+25+15+20+10) / 5=16

B nin Ortalaması: (15+10+35+5+15) / 5=16

A nın Standart Sapması: $\sqrt{\frac{(16-10)^2+(16-25)^2+(16-15)^2+(16-20)^2+(16-10)^2}{(5-1)}} = 6{,}5$

B nin Standart Sapması: $\sqrt{\frac{(16-15)^2+(16-10)^2+(16-35)^2+(16-5)^2+(16-15)^2}{(5-1)}} = 11{,}4$

Bu değerleri yorumlarsak, her iki öğrencinin ortalama okudukları kitap sayısı aynıdır ama A öğrencinin standart sapması düşüktür. Bu da kitap okuma konusunda daha istikrarlı olduğunu gösterir. Bunu öğrencilerin okudukları sayfa sayılarındaki değişimlere bakarak da anlayabilirdik.

GÖRÜNTÜ İŞLEME - (6.Hafta)

GÖRÜNTÜ NETLEŞTİRME ALGORİTMALARI

1. YÖNTEM: KENAR GÖRÜNTÜSÜNÜ KULLANARAK RESMİ NETLEŞTİRME

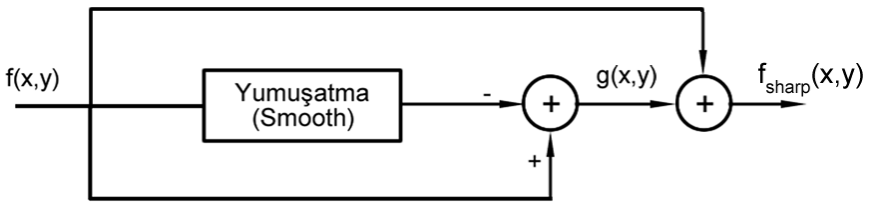
Bu algoritma orjinal görüntüden, görüntünün yumuşatılmış halini çıkararak belirgin kenarların görüntüsünü ortaya çıkarır. Daha sonra orjinal görüntü ile belirginleşmiş kenarların görüntüsünü birleştirerek, kenarları keskinleşmiş görüntüyü (netleşmiş görüntü) elde eder. Görüntünün netleştirilmesi fotogrametri 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.

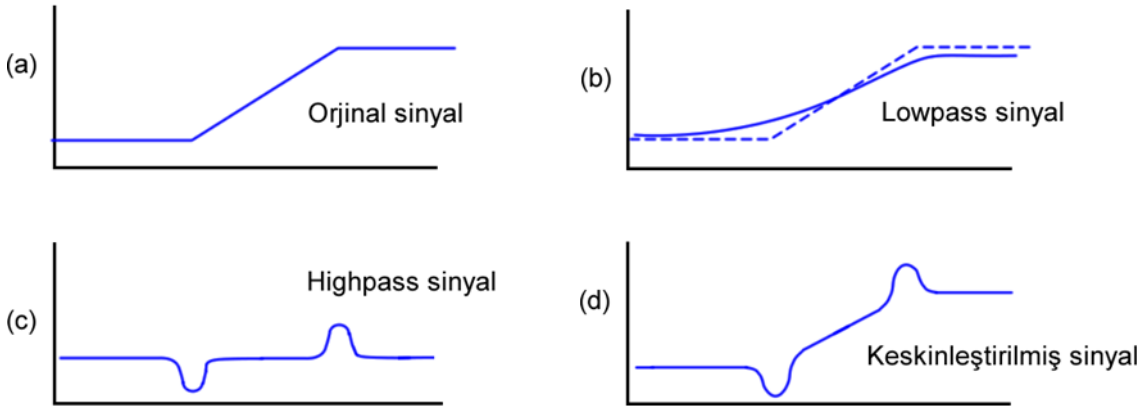
$$g(x,y) = f(x,y) - f_{smooth}(x,y)$$

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

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 aşağıda Şekil a daki gibi bir sinyal varsa, bu sinyal Şekil b'de olduğu gibi lowpass filtreden geçirilirse, yani düşük frekanslar geçirilerek resmin yumuşatılması sağlanırsa, ortaya Şekil c'deki gibi highpass görüntü yani yüksek frekansların geçtiği keskin kenarların ortaya çıktığı bir görüntü elde edilir. Bu keskin kenarlar, orjinal görüntü ile birleştirilirse netleştirilmiş (kenarları keskinlşetirilmiş) görüntü oluşur (Şekil d).



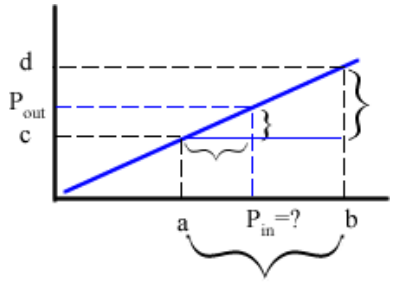
Şekil. Keskinleştirme filtresi için frekans cevabı ile kenar görüntüsü hesaplama

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.

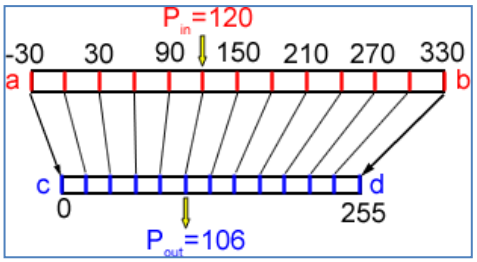
Değerleri normalize etme: Matematiksel işlemler sonunda görüntüdeki R,G,B değerleri 0-255 değerlerinin dışına çıkarsa tüm resmi aynı orantıya tabi tutarak değerleri yeniden normalize etmek gerekir.

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. Aşağıdaki resimde üçgenlerdeki benzerliği kullanarak formülü bulabiliriz.



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

Örneklersek; Keskinleştirme işlemi yapıldıktan sonra resim içindeki üst ve alt sınırlar yani a ve b değerleri -30 ile 330 değerlerini alsın. Bu takdirde sınırlarımız ve aradaki herhangi bir 120 değeri şu şekilde hesaplanır.



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

Not: Burada normalleştirme öncesi üst ve alt sınırlardan biri (a ve b değerlerinden biri) 0 ve 255 değerlerini aşmaz ise (her ikiside aşmıyorsa zaten normalleştirmeye gerekte yoktur!), çıkış resmindeki düzeltilmiş üst ve alt sınırlar olan d ve c sayıları 0 ve 255 alınamaz. Bu durumda normalleşmiş resmindeki d ve c sayıları giriş resmindeki a ve b değerleri ile aynı alınmalıdır. Diğer türlü orijinal resimde hiç beyaz yada siyah bölge yokken, çıkış resminde bu bölgeler ortaya çıkmaya başlar. (dikkat! Netleştirilen resimde her iki uçta 0-255 aşmıyorsa zaten normalleştirmeye de ihtiyaç yoktur. Sınırlardan biri 0 yada 255 aşıyorsa ve diğeri aşmıyorsa, aşmayan taraf önceki değeri olarak alınır. Yani buradaki örnekte -30 yerine bu a değeri +10 olsaydı o zaman c değeri +10 alınmalı. Bu takdirde normalleştirme için 10-330 aralığının 10-255 aralığı olarak yapılmasını gerektirir.

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 (0-255) sınırlarına çekildi. İlgili programlama (normalleştirme) ödevlerde istenmiştir.

```
      //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 = 2; //Keskin kenaları daha iyi görmek için değerini artırıyoruz.
        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 * Math.Abs(OkunanRenk1.R - OkunanRenk2.R));
                G = Convert.ToInt16(Olcekleme * Math.Abs (OkunanRenk1.G - OkunanRenk2.G));
                B = Convert.ToInt16(Olcekleme * Math.Abs (OkunanRenk1.B - OkunanRenk2.B));

                //================================================================
                //Renkler sınırların dışına çıktıysa, sınır değer alınacak. (Dikkat: Normalizasyon
yapılmamıştır. )
                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. //DİKKAT: Burada sınırı aşan değerler NORMALİZASYON yaparak programlanmalıdır.
                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 kullanarak da Netleştirme yapılabilir. Bu matriste temel mantık üzerinde işlem yapılan pikselin kenarlarındaki 4 tane piksele bakar (Köşelere bakmıyor, sıfırla çarpıyor, onları toplama katmıyor) bu piksellerin değerini aşağıya indirir, üzerinde bulunduğu pikselin değerini yukarı çıkarır. Eğer her tarafı aynı olan bir bölgede ise sonuç değişmez. Fakat bir sınır bölgesine gelirse üzerine bastığın pikselin değerini yükseltir. Böylece sınır olan yerler daha parlak gözükür. Bu anlatılanları kendiniz örnek piksel değerleri deneyin. Matris değerleri ile piksel değerleri çarpılıp toplandıktan sonra matris toplamına da bölmek gerekir (yani buradaki 1/3 sayısı bunu anlatıyor).

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

Kullanım şekli;

$$\begin{bmatrix} 1 & 1 & 5 \\ 1 & \textcircled{5} & 5 \\ 5 & 5 & 5 \end{bmatrix} \times \begin{bmatrix} 0 & -2 & 0 \\ -2 & 11 & -2 \\ 0 & -2 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -2 & 0 \\ -2 & 55 & -10 \\ 0 & -10 & 0 \end{bmatrix}$$

$$+ \quad + \quad \frac{+31}{3} = \textcircled{10}$$

$$3$$


```
        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;

            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];


                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));


        }
    }
    pictureBox2.Image =  CikisResmi;

}
```

Normalleştirme Yapılmış Kodlar

Orijinal Resim

```
    private void btnKeskinlestirmeMatris_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;

            int R, G, B;
            int[] Matris = { 0, -2, 0, -2, 11, -2, 0, -2, 0 };
            int MatrisToplami = 3;

            int EnBuyukR = 0;
            int EnBuyukG = 0;
            int EnBuyukB = 0;

            int EnKucukR = 0;
            int EnKucukG = 0;
            int EnKucukB = 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];


                            k++;
                        }

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

                    //==========================================================
                    //Renkler sınırların dışına çıktıysa, sınır değer alınacak.


                    if (R > 255)
                    {
                        if (EnBuyukR < R)
                            EnBuyukR = R;
                        //R = 255;

                    }
                    if (G > 255)
```

```
                {
                    if (EnBuyukG < G)
                        EnBuyukG = G;
                    //G = 255;
                }


                if (B > 255)
                {
                    if (EnBuyukB < B)
                        EnBuyukB = B;
                    //B = 255;
                }

                //------------------------
                if (R < 0)
                {
                    if (EnKucukR > R)
                        EnKucukR = R;
                    //R = 0;

                }
                if (G < 0)
                {
                    if (EnKucukG > G)
                        EnKucukG = G;
                    //G = 0;
                }


                if (B < 0)
                {
                    if (EnKucukB > B)
                        EnKucukB = B;
                    //B = 0;
                }
                //============================================================

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


            }
        }
        txtX1.Text  = EnBuyukR.ToString();
        txtX2.Text = EnBuyukG.ToString();
        txtX3.Text = EnBuyukB.ToString();

        txtY1.Text = EnKucukR.ToString();
        txtY2.Text = EnKucukG.ToString();
        txtY3.Text = EnKucukB.ToString();

        int EnBuyuk=0, EnKucuk=0;

        if (EnBuyukR > EnBuyuk)
            EnBuyuk = EnBuyukR;

        if (EnBuyukG > EnBuyuk)
            EnBuyuk = EnBuyukG;

        if (EnBuyukB > EnBuyuk)
            EnBuyuk = EnBuyukB;


        if (EnKucukR > EnKucuk)
            EnKucuk = EnKucukR;
```

```
            if (EnKucukG > EnKucuk)
                EnKucuk = EnKucukG;

            if (EnKucukB > EnKucuk)
                EnKucuk = EnKucukB;


            Bitmap NormallesmisNetResim =  ResmiNormallestir(GirisResmi, SablonBoyutu,
ResimGenisligi, ResimYuksekligi, EnBuyuk, EnKucuk);

            pictureBox4.Image = NormallesmisNetResim;        }
```

```
        public Bitmap ResmiNormallestir(Bitmap GirisResmi, int SablonBoyutu, int ResimGenisligi,
int ResimYuksekligi, int EnBuyuk, int EnKucuk)
        {
            int toplamR = 0;
            int toplamG = 0;
            int toplamB = 0;

            Bitmap CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

            int x, y, i, j;

            int R, G, B;
            int[] Matris = { 0, -2, 0, -2, 11, -2, 0, -2, 0 };
            int MatrisToplami = 3;

            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++)
                        {
                            Color 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];


                            k++;
                        }

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

                    //NORMALİZASYON-----------------------
                    int YeniR = (255 * (R - EnKucuk)) / (EnBuyuk - EnKucuk) ;
                    int YeniG = (255 * (G - EnKucuk)) / (EnBuyuk - EnKucuk) ;
                    int YeniB = (255 * (B - EnKucuk)) / (EnBuyuk - EnKucuk) ;
```

```
                if (YeniR > 255) YeniR = 255;
                if (YeniG > 255) YeniG = 255;
                if (YeniB > 255) YeniB = 255;

                if (YeniR < 0) YeniR = 0;
                if (YeniG < 0) YeniG = 0;
                if (YeniB < 0) YeniB = 0;
                //=========================================================

                CikisResmi.SetPixel(x, y, Color.FromArgb(YeniR, YeniG, YeniB));

            }
        }

        return CikisResmi;
    }
```

Ödev 1: Netleştirme işleminde resmi bulanıklaştırırken kullanılan buradaki Mean yönteminin dışında Median ve Gauss yöntemlerini kullanarakda netleştirme yapın.

Ödev 2: Bulanık bir resmi netleştirirken daha büyük ölçekli matrislere ihtiyaç olacaktır. Böyle bir resim üzerinde kullanılan matris boyutunu aşama aşama artırarak resmin netliğinde nasıl bir etkisi olduğunu gösterin.

Ödev 3: Aşağıdaki matrislerden hangisi kenar bulma matrisi, hangisi netleştirme matrisidir? Programlayarak deneyin. Deneme öncesi kağıt üzerinde kendiniz örnek değerler vererek bulmaya çalışın.

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

0	-2	0
-2	18	-2
0	-2	0

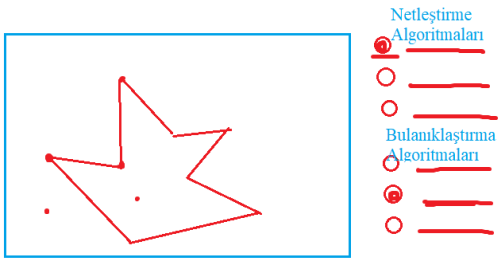
1	-2	1
-2	4	-2
1	-2	1

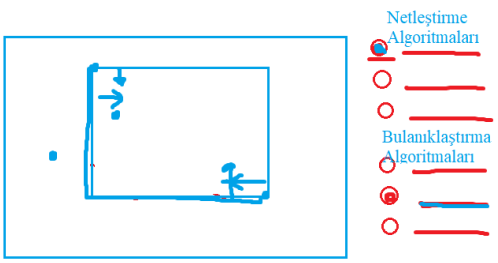
-1	-1	-1
-1	8	-1
-1	-1	-1

Ödev 4: Yukarıdaki tüm uygulamaları Normalizasyon yaparak düzenleyin. Eğer sınırlar 0-255 dışına çıkıyorsa ve elimizdeki resimde hiç beyaz yada siyah yoksa yani (30 – 230) gibi bir değerde ise Dışarı çıkan bu sınırlar resmin en üç noktaları arasında normalize edilmelidir.

Ödev 5: Bir resmin üzerindeki bir bölgeyi seçmek için köşe noktalarına mouse tıklayalım. Tıkladığımız her noktanının arasına bir çizgi çizsin. Tüm bölgeyi kapattıktan sonra, çerçevenin iç bölgesinde netleştirme uygulasın, dış bölgesinde ise Bulanıklaştırma uygulasın. Hangi algoritma ile netleştirme, hangi ile bulanıklaştırma uygulayacağını ise yan taraftaki menüden kişi seçebilsin. Bunun için RadioButton seçeneği kullanılabilir.

Çoklu nokta şeklinde yapamayanlar, dikdörtgen çerçeve şeklinde bölgeyi seçsinler. Bu şekilde yapanlar daha düşük puan alır.

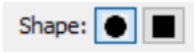




Ödev 6: Sharpen (netleştirme) aracı kullanarak bir resim üzerinde mouse takip eden bulur işlemi yaptırın. Bunun için aşağıdaki adımları takip edin.

a) Mouse ın etrafında büyüklüğü ayarlanabilir belli bir bölge olsun. Örneğin açılır menüden 50 piksel seçilirse mouuse etrafında 50x50 lik kare bir şekilde dolaşsın. Bu alanın içi netleşsin.

b) Kişi ister bu şekli Kare yada Daire olarak değiştirebilsin. Bu bölge Kare yada Daire olarak seçilebilsin. Bu bölgenin içerisi mouse hareket ettikçe, bulanıklaşsın.



c) Ayrıca bu aracın kenarları üzerinde geçişli bir netlik oluşturmak için bir ayar daha eklensin. Örneğin burada %100 seçilirse Dairenin merkezinden kenara kadar yavaş yavaş netlik azalarak gitsin. %50 seçilirse darenin yarısından sonra yavaş yavaş netlik azalsın.

d) Netliğin yoğunluğunun ayarlanabildiği bir seçenek daha ekleyin. Aşağıda Intensity olarak gösterilmiş.

e) Kullanılan algoritma açılır menüden, yada radyo buton listesinden seçilebilsin. Örneğin; Kenar bulma algoritması yada Matris algoritması gibi.