GÖRÜNTÜ İŞLEME-10.HAFTA

İçindekiler

MANTIKSAL OPERATÖRLER

Mantıksal operatörler, resmin gerçek değerlerine dokunmadan, boolean ifadeler (true yada false) kullanılarak yeni bilgiler veya görüntüler çıkarmak için kullanılır. Örneğin "Ağaç hem yeşil hemde büyük ise" ifadesi A AND B şeklinde ifade edilebilir. Bu durumda her iki kavramda doğru (true) ise sonuç True olacaktır ve ona göre işlem yapılacaktır. Bu şekilde girilen iki tane görüntü üzerinde yada tek bir görüntü üzerinde, piksel değerlerine mantıksal operatörlerin doğruluk tablosu kuralları uygulanarak görüntüler üzerinde mantıksal işlemler gerçekleştirebiliriz. Normalde, aynı boyutta iki görüntüden çıktı görüntüsünü üretmek için, giriş görüntüsülerine ait birbirine karşılık gelen pikseller karşılaştırılır. Tek bir giriş görüntüsünü sabit bir mantıksal değerle mantıksal olarak birleştirmek de mümkündür. Bu durumda giriş görüntüsündeki her piksel karşılık gelen çıkış pikselini üretmek için aynı sabit ile karşılaştırılır.

Mantıksal işlemler, tamsayı piksel değerlerine sahip görüntüler üzerinde, bu tamsayıların ikili gösterimleri (binary) yapılarak bunlar üzerinde bitsel bir şekilde işlem yapılarak, çıkış piksel değerine bitsel sonuçlar üretilip buradan tam sayı değerine ulaşılabilir. Örneğin, 47 ve 255 tam sayılarını 8 bitlik tam sayıları kullanarak birlikte XOR yapmak istediğimizi varsayalım. Bunları ikili biçimde göserirsek 45 sayısı 00101111 ve 255 sayısı 11111111'dir. Bunları bitsel olarak birlikte XORing yaparak, ikili olarak 11010000 değerine ulaşılır. Bunun karşılığı ise tam sayı olarak 208 dir.

Mantıksal işlemler illa böyle bitsel (1/0) olarak uygulanması gerekmez. Örneğin, bazı uygulamalarda sıfır değerleri false kabul edilirken, sıfır haricindeki herhangi bir değer true olarak kabul edilebilir. Çıktı görüntüsünü elde etmek için ise 1-bit mantıksal işlemleri uygulanır. Çıktı görüntüsü basit bir ikili (binary) görüntünün kendisi olabilir ya da belki de ikili (binar) çıktı görüntüsü (sadece siyah ve beyaz görüntü) giriş görüntülerinden hehangi biriyle çarpılarak gri seviye bir görüntü elde edilebilir..

İnteger sayıyı ikili (binary) sayıya 8 bit olarak dönüştürme


```
int intSayi1 = Convert.ToInt16(txtDeger1.Text);

string binarySayi = Convert.ToString(intSayi1, 2).PadLeft(8, '0');

txtDeger1.Text = binarySayi;

int intSayi2 = Convert.ToInt32(binarySayi, 2);

txtDeger2.Text = intSayi2.ToString();
```

Mantıksal AND/NAND Operatörleri

Mantıksal operatörler, Boolean değerler üzerinde (1 yada 0)(Doğru yada Yanlış) işlem yapılarak elde edilir. AND ve NAND işlem tablosu aşağıdaki gibidir.

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

AND

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

NAND

Dikkat edilirse iki tablo birbirinin tam zıttıdır. Bu Operatörlerde Birinci resmi alıp, ikinci resimle AND veya NAND işlemine tabi tutularak çıktı resmi oluşturulur. Yada sadece tek bir Girdi resmi alınır ve belli bir sabit sayı ile AND yada NAND işlemine tabi tutulabilir. Tablolara dikkat edilirse bu iki operatör birbirinin tam tersidir.

AND ve NAND operatörü giriş olarak iki adet ikili (binary-siyah/beyaz resim) veya tamsayı ile ifade edilen gri seviye görüntüyü alır ve piksel değerleri üzerinde işlem yaparak ilk görüntünün değerlerini ikinci görüntü ile işlem yaparak üçüncü bir görüntü oluşturulur. Bu operatörün farklı bir uygulaması olarak tek bir giriş görüntüsü alınıp her pikseli sabit bir değer ile işleme tutarak çıktı görüntüsü oluşturulabilir.

İşlem tek bir geçişte doğrudan gerçekleştirilir. Üzerinde çalışılan tüm giriş pikseli değerlerinin aynı sayıda bite sahip olması önemlidir. Giriş görüntülerindeki piksel değerlerinin basit 1 bitlik sayılar olmadığı durumlarda, AND işlemi normalde (ancak her zaman değil) piksel değerlerindeki her bir bit üzerinde ayrı ayrı, bitsel şekilde gerçekleştirilir.

AND/NAND operatörü nerede kullanılır?

Görüntü işleme alanında, AND operatörü genellikle görüntüler arasında piksel bazında birleştirme, maskeleme veya sınırlama gibi işlemlerde kullanılır. İşte bazı örnekler:

Piksel birleştirme: İki görüntü arasında piksel bazında birleştirme yapmak için AND operatörü kullanılabilir. Bu işlemde, iki görüntünün aynı konumdaki pikselleri AND operatörü ile birleştirilir ve sonuç görüntüsünde yalnızca ortak olan piksellerin değeri korunur.

Maskeleme: Bir görüntüyü başka bir görüntüyle maskelemek için AND operatörü kullanılabilir. Maskeleme işlemi, bir maske görüntüsünün belirli bir değeri olan pikselleri korurken, diğer pikselleri göz ardı etme işlemidir. AND operatörü, görüntüyü maskeleme işleminde kullanılan maske görüntüsüyle birleştirerek istenilen piksellerin sınırlanmasını sağlar.

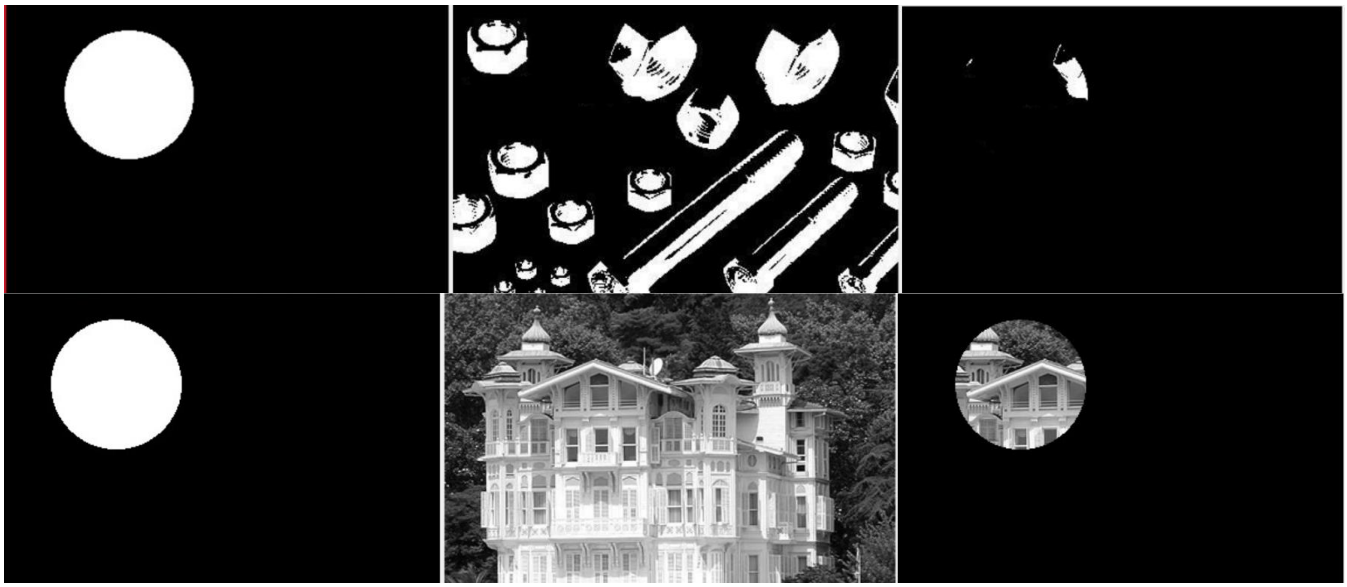
Sınırlama: Görüntüdeki pikselleri belirli bir değer aralığıyla sınırlamak için AND operatörü kullanılabilir. Örneğin, bir görüntüdeki piksellerin belirli bir aralıkta (örneğin, minimum ve maksimum değerler arasında) olmasını istiyorsak, bu sınırlama işlemi için AND operatörü kullanılabilir. AND operatörü, piksellerin değerlerini sınırlama aralığıyla birleştirerek istenen sınırlamayı uygular.

Bu örnekler, görüntü işleme uygulamalarında AND operatörünün yaygın olarak kullanıldığı bazı senaryoları göstermektedir. Ancak, bu sadece bazı örneklerdir ve AND operatörü, diğer görüntü işleme işlemlerinde de farklı şekillerde kullanılabilir. Diğer mantıksal işleçlerde olduğu gibi, AND ve NAND genellikle daha karmaşık görüntü işleme görevlerinin alt bileşenleri olarak kullanılır.

```
private void AND_NAND_IslemiToolStripMenuItem_Click(object sender, EventArgs e)
{
    Bitmap Resim1, Resim2, CikisResmi;
    Resim1 = new Bitmap(pictureBox1.Image);
    Resim2 = new Bitmap(pictureBox2.Image);

    int ResimGenisligi = Resim1.Width;
    int ResimYuksekligi = Resim1.Height;

    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

    Color Renk1, Renk2;
    int x, y;
    int R = 0, G = 0, B = 0;

    for (x = 0; x < ResimGenisligi; x++) //Resmi taramaya şablonun yarısı kadar dış kenarlardan
içeride başlayacak ve bitirecek.
    {
        for (y = 0; y < ResimYuksekligi; y++)
        {
            Renk1 = Resim1.GetPixel(x, y);
            Renk2 = Resim2.GetPixel(x, y);

            string binarySayi1 = Convert.ToString(Renk1.R, 2).PadLeft(8, '0'); //Gri renk
olduğundan tek kanal üzerinden yapılıyor.
            string binarySayi2 = Convert.ToString(Renk2.R, 2).PadLeft(8, '0');

            string Bit1 = null, Bit2 = null, StringIkiliSayi = null;

            for (int i = 0; i < 8; i++)
            {
                Bit1 = binarySayi1.Substring(i, 1);
                Bit2 = binarySayi2.Substring(i, 1);

                ////AND İŞLEMİ
                //if (Bit1 == "0" && Bit2 == "0") StringIkiliSayi = StringIkiliSayi + "0";
                //else if (Bit1 == "1" && Bit2 == "1") StringIkiliSayi = StringIkiliSayi + "1";
                //else StringIkiliSayi = StringIkiliSayi + "0";

                //NAND İŞLEMİ
                if (Bit1 == "0" && Bit2 == "0") StringIkiliSayi = StringIkiliSayi + "1";
                else if (Bit1 == "1" && Bit2 == "1") StringIkiliSayi = StringIkiliSayi + "0";
                else StringIkiliSayi = StringIkiliSayi + "1";
```

```
            }
            R = Convert.ToInt32(StringIkiliSayi, 2); //İkili sayıyı tam sayıya dönüştürüyor.

            CikisResmi.SetPixel(x, y, Color.FromArgb(R, R, R)); //Gri resim

        }
    }
    pictureBox3.Image = CikisResmi;
```

AND için oluşan çıktı.



NAND için oluşan çıktı.

Renkli Resim için Kodlar

```
private void MANTIKSAL_OPERATOR_Click(object sender, EventArgs e)
{

Bitmap Resim1, Resim2, CikisResmi;
Resim1 = new Bitmap(pictureBox1.Image);
Resim2 = new Bitmap(pictureBox2.Image);

int ResimGenisligi = Resim1.Width;
int ResimYuksekligi = Resim1.Height;


CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

Color Renk1, Renk2;
int x, y;
int R = 0, G = 0, B = 0;

for (x = 0; x < ResimGenisligi; x++) //Resmi taramaya şablonun yarısı kadar dış kenarlardan
içeride başlayacak ve bitirecek.
```

```
{
    for (y = 0; y < ResimYuksekligi; y++)
    {
        Renk1 = Resim1.GetPixel(x, y);
        Renk2 = Resim2.GetPixel(x, y);

        string binarySayi1R = Convert.ToString(Renk1.R, 2).PadLeft(8, '0'); //Gri renk
olduğundan tek kanal üzerinden yapılıyor.
        string binarySayi2R = Convert.ToString(Renk2.R, 2).PadLeft(8, '0');

        string binarySayi1G = Convert.ToString(Renk1.G, 2).PadLeft(8, '0'); //Gri renk
olduğundan tek kanal üzerinden yapılıyor.
        string binarySayi2G = Convert.ToString(Renk2.G, 2).PadLeft(8, '0');

        string binarySayi1B = Convert.ToString(Renk1.B, 2).PadLeft(8, '0'); //Gri renk
olduğundan tek kanal üzerinden yapılıyor.
        string binarySayi2B = Convert.ToString(Renk2.B, 2).PadLeft(8, '0');

        string Bit1R = null, Bit1G = null, Bit1B = null, Bit2R = null, Bit2G = null, Bit2B =
null;
        string StringIkiliSayiR = null, StringIkiliSayiG = null, StringIkiliSayiB = null;

        for (int i = 0; i < 8; i++)
        {
            Bit1R = binarySayi1R.Substring(i, 1);
            Bit2R = binarySayi2R.Substring(i, 1);

            //AND İŞLEMİ
            if (Bit1R == "0" && Bit2R == "0") StringIkiliSayiR = StringIkiliSayiR + "0";
            else if (Bit1R == "1" && Bit2R == "1") StringIkiliSayiR = StringIkiliSayiR + "1";
            else StringIkiliSayiR = StringIkiliSayiR + "0";

            Bit1G = binarySayi1G.Substring(i, 1);
            Bit2G = binarySayi2G.Substring(i, 1);

            //AND İŞLEMİ
            if (Bit1G == "0" && Bit2G == "0") StringIkiliSayiG = StringIkiliSayiG + "0";
            else if (Bit1G == "1" && Bit2G == "1") StringIkiliSayiG = StringIkiliSayiG + "1";
            else StringIkiliSayiG = StringIkiliSayiG + "0";

            Bit1B = binarySayi1B.Substring(i, 1);
            Bit2B = binarySayi2B.Substring(i, 1);

            //AND İŞLEMİ
            if (Bit1B == "0" && Bit2B == "0") StringIkiliSayiB = StringIkiliSayiB + "0";
            else if (Bit1B == "1" && Bit2B == "1") StringIkiliSayiB = StringIkiliSayiB + "1";
            else StringIkiliSayiB = StringIkiliSayiB + "0";



        }
        R = Convert.ToInt32(StringIkiliSayiR, 2); //İkili sayıyı tam sayıya dönüştürüyor.
        G = Convert.ToInt32(StringIkiliSayiG, 2); //İkili sayıyı tam sayıya dönüştürüyor.
        B = Convert.ToInt32(StringIkiliSayiB, 2); //İkili sayıyı tam sayıya dönüştürüyor.

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

    }
}
pictureBox3.Image = CikisResmi;
}
```

Çıktısı: İki resimdeki yüksek değerli (255) ortak bölgeyi göstermiş oldu. Birinde yüksek diğerinde düşükse onu yok etti. Her ikisi de düşük olunca zaten yok oldu. Zaten AND operatörünü inceleyecek olursak tablo ile aynı mantığı burada verdi. Resimler iki renkli olduğu için bu yorumu yapmak kolay oldu, fakat renkli resimlerde bu kadar değişimi görmek kolay olmayacaktır .

C# Kendi Fonksiyonları ile AND operatörünü kullanma

Aşağıdaki programda AND operatörünü C# kendi fonksiyonları ile nasıl yazıldığını göreceksiniz.

```
private void btnDENEME_Click(object sender, EventArgs e)
{
    Bitmap Resim1 = new Bitmap(pictureBox1.Image);
    Bitmap Resim2 = new Bitmap(pictureBox2.Image);

    // Görüntülerin boyutlarını kontrol et
    if (Resim1.Width != Resim2.Width || Resim1.Height != Resim2.Height)
    {
        MessageBox.Show("Görüntüler aynı boyutta değil!");
        return;
    }

    // Sonuç görüntüsü için yeni bir bitmap oluştur
    Bitmap SonucGoruntusu = new Bitmap(Resim1.Width, Resim1.Height);

    // Görüntülerin piksellerini dolaşarak AND operatörü uygula
    for (int x = 0; x < Resim1.Width; x++)
    {
        for (int y = 0; y < Resim1.Height; y++)
        {
            // İki görüntünün piksellerini al
            Color renk1 = Resim1.GetPixel(x, y);
            Color renk2 = Resim2.GetPixel(x, y);

            // Piksellerin R, G ve B bileşenlerini AND operatörü ile birleştir
            Color sonucRenk = Color.FromArgb(
                renk1.R & renk2.R,
                renk1.G & renk2.G,
                renk1.B & renk2.B
            );
           // Piksellerin R, G ve B bileşenlerini NAND operatörü ile birleştir (NEGATİF SAYI
HATASI VERİYOR)
            //Color sonucRenk = Color.FromArgb(
            //    ~(renk1.R & renk2.R),
            //    ~(renk1.G & renk2.G),
            //    ~(renk1.B & renk2.B)
            //);

            // Piksellerin R, G ve B bileşenlerini OR operatörü ile birleştir
            //Color sonucRenk = Color.FromArgb(
            //    renk1.R | renk2.R,
            //    renk1.G | renk2.G,
            //    renk1.B | renk2.B
            //);

            // Sonuç görüntüsüne pikseli yerleştir
```

```
            SonucGoruntusu.SetPixel(x, y, sonucRenk);
        }
    }

    // Sonuç görüntüsünü göster
    pictureBox3.Image = SonucGoruntusu;

}
```

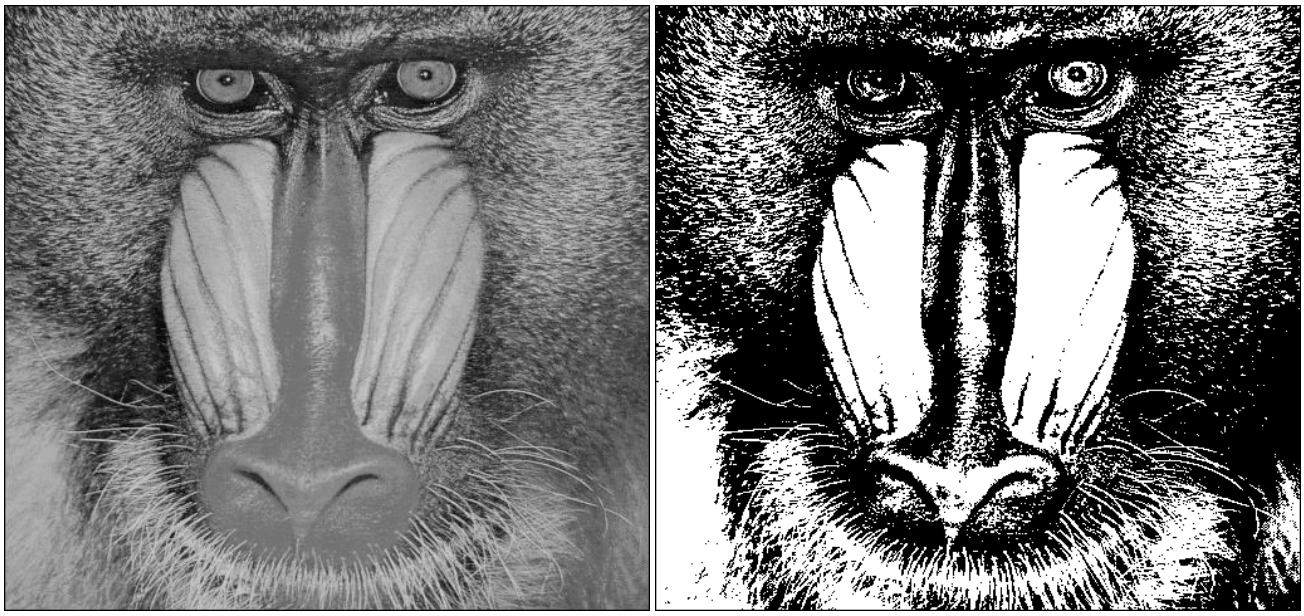
Uygulama 1-İki görüntünün kesişimini bulma

Uygulama 2-Görüntülerden birini maske olarak kullanma

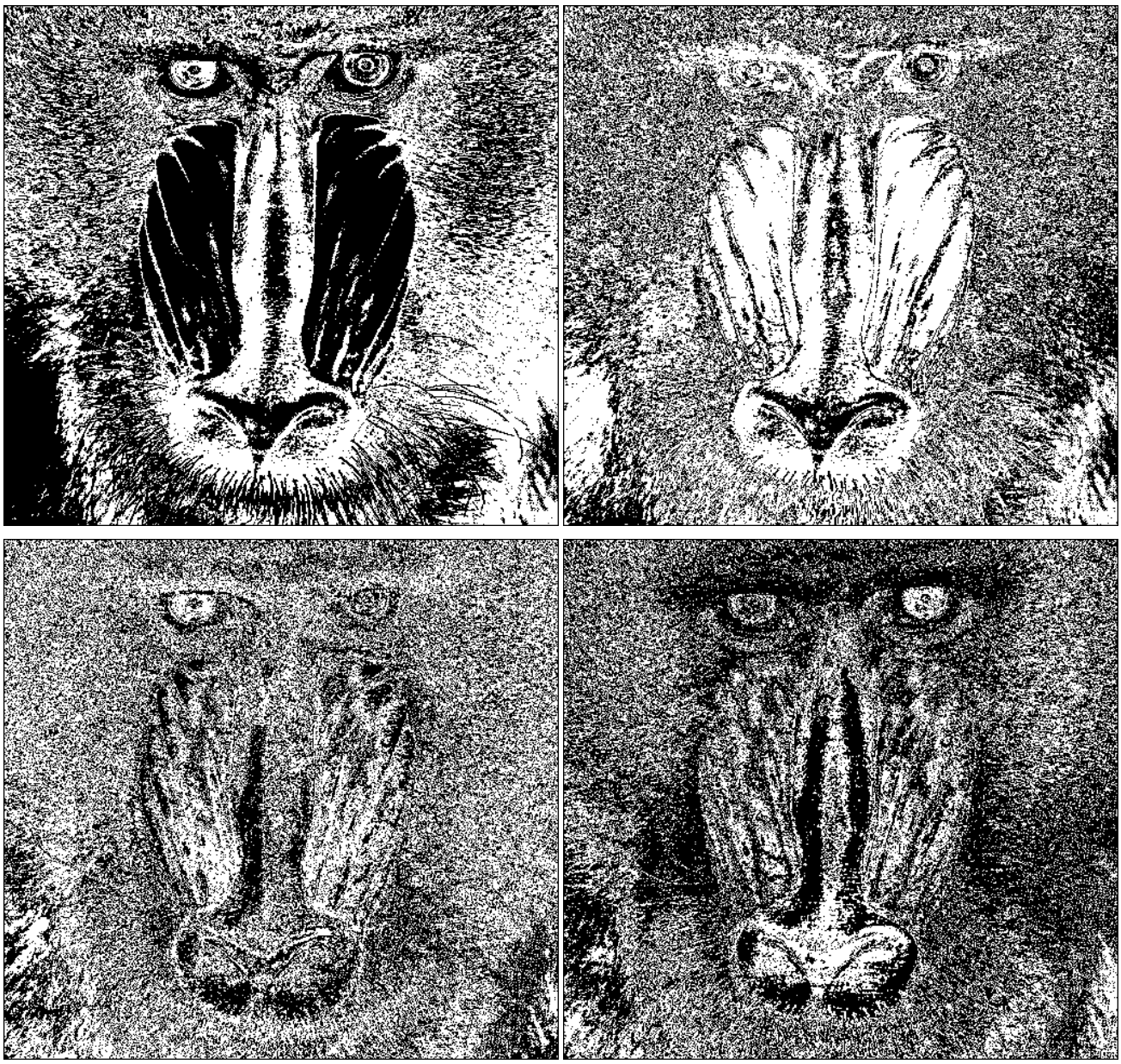
BİT DİLİMLEME

AND operatörü, 8 bitlik bir görüntüde "**bit dilimleme**" adı verilen işlemi gerçekleştirmek için de kullanılabilir. Belirli bir bitin görüntü üzerindeki etkisini belirlemek için, ilgili bitin 1'e ve geri kalan 7 bitin 0'a ayarlanır. sabit bir sayı ile bitsel bir şekilde AND işlemi uygulanır.

Örneğin, görüntünün 8. Bitini (en önemli bite karşılık gelir) işlemi uygularsak ve AND işlemine tabi tuttuktan sonra 128 değeri ile eşiklersek arşağıdaki ikinci görüntüyü elde ederiz.



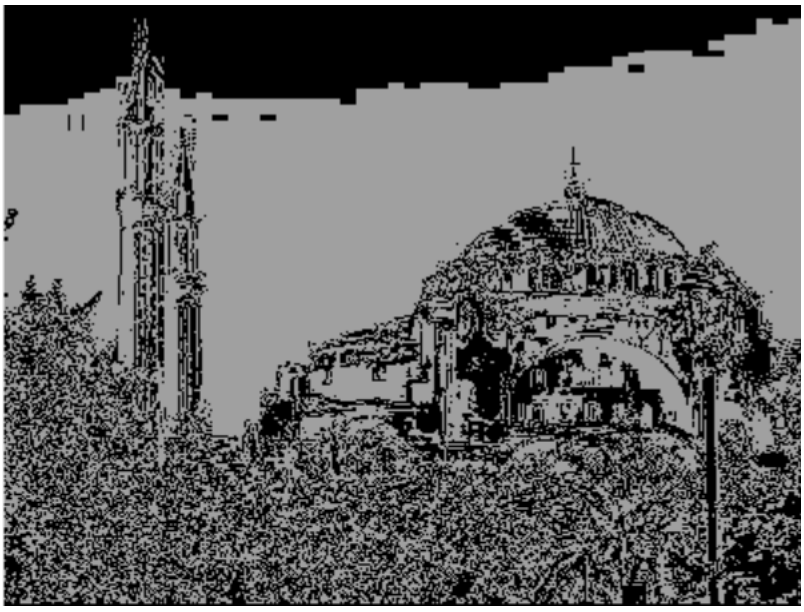
Aşağıdaki resimler sırasıyla görüntünün 7., 6., 4., 1. bit düzlemlerine uygulanan işlemlere karşılık gelir. Görüntü bilgisinin daha çok yüksek değerli bitlerde olduğu görülecektir. Daha az anlamlı bitler daha ince detayları yada paraziti içerir.



Orijinal resim ve 1 kat ölçeklenmiş hali. 0 bit 1x ölçekleme

1. bit 1x ölçekleme ve 3x ölçekleme

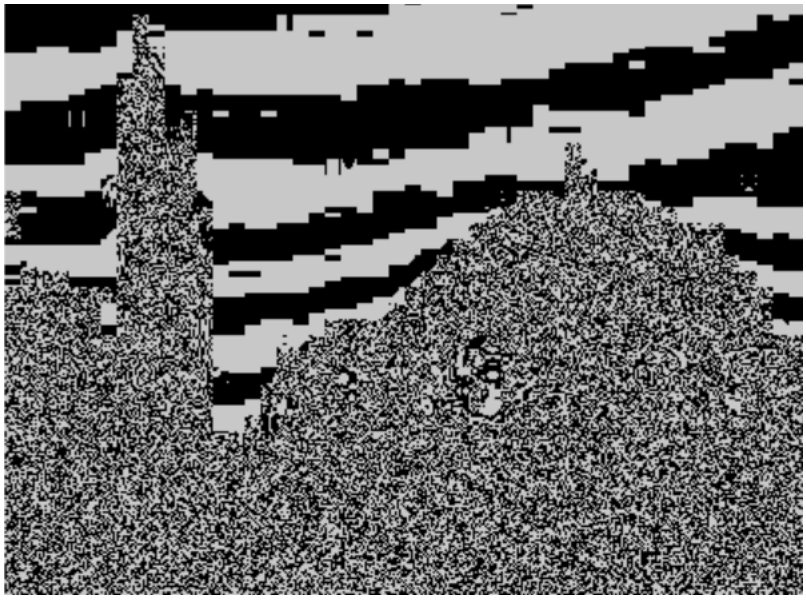
2. bit 5x öçekleme



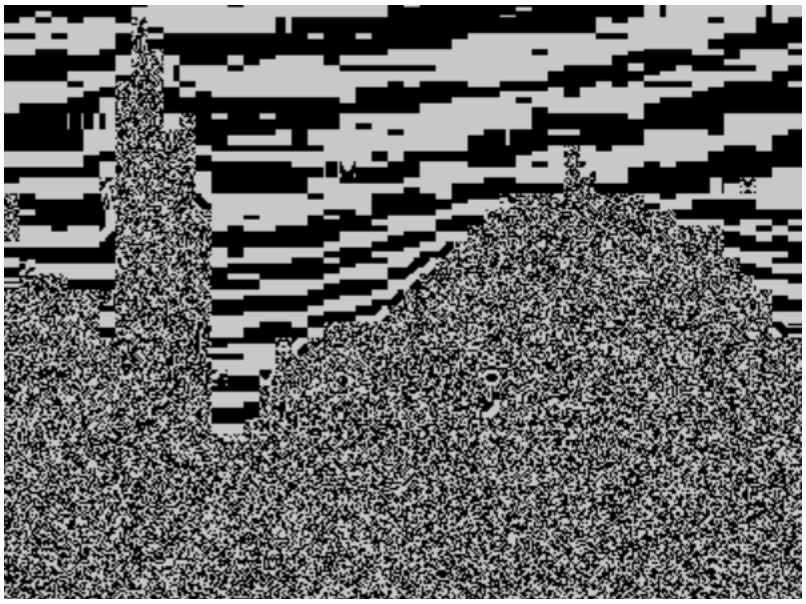
3 Bit 10x ölçekleme

4 bit 20x

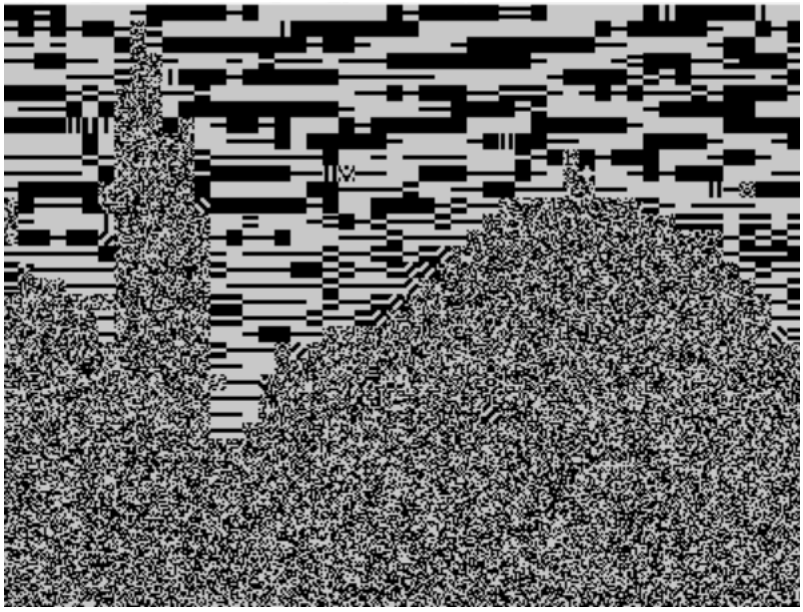
5 bit 50x



6 bit 100x



7 200x



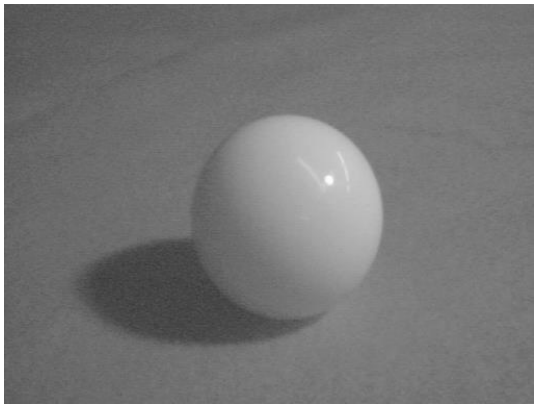
```
private void BIT_DILIMLEME_Click(object sender, EventArgs e)
{

    Bitmap Resim1, CikisResmi;
    Resim1 = new Bitmap(pictureBox1.Image);

    int ResimGenisligi = 400;
    int ResimYuksekligi = 250;


    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

    Color Renk1;
    int x, y;
    int R = 0, G = 0, B = 0;

    for (x = 0; x < ResimGenisligi; x++) //Resmi taramaya şablonun yarısı kadar dış kenarlardan
içeride başlayacak ve bitirecek.
    {
        for (y = 0; y < ResimYuksekligi; y++)
        {
            Renk1 = Resim1.GetPixel(x, y);
```

```
            string binarySayi1R = Convert.ToString(Renk1.R, 2).PadLeft(8, '0'); //Gri renk
olduğundan tek kanal üzerinden yapılıyor.

            string binarySayi1G = Convert.ToString(Renk1.G, 2).PadLeft(8, '0'); //Gri renk
olduğundan tek kanal üzerinden yapılıyor.

            string binarySayi1B = Convert.ToString(Renk1.B, 2).PadLeft(8, '0'); //Gri renk
olduğundan tek kanal üzerinden yapılıyor.

            string Bit1R = null, Bit1G = null, Bit1B = null;
            string StringIkiliSayiR = null, StringIkiliSayiG = null, StringIkiliSayiB = null;

            int BitSiraNo = Convert.ToInt32(textBox1.Text);

            for (int i = 0; i < 8; i++)
            {
                if (i == BitSiraNo)
                {
                    Bit1R = binarySayi1R.Substring(i, 1);
                    StringIkiliSayiR = StringIkiliSayiR + Bit1R;

                    Bit1G = binarySayi1G.Substring(i, 1);
                    StringIkiliSayiG = StringIkiliSayiG + Bit1G;

                    Bit1B = binarySayi1B.Substring(i, 1);
                    StringIkiliSayiB = StringIkiliSayiB + Bit1B;

                }
                else
                {
                    StringIkiliSayiR = StringIkiliSayiR + "0";
                    StringIkiliSayiG = StringIkiliSayiG + "0";
                    StringIkiliSayiB = StringIkiliSayiB + "0";
                }

            }
            R = Convert.ToInt32(StringIkiliSayiR, 2); //İkili sayıyı tam sayıya dönüştürüyor.
            G = Convert.ToInt32(StringIkiliSayiG, 2); //İkili sayıyı tam sayıya dönüştürüyor.
            B = Convert.ToInt32(StringIkiliSayiB, 2); //İkili sayıyı tam sayıya dönüştürüyor.

            int Olcek = Convert.ToInt32(textBox2.Text);

            R = R * Olcek;
            G = G * Olcek;
            B = B * Olcek;

            if (R > 255) R = 255;
            if (G > 255) G = 255;
            if (B > 255) B = 255;

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

        }
    }
    pictureBox3.Image = CikisResmi;

}
```




Sabit Sayı ile AND leme

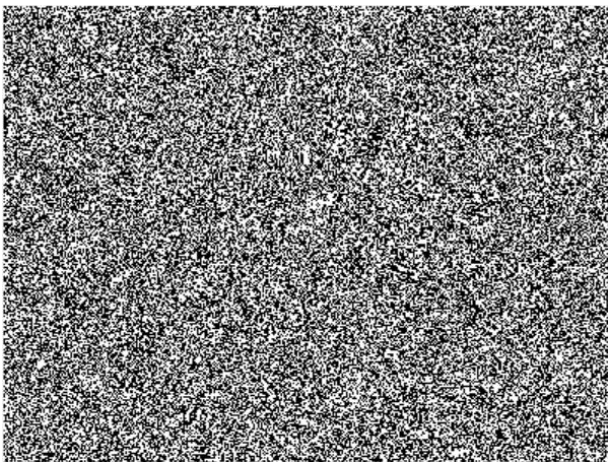
Aşağıdaki resmi çeşitli **sabit sayılar ile AND işlemine** tabi tutarsak şu sonuçları elde ederiz. Sayı küçük iken görüntü koyu olduğu için görebilmek için ölçeklemek gerekmiştir.



1 sayısı ile AND leme ve 1x Ölçekleme

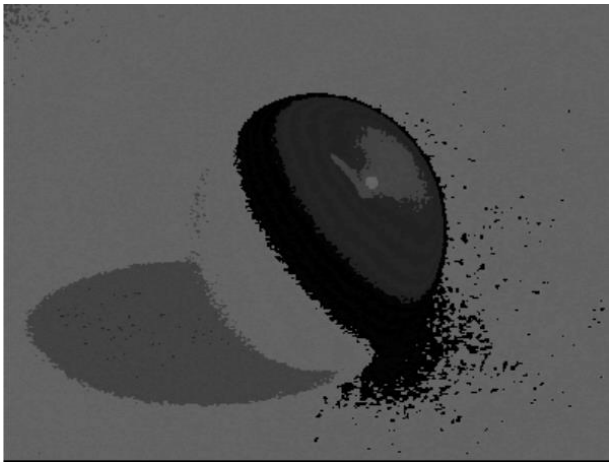
1 sayısı ile AND leme ve 255x Ölçekleme

15 sayısı ile AND leme ve 10x Ölçekleme

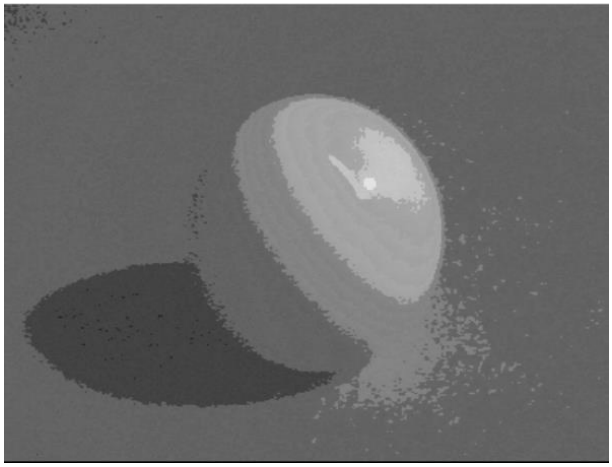


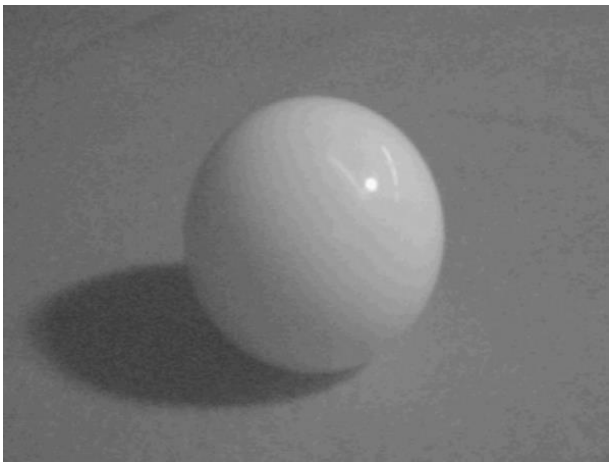
100 sayısı ile AND leme ve 1x Ölçekleme

230 sayısı ile AND leme ve 1x Ölçekleme



250 sayısı ile AND leme ve 1x Ölçekleme

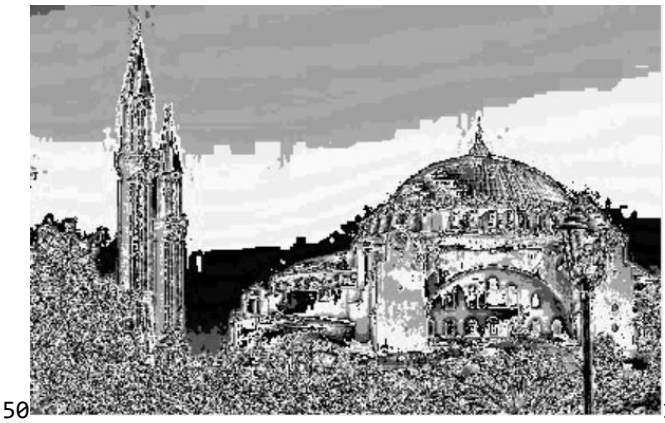


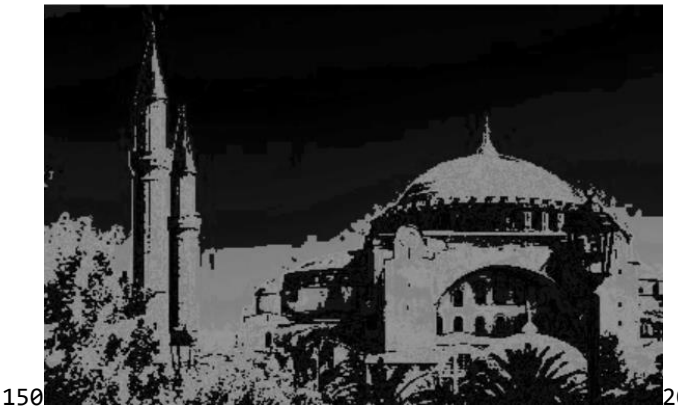


Benzer şekilde 0 sayısı ile AND leme yaparsak tamamen siyah resim, 255 sayısı ile AND leme yaparsak tamamen Orijinal resmi elde ederiz.

```
private void ANDSabitSayiToolStripMenuItem_Click(object sender, EventArgs e)
{
    Bitmap Resim1, Resim2, CikisResmi;
    Resim1 = new Bitmap(pictureBox1.Image);

    int ResimGenisligi = Resim1.Width;
    int ResimYuksekligi = Resim1.Height;
```

```
    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

    Color Renk1;
    int x, y;
    int R = 0, G = 0, B = 0;

    for (x = 0; x < ResimGenisligi; x++) //Resmi taramaya şablonun yarısı kadar dış kenarlardan
içeride başlayacak ve bitirecek.
    {
        for (y = 0; y < ResimYuksekligi; y++)
        {
            Renk1 = Resim1.GetPixel(x, y);

            string binarySayi1 = Convert.ToString(Renk1.R, 2).PadLeft(8, '0'); //Gri renk
olduğundan tek kanal üzerinden yapılıyor.

            int SabitSayi = Convert.ToInt32(txtDeger1.Text);
            string binarySayi2 = Convert.ToString(SabitSayi, 2).PadLeft(8, '0');

            string Bit1 = null, Bit2 = null, StringIkiliSayi = null;

            for (int i = 0; i < 8; i++)
            {
                Bit1 = binarySayi1.Substring(i, 1);
                Bit2 = binarySayi2.Substring(i, 1);

                //AND İŞLEMİ
                if (Bit1 == "0" && Bit2 == "0") StringIkiliSayi = StringIkiliSayi + "0";
                else if (Bit1 == "1" && Bit2 == "1") StringIkiliSayi = StringIkiliSayi + "1";
                else StringIkiliSayi = StringIkiliSayi + "0";

            }
            R = Convert.ToInt32(StringIkiliSayi, 2); //İkili sayıyı tam sayıya dönüştürüyor.

            int Olcek = Convert.ToInt32(txtDeger2.Text);
            R = R * Olcek;

            //Sınırı aşan değerleri 255 ayarlama
            if (R > 255) R = 255;
            if (G > 255) G = 255;
            if (B > 255) B = 255;

            CikisResmi.SetPixel(x, y, Color.FromArgb(R, R, R)); //Gri resim

        }
    }
    pictureBox2.Image = CikisResmi;

}
```





Mantıksal OR/NOR Operatörleri

Doğrulama tabloları aşağıda gösterilmiştir. Görüleceği gibi her ikisi birbirinin tam tersidir. Her iki operatörde yukarıda olduğu gibi iki tane gri seviye resmi alıp bundan üçüncü bir resim üretir. Okunan her gri renk değeri 8 bit ikili sayı sistemine çevrilerek karşılıklı bitler OR yada NOR işlemine tabi tutulur.

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

OR

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

NOR

OR/NOR operatörü nerede kullanılır?

Görüntü işleme alanında, OR operatörü genellikle görüntüler arasında piksel bazında birleştirme veya bölge genişletme gibi işlemlerde kullanılır. İşte bazı örnekler:

Piksel birleştirme: İki veya daha fazla görüntüyü piksel bazında birleştirmek için OR operatörü kullanılabilir. Bu işlemde, iki görüntünün aynı konumdaki pikselleri OR operatörü ile birleştirilir ve sonuç görüntüsünde her iki pikselin değeri de korunur.

Bölge genişletme: Bir görüntüdeki belirli bir bölgeyi genişletmek veya büyütmek için OR operatörü kullanılabilir. Bu işlemde, bir başlangıç noktası olarak seçilen piksel, etrafındaki benzer pikselleri belirli bir eşik değeriyle

karşılaştırarak genişletilecek bölgeyi belirler. OR operatörü, bu benzer pikselleri birleştirerek bölgenin genişletilmesini sağlar.

Görüntü maskeleri: OR operatörü, görüntü maskelerini birleştirmek veya uygulamak için kullanılabilir. Bir maske, belirli bir özellik veya bölgeyi temsil eden piksellerden oluşan bir görüntüdür. OR operatörü, farklı maskeleri birleştirerek birden fazla özelliği veya bölgeyi temsil eden pikselleri korur.

Bu örnekler, görüntü işleme uygulamalarında OR operatörünün yaygın olarak kullanıldığı bazı senaryoları göstermektedir. Ancak, OR operatörü, diğer görüntü işleme işlemlerinde de farklı şekillerde kullanılabilir.

OR Uygulamaları

```
private void btnDENEME_Click(object sender, EventArgs e)
{
    Bitmap Resim1 = new Bitmap(pictureBox1.Image);
    Bitmap Resim2 = new Bitmap(pictureBox2.Image);

    // Görüntülerin boyutlarını kontrol et
    if (Resim1.Width != Resim2.Width || Resim1.Height != Resim2.Height)
    {
        MessageBox.Show("Görüntüler aynı boyutta değil!");
        return;
    }

    // Sonuç görüntüsü için yeni bir bitmap oluştur
    Bitmap SonucGoruntusu = new Bitmap(Resim1.Width, Resim1.Height);

    // Görüntülerin piksellerini dolaşarak AND operatörü uygula
    for (int x = 0; x < Resim1.Width; x++)
    {
        for (int y = 0; y < Resim1.Height; y++)
        {
            // İki görüntünün piksellerini al
            Color renk1 = Resim1.GetPixel(x, y);
            Color renk2 = Resim2.GetPixel(x, y);


            // Piksellerin R, G ve B bileşenlerini OR operatörü ile birleştir
            Color sonucRenk = Color.FromArgb(
                renk1.R | renk2.R,
                renk1.G | renk2.G,
                renk1.B | renk2.B
            );

            // Sonuç görüntüsüne pikseli yerleştir
            SonucGoruntusu.SetPixel(x, y, sonucRenk);
        }
    }

    // Sonuç görüntüsünü göster
    pictureBox3.Image = SonucGoruntusu;

}
```

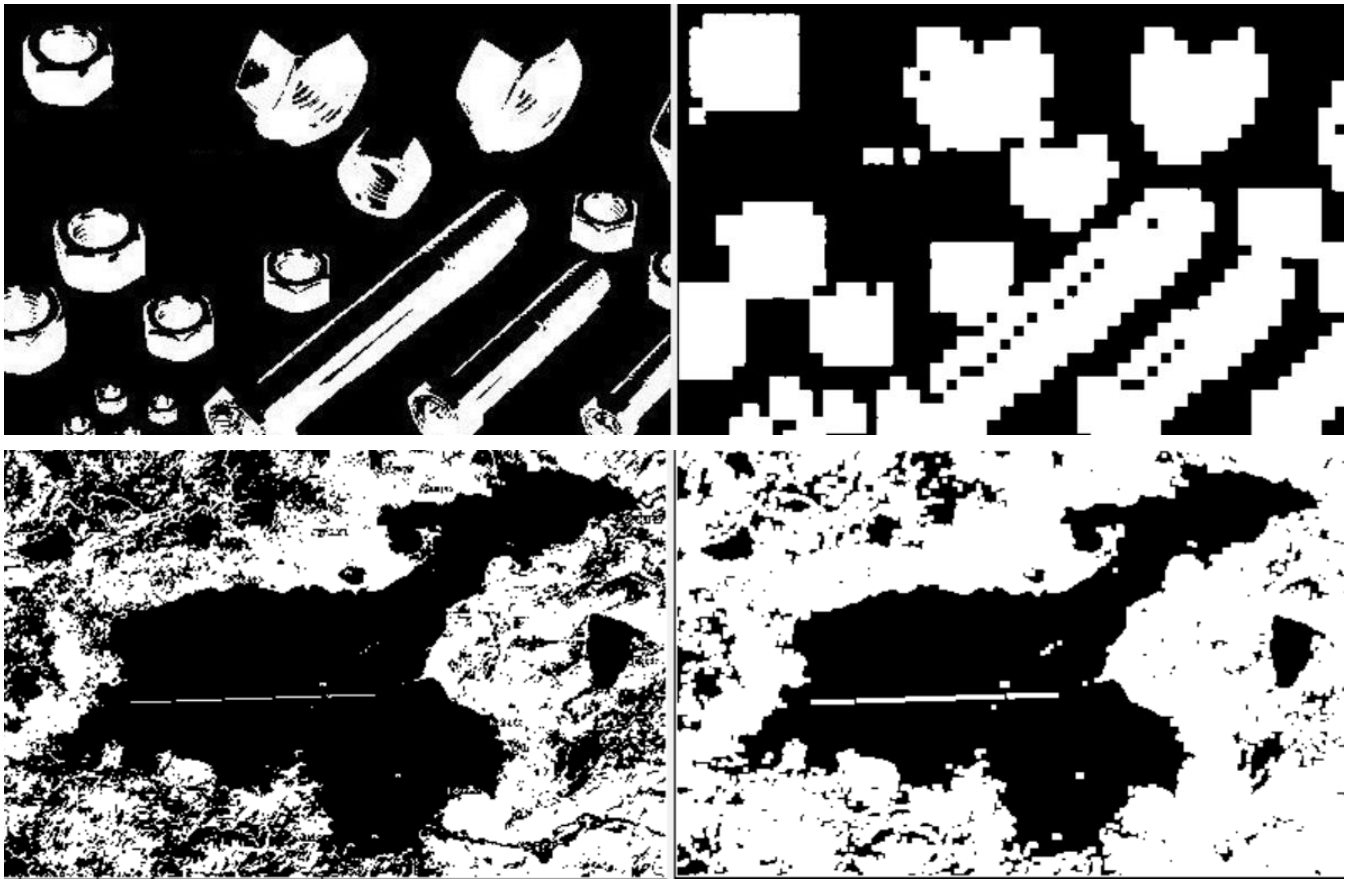
Maske olarak kullanımı

İki görüntüyü birleştirme

OR ile Bölge Genişletme

```
private void btnDENEME_Click(object sender, EventArgs e)
{
    // Görüntüyü yükle
    Bitmap image = new Bitmap(pictureBox1.Image);

    // Yeni bir bitmap oluştur
    Bitmap dilatedImage = new Bitmap(image.Width, image.Height);

    // Genişletme için kullanılacak yapılandırma elemanı
    int structuringElementSize = 3; // Yapılandırma elemanının boyutu (örneğin, 3x3)

    // Pikselleri dolaşarak bölge genişletme yap
    for (int x = structuringElementSize / 2; x < image.Width - structuringElementSize / 2; x++)
    {
        for (int y = structuringElementSize / 2; y < image.Height - structuringElementSize / 2;
y++)
        {
            // Pikselin etrafındaki yapılandırma elemanına bakarak bölge genişletme yap
            bool isDilated = false;

            for (int i = -structuringElementSize / 2; i <= structuringElementSize / 2; i++)
            {
                for (int j = -structuringElementSize / 2; j <= structuringElementSize / 2; j++)
                {
                    Color neighborPixel = image.GetPixel(x + i, y + j);

                    if (neighborPixel.GetBrightness() > 0)
                    {
                        isDilated = true;
                        break;
                    }
                }

                if (isDilated)
                    break;
            }

            // Genişletilmiş pikseli yerleştir
            if (isDilated)
                dilatedImage.SetPixel(x, y, Color.White);
            else
```

```
                    dilatedImage.SetPixel(x, y, Color.Black);
                }
            }

            // Genişletilmiş görüntüyü kaydet
            pictureBox2.Image = dilatedImage;

        }
```



Bu kod, "image.jpg" adlı bir görüntüyü yükler. Ardından, dilatedImage adlı yeni bir görüntü oluşturulur ve pikselleri dolaşarak her pikselin etrafındaki yapılandırma elemanına (structuring element) bakarak bölge genişletme işlemi yapar. Bölge genişletme, pikselin etrafındaki yapılandırma elemanında en az bir beyaz (parlaklık değeri 0'dan büyük) piksel varsa, pikseli beyaz yapar; aksi takdirde, pikseli siyah yapar. Bu işlem, görüntü üzerinde beyaz bölgeleri genişletir ve siyah bölgeleri daraltır. Sonuç olarak, "dilated_image.jpg" adıyla genişletilmiş görüntü kaydedilir.

Mantıksal XOR/XNOR Operatörleri

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

XOR

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

XNOR

XOR operatörü nerede kullanılır? (GPT)

Görüntü işleme alanında, XOR (exclusive OR) operatörü genellikle görüntüler arasında farklılık analizi, kenar tespiti veya renk efektleri gibi işlemlerde kullanılır. İşte bazı örnekler:

Farklılık analizi: İki görüntü arasındaki farkları analiz etmek için XOR operatörü kullanılabilir. XOR operatörü, her iki görüntünün aynı konumdaki piksellerini karşılaştırır ve farklı olan pikselleri belirgin hale getirir. Bu şekilde, iki görüntü arasındaki farklı bölgeleri vurgulamak veya çıkarmak için XOR operatörü kullanılabilir.

Kenar tespiti: XOR operatörü, kenar tespiti algoritmalarında kullanılabilir. Bir görüntünün kenarlarını tespit etmek için, görüntüdeki her pikseli çevresindeki piksellerle XOR operatörü ile karşılaştırabilirsiniz. Bu işlem, piksellerin benzerliklerini analiz ederek kenarlara karşılık gelen farklı pikselleri belirler.

Renk efektleri: XOR operatörü, renk efektleri oluşturmak için de kullanılabilir. Örneğin, bir görüntünün pikselleri ile sabit bir renk değeri arasında XOR operatörü uygulanabilir. Bu, görüntüye ilginç ve dikkat çekici renkli efektler eklemek için kullanılabilir.

Bu örnekler, görüntü işleme uygulamalarında XOR operatörünün yaygın olarak kullanıldığı bazı senaryoları göstermektedir. Ancak, XOR operatörü, diğer görüntü işleme işlemlerinde de farklı şekillerde kullanılabilir.

XoR ile Kenar Tespiti Yapan Algoritma

```
private void DENEME2_Click(object sender, EventArgs e)
{
    // Görüntüyü yükle
    Bitmap image = new Bitmap(pictureBox1.Image);

    // Yeni bir bitmap oluştur
    Bitmap edgeImage = new Bitmap(image.Width, image.Height);

    // Pikselleri dolaşarak kenar tespiti yap
    for (int x = 1; x < image.Width - 1; x++)
    {
        for (int y = 1; y < image.Height - 1; y++)
        {
            // Pikselin etrafındaki komşu pikselleri al
            Color pixel = image.GetPixel(x, y);
            Color leftPixel = image.GetPixel(x - 1, y);
            Color rightPixel = image.GetPixel(x + 1, y);
            Color topPixel = image.GetPixel(x, y - 1);
            Color bottomPixel = image.GetPixel(x, y + 1);

            // Piksellerin R, G ve B değerlerini XOR operatörü ile karşılaştır
            bool isEdgePixel = (pixel.R ^ leftPixel.R) > 128 ||
                               (pixel.R ^ rightPixel.R) > 128 ||
                               (pixel.R ^ topPixel.R) > 128 ||
                               (pixel.R ^ bottomPixel.R) > 128;

            // Eğer kenar pikseli ise beyaz yap, değilse siyah yap
```

```
            if (isEdgePixel)
                edgeImage.SetPixel(x, y, Color.White);
            else
                edgeImage.SetPixel(x, y, Color.Black);
        }
    }

    // Kenar tespiti yapılan görüntüyü kaydet
    pictureBox2.Image = edgeImage;
}
```

Bu kod, "image.jpg" adlı bir görüntüyü yükler. Ardından, edgeImage adlı yeni bir görüntü oluşturulur ve pikselleri dolaşarak her pikselin etrafındaki komşu pikselleri alır. Daha sonra, piksellerin R (kırmızı) bileşenlerini XOR operatörü ile karşılaştırarak kenar piksellerini belirler. Eğer XOR işlemi sonucu elde edilen değer 128'den büyük ise piksel kenar pikseli olarak kabul edilir. Kenar pikselleri beyaz yapılır, diğer pikseller siyah yapılır. Sonuç olarak, "edge_image.jpg" adıyla kenar tespiti yapılan görüntü kaydedilir.

Bu kod örneği, basit bir kenar tespiti algoritmasıdır ve çıktıda yalnızca yatay ve dikey kenarlar belirginleştirilir. Daha gelişmiş kenar tespiti algoritmaları, farklı filtreler veya yöntemler kullanarak daha hassas sonuçlar elde edebilir.

XoR ile İki Resimde Farklı Bölgeleri Tespit Etme

```
private void btnDENEME_Click(object sender, EventArgs e)
{
    Bitmap Resim1 = new Bitmap(pictureBox1.Image);
    Bitmap Resim2 = new Bitmap(pictureBox2.Image);

    // Görüntülerin boyutlarını kontrol et
    if (Resim1.Width != Resim2.Width || Resim1.Height != Resim2.Height)
    {
        MessageBox.Show("Görüntüler aynı boyutta değil!");
        return;
    }

    // Sonuç görüntüsü için yeni bir bitmap oluştur
    Bitmap SonucGoruntusu = new Bitmap(Resim1.Width, Resim1.Height);

    // Görüntülerin piksellerini dolaşarak AND operatörü uygula
    for (int x = 0; x < Resim1.Width; x++)
    {
        for (int y = 0; y < Resim1.Height; y++)
        {
            // İki görüntünün piksellerini al
            Color pixel1 = Resim1.GetPixel(x, y);
            Color pixel2 = Resim2.GetPixel(x, y);
```

```
            // Piksellerin R, G ve B değerlerini XOR operatörü ile karşılaştır
            bool isDifferent = (pixel1.R ^ pixel2.R) > 128 ||
                               (pixel1.G ^ pixel2.G) > 128 ||
                               (pixel1.B ^ pixel2.B) > 128;

            // Eğer pikseller farklı ise beyaz yap, değilse siyah yap
            if (isDifferent)
                SonucGoruntusu.SetPixel(x, y, Color.White);
            else
                SonucGoruntusu.SetPixel(x, y, Color.Black);
        }
    }

    // Sonuç görüntüsünü göster
    pictureBox3.Image = SonucGoruntusu;

}
```

Bu kod, "image1.jpg" ve "image2.jpg" adlı iki görüntüyü yükler. Ardından, görüntülerin boyutlarını kontrol eder ve eşit olduklarını doğrular. Sonuç olarak, diffImage adlı yeni bir görüntü oluşturulur ve pikselleri dolaşarak her iki görüntünün piksellerini XOR operatörü ile karşılaştırır. XOR operatörü, her iki pikselin R, G ve B bileşenlerini XOR işlemine tabi tutar. Eğer XOR işlemi sonucu elde edilen değer 128'den büyük ise pikseller farklı olduğu kabul edilir. Farklı pikseller beyaz yapılır, diğer pikseller siyah yapılır. Sonuç olarak, "diff_image.jpg" adıyla farklılık analizi yapılan görüntü kaydedilir.

XoR ile Görüntüye Efekt Uygulama

```
private void DENEME2_Click(object sender, EventArgs e)
{
    // Görüntüyü yükle
    Bitmap image = new Bitmap(pictureBox1.Image);

    // Renk efekti için XOR ile kullanılacak renk değeri
    Color effectColor = Color.FromArgb(255, 128, 0); // Örnek olarak turuncu

    // Yeni bir bitmap oluştur
    Bitmap effectImage = new Bitmap(image.Width, image.Height);

    // Pikselleri dolaşarak renk efekti uygula
    for (int x = 0; x < image.Width; x++)
    {
        for (int y = 0; y < image.Height; y++)
        {
            // Pikselin renk değerini al
            Color pixel = image.GetPixel(x, y);

            // Pikselin R, G ve B değerlerini XOR operatörü ile belirtilen renkle birleştir
            Color resultPixel = Color.FromArgb(
                pixel.R ^ effectColor.R,
                pixel.G ^ effectColor.G,
                pixel.B ^ effectColor.B
            );
```

```
            // Efekt uygulanmış pikseli yerleştir
            effectImage.SetPixel(x, y, resultPixel);
        }
    }

    // Efekt uygulanmış görüntüyü göster
    pictureBox2.Image = effectImage;
}
```

Bu kod, "image.jpg" adlı bir görüntüyü yükler. Ardından, bir renk efekti oluşturmak için effectColor adlı bir renk değeri belirler. Sonuç olarak, effectImage adlı yeni bir görüntü oluşturulur ve pikselleri dolaşarak her pikselin R, G ve B değerlerini XOR operatörü ile effectColor değeriyle birleştirir. Bu işlem, pikselin renk değerleri ile belirtilen efekt rengi arasında XOR işlemi yaparak yeni bir renk oluşturur. Sonuç olarak, "effect_image.jpg" adıyla efekt uygulanmış görüntü kaydedilir.

Mantıksal NOT Operatörü (Invert-Tersini alma)

A	Q
0	1
1	0

NOT

Mantıksal operatörlerden NOT operatörü, görüntü işlemede bazı kullanım örnekleri:

Negatif Görüntü Oluşturma: Görüntünün her pikselinin renk değerlerini tersine çevirmek için NOT operatörü kullanılabilir. Bu işlem, görüntünün negatifini oluşturarak parlaklık ve kontrastı değiştirir.

Görüntü Binarizasyonu: Görüntüdeki pikselleri siyah ve beyaz olarak ayırmak için eşik değeri kullanılır. NOT operatörü, eşik değerine göre piksellerin rengini tersine çevirerek binarizasyon işlemini gerçekleştirebilir.

Kenar Tespiti: Kenar tespiti algoritmalarında, görüntü üzerindeki yoğunluk değişimlerini tespit etmek için NOT operatörü kullanılabilir. Böylece, kenarlar daha belirgin hale gelir ve görüntünün sadece kenarları içeren bir versiyonu elde edilir.

Gürültü Azaltma: Görüntüdeki gürültüyü azaltmak için bazı filtreler kullanılır. NOT operatörü, görüntünün negatifini alarak gürültüyü azaltabilir veya belirli bir deseni veya yapıyı ortaya çıkarabilir.

Bu sadece birkaç örnektir ve NOT operatörü, görüntü işlemede farklı amaçlar için kullanılabilir. Uygulama senaryosuna ve ihtiyaçlara bağlı olarak farklı kullanımlar da mümkündür.

NOT operatörü ile Kenar Tespiti

```
private void btnDENEME_Click(object sender, EventArgs e)
{

    // Görüntüyü yükle
    Bitmap image = new Bitmap(pictureBox1.Image);

    // Yeni bir bitmap oluştur
    Bitmap edgeImage = new Bitmap(image.Width, image.Height);

    // Kenar tespiti için kullanılacak eşik değeri
    int threshold = 30;

    // Pikselleri dolaşarak kenar tespiti yap
    for (int x = 1; x < image.Width - 1; x++)
    {
        for (int y = 1; y < image.Height - 1; y++)
        {
            // Pikselin renk değerlerini al
            Color pixel = image.GetPixel(x, y);
            Color pixelRight = image.GetPixel(x + 1, y);
            Color pixelDown = image.GetPixel(x, y + 1);

            // Pikselin parlaklık değerlerini hesapla
            int brightness = (pixel.R + pixel.G + pixel.B) / 3;
            int brightnessRight = (pixelRight.R + pixelRight.G + pixelRight.B) / 3;
            int brightnessDown = (pixelDown.R + pixelDown.G + pixelDown.B) / 3;

            // Parlaklık farkını hesapla
            int differenceX = Math.Abs(brightness - brightnessRight);
            int differenceY = Math.Abs(brightness - brightnessDown);

            // Eşik değeri ile karşılaştır ve kenar pikseli belirle
            if (differenceX > threshold || differenceY > threshold)
            {
                edgeImage.SetPixel(x, y, Color.Black);
            }
            else
            {
                edgeImage.SetPixel(x, y, Color.White);
            }
        }
    }

    // Kenar tespit edilmiş görüntü

    pictureBox2.Image = edgeImage;

}
```


Bu kod, "image.jpg" adlı bir görüntüyü yükler. Ardından, edgeImage adlı yeni bir görüntü oluşturulur ve pikselleri dolaşarak her pikselin etrafındaki piksellerle parlaklık farkını hesaplar. Parlaklık farkı, pikselin merkezindeki pikselin parlaklık değeri ile sağdaki veya altındaki pikselin parlaklık değeri arasındaki farktır. Bu fark, belirlenen eşik değeri (threshold) ile karşılaştırılır ve eşik değerinden büyükse piksel kenar pikseli olarak kabul edilir ve siyah (Color.Black), değilse beyaz (Color.White) olarak atanır. Bu işlem, kenar tespiti yaparak görüntüdeki kenarları belirginleştirir. Sonuç olarak, "edge_image.jpg" adıyla kenar tespit edilmiş görüntü kaydedilir. Bu örnekte, eşik değeri olarak 30 kullanıldı. Siz istediğiniz farklı bir eşik değeriyle (threshold) deneme yaparak kenar tespit edebilirsiniz

Bitshift (Bit kaydırma) Operatörü

Mantıksal operatörlerden biri olan bit kaydırma (bit shifting) operatörleri, görüntü işleme alanında çeşitli amaçlarla kullanılabilir. İşte bit kaydırma operatörlerinin bazı kullanım alanları:

Piksel İşleme: Görüntü işleme uygulamalarında, piksel değerlerini manipüle etmek için bit kaydırma operatörleri kullanılabilir. Örneğin, bir görüntünün parlaklık değerlerini artırmak veya azaltmak için piksel değerlerini sola veya sağa kaydırarak işlemler yapılabilir.

Maskeleme (Masking): Maskeleme işlemlerinde, belirli bir desenin veya maskeyi görüntü üzerinde uygulamak için bit kaydırma operatörleri kullanılabilir. Maskeleme, belirli bir desenin belirli bölgeleri seçmek veya gizlemek için kullanılır.

Kenar Tespiti: Kenar tespiti işlemlerinde, görüntünün kenarlarını belirlemek için bit kaydırma operatörleri kullanılabilir. Özellikle Canny kenar tespiti gibi algoritmalarda, kenarları belirlemek için bit kaydırma operatörleri kullanılan filtreler bulunur.

Örtüşme (Overlap): Bazı görüntü işleme işlemlerinde, iki veya daha fazla görüntüyü birleştirirken bit kaydırma operatörleri kullanılabilir. Örneğin, iki görüntünün piksellerini birleştirmek veya üst üste bindirmek için bit kaydırma operatörleri kullanılabilir.

İşaretçi Operasyonları: Görüntü işleme uygulamalarında, bellek manipülasyonu için işaretçi operasyonları kullanılır. Bit kaydırma operatörleri, bellek adreslerini hesaplarken veya veri bloklarını işaretçi aracılığıyla taşırken kullanılabilir.

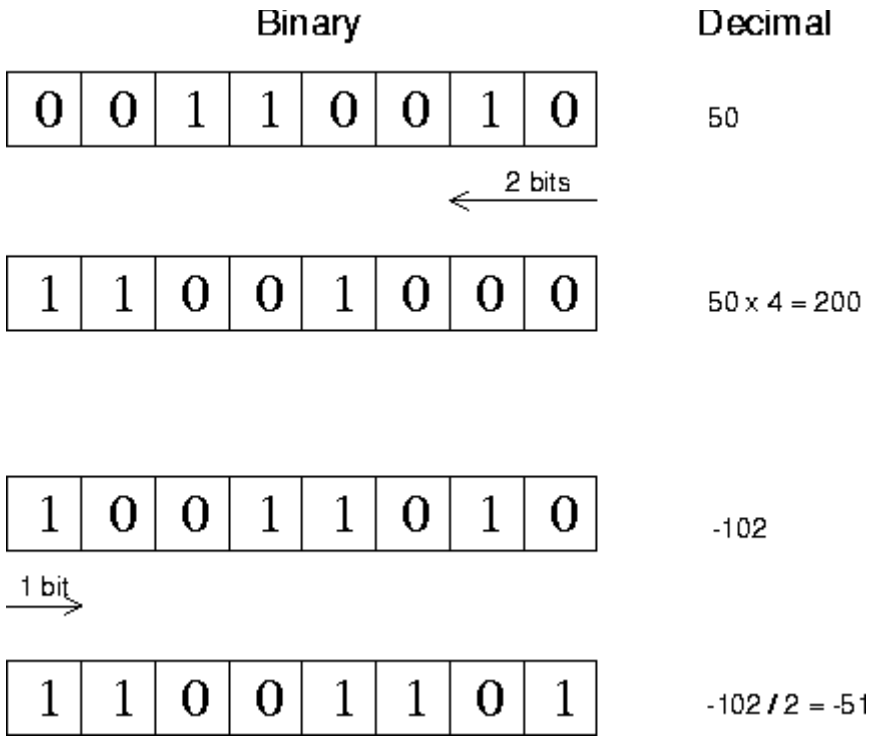
Bu sadece bazı örneklerdir ve bit kaydırma operatörleri daha geniş bir şekilde görüntü işleme alanında kullanılabilir. Uygulamanıza ve kullanım senaryonuza bağlı olarak, bit kaydırma operatörlerini farklı şekillerde kullanabilirsiniz.

$$\begin{aligned} \text{Shifting } i \text{ bits to the right} &\Leftrightarrow Q(i,j) = P(i,j) \div 2^i \\ \text{Shifting } i \text{ bits to the left} &\Leftrightarrow Q(i,j) = P(i,j) \times 2^i \end{aligned}$$

İşlem, tek bir geçişte doğrudan gerçekleştirilir. Bir sayının ikili gösterimi bir yönde kaydırılırsa, karşı tarafta boş bir konum elde ederiz. Bu boş pozisyonun nasıl doldurulacağına dair genellikle üç olasılık vardır: boş bitleri 0 veya 1

ile doldurabiliriz veya diğer taraftaki sayının ikili gösteriminden kaydırılan bitleri sarabiliriz. Son olasılık, ikili sayıyı döndürmeye eşdeğerdir.

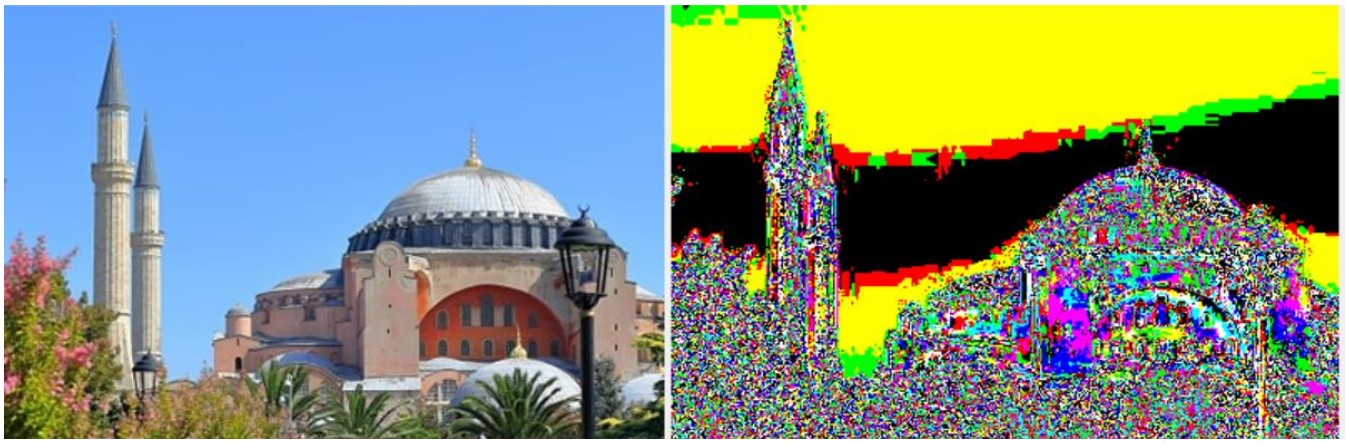
Kullanılan tekniğin seçimi operatörün uygulamasına ve uygulamaya bağlıdır. Çoğu durumda, hızlı bir çarpma veya bölme gerçekleştirmek için bit kaydırma kullanılır. Bu uygulama için doğru sonuçları elde etmek için boş bitleri 0 ile doldurmamız gerekir. Sadece negatif bir sayının 2'ye bölünmesi durumunda sol bitleri 1 ile doldurmamız gerekir, çünkü negatif bir sayı, pozitif sayının ikiye tümleyeni olarak temsil edilir, yani işaret biti 1'dir. Bu şekilde bit kaydırma uygulamanın sonucu aşağıdaki formülde gösterilmektedir:



Çarpma ve bölme için bit kaydırma kullanımına örnekler. Alttaki örneğin, bir baytın -128 ile +127 arasında bir sayıyı temsil ettiği bir imzalı bayt kuralı kullandığını unutmayın.

```
private void btnDENEME_Click(object sender, EventArgs e)
{
    // Görüntüyü sola kaydır
    int shiftAmount = -5;

    // Görüntüyü yükle
    Bitmap image = new Bitmap(pictureBox1.Image);

    // Yeni bir bitmap oluştur
    Bitmap shiftedImage = new Bitmap(image.Width, image.Height);

    // Pikselleri dolaşarak kaydırma işlemini yap
    for (int x = 0; x < image.Width; x++)
    {
        for (int y = 0; y < image.Height; y++)
        {
            // Pikselin renk değerini al
            Color pixel = image.GetPixel(x, y);

            // Renk değerlerini sola kaydır
            int red = pixel.R << shiftAmount;
            int green = pixel.G << shiftAmount;
            int blue = pixel.B << shiftAmount;

            // Renk değerlerini sınırla (0-255 aralığında)
            red = Math.Max(0, Math.Min(red, 255));
            green = Math.Max(0, Math.Min(green, 255));
```

```
            blue = Math.Max(0, Math.Min(blue, 255));

            // Kaydırılmış pikseli ata
            Color shiftedPixel = Color.FromArgb(red, green, blue);
            shiftedImage.SetPixel(x, y, shiftedPixel);
        }
    }

    pictureBox2.Image = shiftedImage;

}
```



Bit kaydırma ile Örnek Overlap (Örtüşme) yapan kod.

```
        private void DENEME2_Click(object sender, EventArgs e)
        {
            // İlk görüntüyü yükle
            Bitmap image1 = new Bitmap(pictureBox1.Image);

            // İkinci görüntüyü yükle
            Bitmap image2 = new Bitmap(pictureBox2.Image);

            // Örtüşme işlemi için kullanılacak kaydırma miktarı
            int shiftAmount = 50;

            // İlk görüntüyü sola kaydır
            Bitmap shiftedImage1 = ShiftImage(image1, -shiftAmount);

            // İkinci görüntüyü sağa kaydır
            Bitmap shiftedImage2 = ShiftImage(image2, shiftAmount);

            // İki görüntüyü birleştir
            Bitmap overlappedImage = OverlapImages(shiftedImage1, shiftedImage2);

            // Örtüşme sonucunu kaydet
            pictureBox3.Image = overlappedImage;
        }
        static Bitmap ShiftImage(Bitmap image, int shiftAmount)
        {
            // Yeni bir bitmap oluştur
            Bitmap shiftedImage = new Bitmap(image.Width, image.Height);

            // Pikselleri dolaşarak kaydırma işlemini yap
            for (int x = 0; x < image.Width; x++)
            {
                for (int y = 0; y < image.Height; y++)
                {
                    // Pikselin renk değerini al
                    Color pixel = image.GetPixel(x, y);
```

```
                // Renk değerlerini kaydır
                int red = pixel.R << shiftAmount;
                int green = pixel.G << shiftAmount;
                int blue = pixel.B << shiftAmount;

                // Renk değerlerini sınırla (0-255 aralığında)
                red = Math.Max(0, Math.Min(red, 255));
                green = Math.Max(0, Math.Min(green, 255));
                blue = Math.Max(0, Math.Min(blue, 255));

                // Kaydırılmış pikseli ata
                Color shiftedPixel = Color.FromArgb(red, green, blue);
                shiftedImage.SetPixel(x, y, shiftedPixel);
            }
        }

        return shiftedImage;
    }

    static Bitmap OverlapImages(Bitmap image1, Bitmap image2)
    {
        // Yeni bir bitmap oluştur
        Bitmap overlappedImage = new Bitmap(image1.Width, image1.Height);

        // Pikselleri dolaşarak örtüşme işlemini yap
        for (int x = 0; x < image1.Width; x++)
        {
            for (int y = 0; y < image1.Height; y++)
            {
                // İlk görüntünün pikselini al
                Color pixel1 = image1.GetPixel(x, y);

                // İkinci görüntünün pikselini al
                Color pixel2 = image2.GetPixel(x, y);

                // Pikselleri topla
                int red = pixel1.R + pixel2.R;
                int green = pixel1.G + pixel2.G;
                int blue = pixel1.B + pixel2.B;

                // Renk değerlerini sınırla (0-255 aralığında)
                red = Math.Max(0, Math.Min(red, 255));
                green = Math.Max(0, Math.Min(green, 255));
                blue = Math.Max(0, Math.Min(blue, 255));

                // Birleştirilmiş pikseli ata
                Color overlappedPixel = Color.FromArgb(red, green, blue);
                overlappedImage.SetPixel(x, y, overlappedPixel);
            }
        }

        return overlappedImage;
    }
```

********** 000 **********

Ödev1: Mantıksal Operatörlerin günlük yaşamda kullanıldığı bir alanı bulun, programlayın.

Ödev 2: 0 bitlerle oluşan resimden 5 bitlerle oluşan resmi çıkarma ve ekleme yaparak yeni görüntüler elde edin.

Ödev 3: Normalde 0 biti yansıtıp diğerlerini yansıtmadık. Bunun tarsi olabilir. 0 biti yansıtma diğerlerini yansıt.

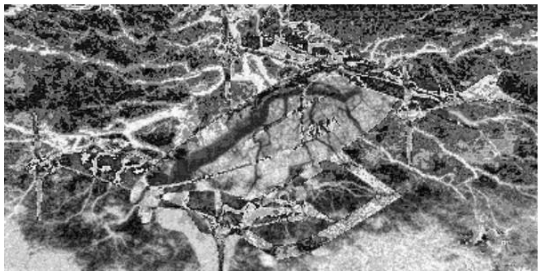
Ödev 4: Bitshıft (Kaydırma) İşlemini Ödev Olarak Deneyin…

Ödev 5: Resmi anahtar resim ve anahtar sayı ile şifreleyerek gönderme: Bir resmin internet üzerinden gizlenip gönderilmesi için örnek bir algoritma geliştirin. Resmi başka bir anahtar resimle bit işlemlerine tabi tutun. Resmi alan kişinin elinde anahtar resim bulunsun. O anahtar resmi kullanarak gizlenen resmi tekrar orijinal haline dönüştürebilsin.

Gizlenecek Resim

Anahtar resim

Gizlenmiş resim (başkaları resmi anlayamayacak şekilde karışmış olacak)



Gizlenmiş resim anahtar resimle çözülünce tekrar bu resim elde edilecek.