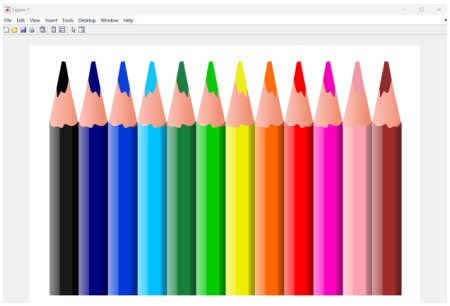
1) Renkli bir görüntüyü gri tonlu hale dönüştürmek

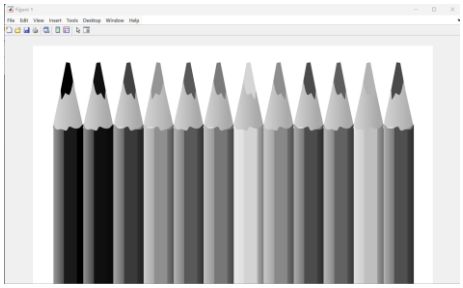
Renkli bir görüntüyü gri tonlu hale dönüştürmek, görüntünün her bir pikselindeki renk bilgilerini tek bir gri ton değerine indirgemek anlamına gelir. Gri tonlama işlemi, genellikle insan gözünün farklı renk tonlarına duyarlılığını dikkate alarak yapılır.

Matlab kodu:

```
 % Görüntünün okunması

I1 = imread('kalem.png');

% Görüntünün gri tona dönüştürülmesi

I2 = rgb2gray(I1);

% Yeni görüntünün gösterilmesi

imshow(I2);

I1 = imread('kalem.png');
```



Şekil 1- orjinal görüntü



Şekil 1-Gri tonlu Görüntü

Algoritma ile yazılımı Python:

```
from PIL import Image
def griye_cevir_ve_goster(kalem, kaydetme_yolu):
    original_gorsel = Image.open(kalem)
    # Görüntünün RGB formatında gri tonlamaya dönüştürülmesi
    griye_dondurme = original_gorsel.convert("L")
    # Gri tonlamalı görüntünün belirtilen yola kaydedilmesi
    griye_dondurme.save(kaydetme_yolu)
    # Gri tonlamalı görüntünün gösterilmesi
    griye_dondurme.show()
goruntu_yolu = r'C:\Users\tugba\OneDrive\Masaüstü\grtpython\kalem.png'
kayit_yolu = r'C:\Users\tugba\OneDrive\Masaüstü\grtpython\kalem_gri.png'
griye_cevir_ve_goster(goruntu_yolu, kayit_yolu)
```

Python kodu:

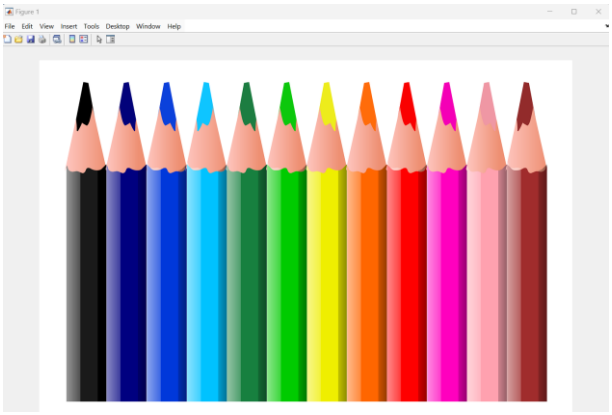
```
import cv2
from matplotlib import pyplot as plt
I1 = cv2.imread('/content/kalem.png
I2 = cv2.cvtColor(I1, cv2.COLOR_BGR2GRAY)
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(I1, cv2.COLOR_BGR2RGB))
plt.title('Orijinal Görüntü')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(I2, cmap='gray')
plt.title('Gri Tonlama Görüntü')
plt.axis('off')
plt.show()
```

2) Görüntünün Histogramının Elde Edilmesi

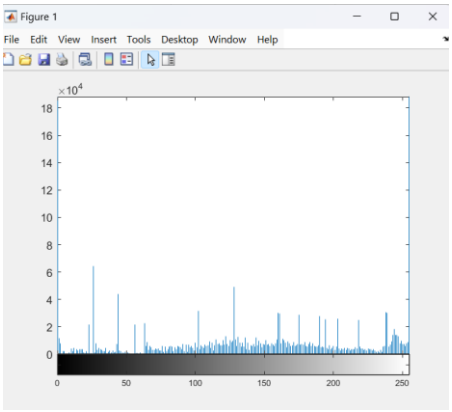
Görüntünün histogramı, bir görüntüdeki piksel değerlerinin frekans dağılımını gösteren bir grafiktir. Histogram, görüntüdeki parlaklık seviyelerini veya renk yoğunluklarını analiz etmek için kullanılır.

Matlab:

```
I1 = imread('kalem.png');

% Görüntünün histogramının çizilmesi

imhist(I1)

I1 = imread('kalem.png');
```



Şekil 2- Orijinal görüntü

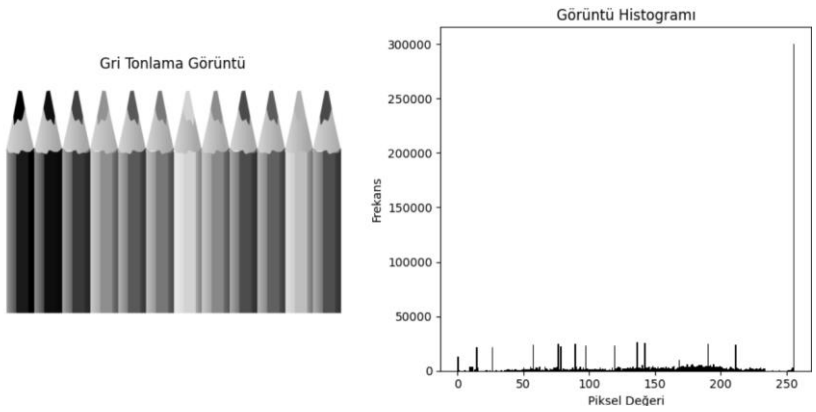


Şekil 2- Görüntünün histogramı

Python kodu:

```
 import cv2
import matplotlib.pyplot as plt
import numpy as np

def calculate_histogram_manual(image, bins=256):
   # Histogram hesaplama
   hist = np.zeros(bins, dtype=int)
   for i in range(image.shape[0]):
      for j in range(image.shape[1]):
         pixel_value = image[i, j]
         hist[pixel_value] += 1
   return hist

def display_histogram(image, hist):
   plt.figure(figsize=(8, 6))

   # Resmin histogramını çizme
   plt.subplot(1, 2, 1)
   plt.imshow(image, cmap='gray')
   plt.title('Gri Tonlama Resmi')

   plt.subplot(1, 2, 2)
   plt.plot(hist, color='black')
   plt.title('Resmin Histogramı')

   plt.tight_layout()
   plt.show()



image_path = 'kalem.png'
```

```
original_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)


# Histogramı hesapla

grayscale_hist_manual = calculate_histogram_manual(original_image)


# Histogramı göster

display_histogram(original_image, grayscale_hist_manual)
```



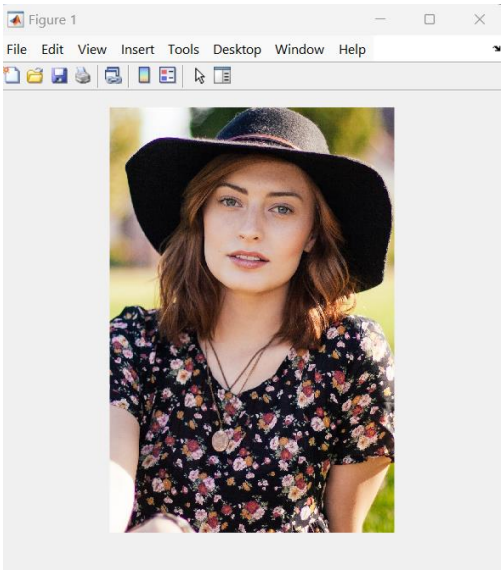
3) Eşikleme

Eşikleme, dijital görüntü işleme tekniklerinden biridir ve bir görüntüyü daha basit bir forma dönüştürerek analiz etmeyi kolaylaştırır. Bu işlem, görüntüdeki piksel değerlerini bir eşik değeriyle karşılaştırarak her bir pikselin değerini **0** veya **1** (ikili görüntü) ya da başka bir sabit değere dönüştürür.

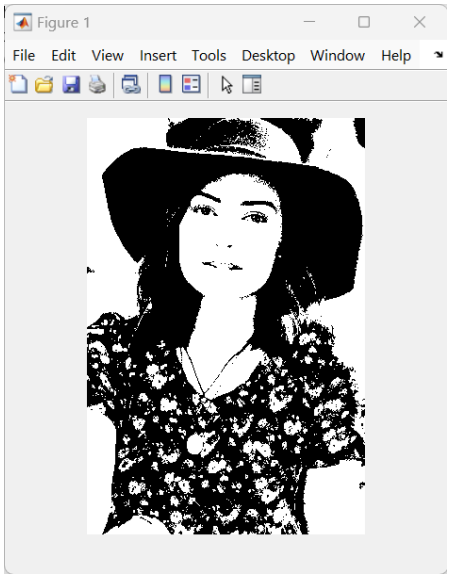
Matlab Kodu:

```
% Görüntünün okunması

I1 = imread('woman.jpg');

% Thresholding seviyesinin belirlenmesi

level = graythresh(I1);
```

```
% Thresholdingin uygulanması

bw = im2bw(I1,level);

% Yeni görüntünün gösterilmesi

figure, imshow(bw)
```



Şekil 3-Orijinal Görüntü



Şekil 3- Eşiklenmiş görüntü

Python Kodu:

```
import cv2

import matplotlib.pyplot as plt

import numpy as np


def rgb_to_grayscale_manual(image):

    # Görüntüyü gri tonlamaya dönüştürme (manuel yöntem)

    height, width, channels = image.shape
```

```
    gray_image = np.zeros((height, width), dtype=np.uint8)

    for i in range(height):
        for j in range(width):
            r, g, b = image[i, j]  # R, G, B bileşenlerini al
            # Gri tonlama formülü uygula
            gray_value = int(0.2989 * r + 0.5870 * g + 0.1140 * b)
            gray_image[i, j] = gray_value

    return gray_image

def manual_threshold(image, threshold_value):
    # Eşikleme işlemi (manuel yöntem)
    height, width = image.shape
    thresholded_image = np.zeros_like(image, dtype=np.uint8)

    for i in range(height):
        for j in range(width):
            if image[i, j] >= threshold_value:
                thresholded_image[i, j] = 255  # Beyaz
            else:
                thresholded_image[i, j] = 0  # Siyah

    return thresholded_image
```

```
def display_images(original, grayscale, thresholded):
    plt.figure(figsize=(12, 6))

    # Orijinal resim
    plt.subplot(1, 3, 1)
    plt.imshow(cv2.cvtColor(original, cv2.COLOR_BGR2RGB))
    plt.title('Orjinal Resim')

    # Gri tonlama resmi
    plt.subplot(1, 3, 2)
    plt.imshow(grayscale, cmap='gray')
    plt.title('Gri Tonlama Resmi')

    # Eşiklenmiş resim
    plt.subplot(1, 3, 3)
    plt.imshow(thresholded, cmap='gray')
    plt.title('Eşiklenmiş Resim')

    plt.tight_layout()
    plt.show()

# Kullanım
image_path = 'woman.jpg'
```

original_image = cv2.imread(image_path) # OpenCV kullanılarak görüntüyü oku

grayscale_image_manual = rgb_to_grayscale_manual(original_image) # Manuel gri tonlama

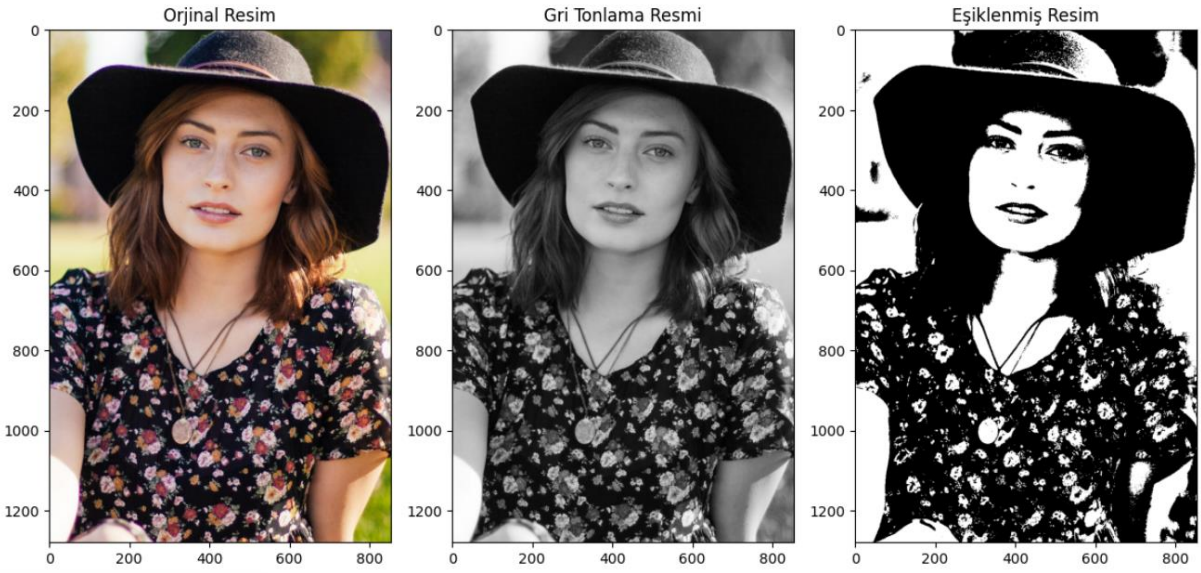
Eşikleme (manuel)

threshold_value = 128 # Eşik değeri (örneğin 128)

thresholded_image = manual_threshold(grayscale_image_manual, threshold_value)

Resimleri görüntüle

display_images(original_image, grayscale_image_manual, thresholded_image)



4) **Negatif görüntüleme**

Negatif görüntüleme, dijital görüntü işleme tekniklerinden biridir ve bir görüntünün piksel değerlerini tersine çevirerek negatifini oluşturur. Bu işlem, özellikle tıbbi görüntüleme ve fotoğrafçılık gibi alanlarda görüntülerin daha detaylı analiz edilmesini sağlamak için kullanılır.

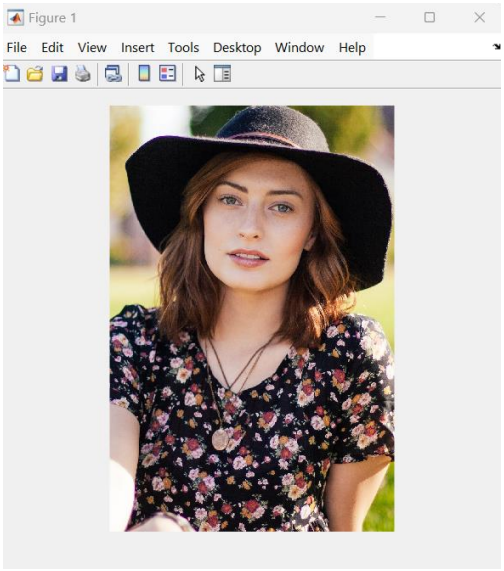
Matlab Kodu:

% Görüntünün okunması

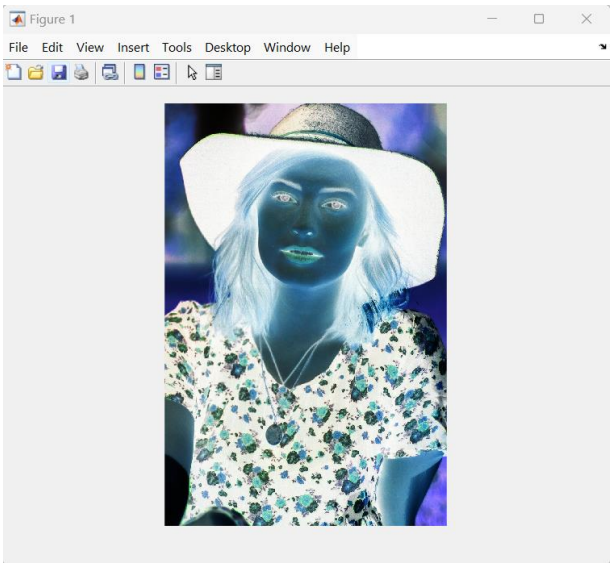
I1 = imread('woman.jpg');

% Görüntünün negatifinin oluşturulması

```
I2 = imcomplement(I1);

% Yeni görüntünün gösterilmesi

imshow(I2);
```



Şekil 4-Orijinal Görüntü

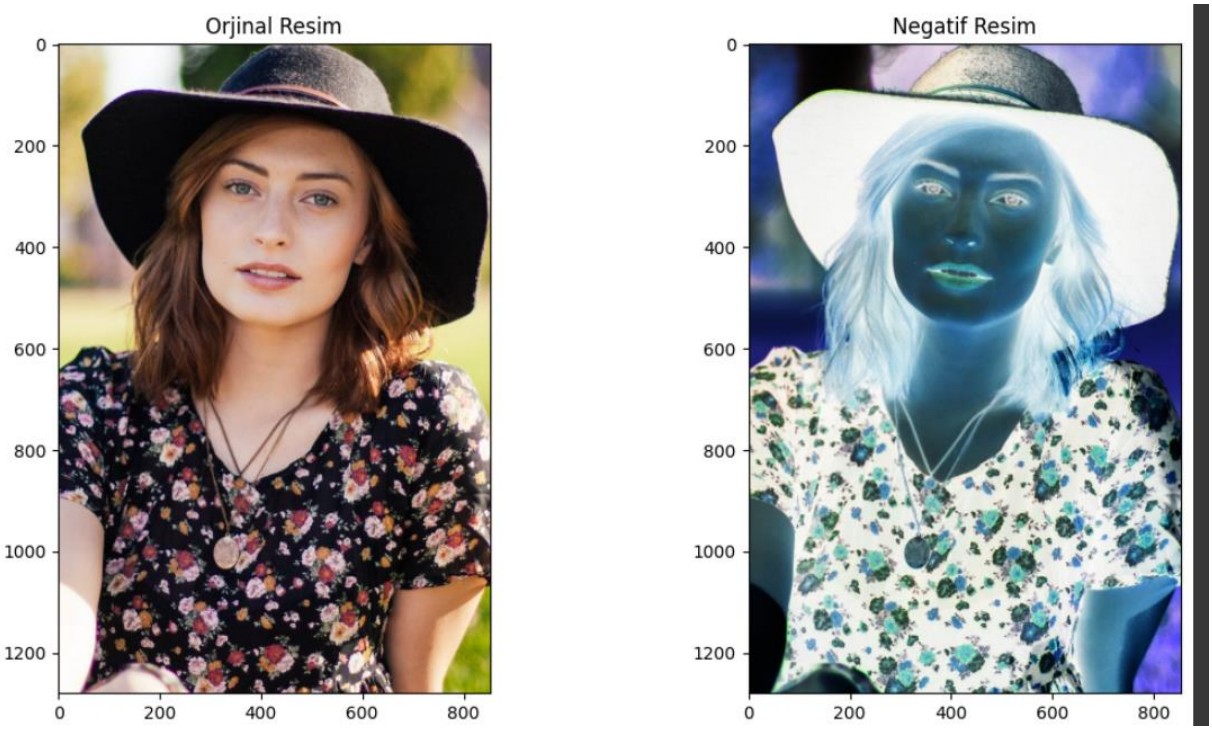


Şekil 5-Orijinal Görüntünün Negatifi Alınmış Hali

Python Kodu:

```
 import cv2

import matplotlib.pyplot as plt

import numpy as np


def negative_image(image):

   # Renkli resmin negatifini almak için her kanalın piksel değerini 255'ten çıkarıyoruz

   return 255 - image


def display_images(original, negative):

   plt.figure(figsize=(12, 6))


   # Orijinal resim

   plt.subplot(1, 2, 1)

   plt.imshow(cv2.cvtColor(original, cv2.COLOR_BGR2RGB))  # OpenCV, BGR formatında okur, RGB'ye dönüştürüyoruz

   plt.title('Orjinal Resim')
```

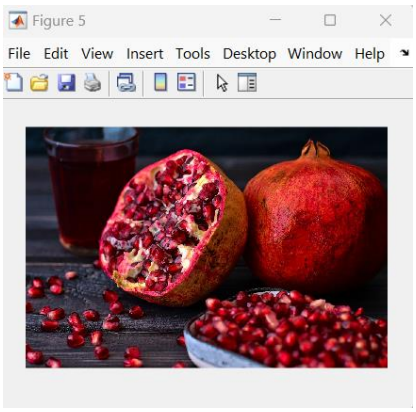
```
    # Negatif resim
    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(negative, cv2.COLOR_BGR2RGB))  # OpenCV, BGR formatında okur, RGB'ye dönüştürüyoruz
    plt.title('Negatif Resim')

    plt.tight_layout()
    plt.show()

image_path = 'woman.jpg'  # Resim dosyasının yolu
original_image = cv2.imread(image_path)  # OpenCV kullanılarak görüntüyü oku

# Orijinal resmin negatifini al
negative_image_result = negative_image(original_image)

# Resimleri görüntüle
display_images(original_image, negative_image_result)
```



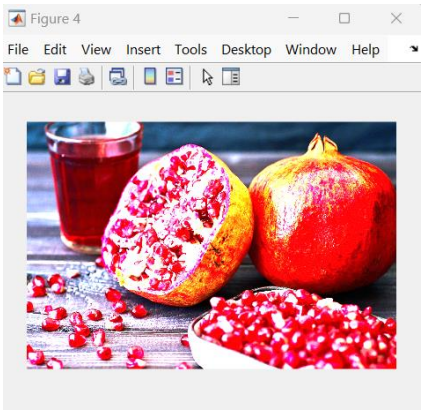
5) Alçak Geçiren Filtreleme

Alçak geçiren filtreleme (low-pass filtering), dijital görüntü işlemekte kullanılan bir yöntemdir ve temel amacı, görüntüdeki **yüksek frekanslı bileşenleri** (ani değişimler, kenarlar ve gürültü) azaltarak görüntüyü yumuşatmaktır. Böylece, görüntünün **düşük frekanslı bileşenleri** (genel yapı ve yumuşak geçişler) korunur.

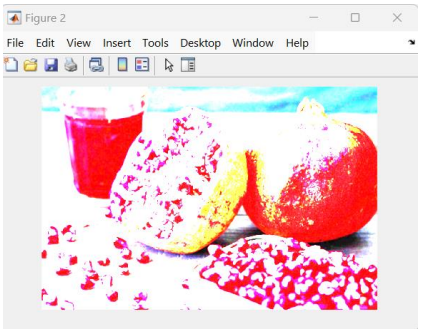
```
% Görüntünün okunup double tipine veri dönüşümünün yapılması
I1 = im2double(imread('nar.jpg'));
% 2x2 ve 5x5 filtrelerin oluşturulması
w1=ones(2); w2=ones(5);
% Filtrelerin uygulanması
I2 = imfilter(I1,w1,'replicate'); I3 = imfilter(I1,w2,'replicate');
% Yeni görüntülerin gösterilmesi
figure, imshow(I2,[]);
figure, imshow(I3,[]);
figure, imshow(I1,[]);
```



Şekil 5-Orjinal Görüntü



Şekil 5- 2x2 filtre uygulanmış



Şekil 6- 5x5 filtre uygulanmış

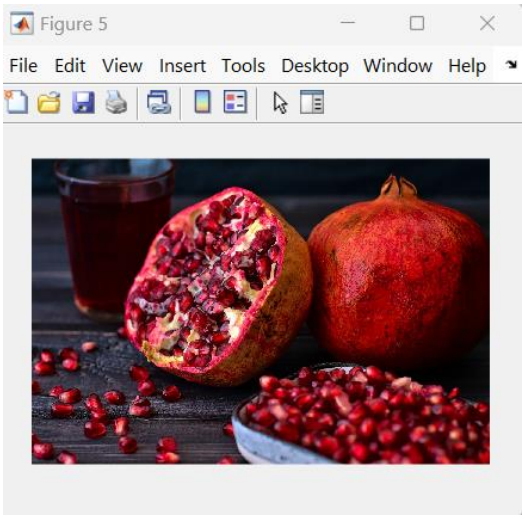
Python Kodu:

```
import cv2

import matplotlib.pyplot as plt

I1 = cv2.imread('nar.jpg')

w1 = np.ones((2, 2), dtype=np.float32)  # 2x2 filtre

w2 = np.ones((5, 5), dtype=np.float32)  # 5x5 filtre

I2 = cv2.filter2D(I1, -1, w1)  # 2x2 filtre

I3 = cv2.filter2D(I1, -1, w2)  # 5x5 filtre

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# 2x2 filtreli görüntü

axes[0].imshow(cv2.cvtColor(I2, cv2.COLOR_BGR2RGB))

axes[0].set_title('2x2 Filtrelenmiş Görüntü')

axes[0].axis('off')  # Eksenleri kapatıyoruz

# 5x5 filtreli görüntü

axes[1].imshow(cv2.cvtColor(I3, cv2.COLOR_BGR2RGB))  # OpenCV BGR formatını RGB'ye dönüştürerek gösteriyoruz

axes[1].set_title('5x5 Filtrelenmiş Görüntü')

axes[1].axis('off')  # Eksenleri kapatıyoruz

# Orijinal görüntü

axes[2].imshow(cv2.cvtColor(I1, cv2.COLOR_BGR2RGB))  # Orijinal görüntüyü RGB formatında gösteriyoruz

axes[2].set_title('Orijinal Görüntü')

axes[2].axis('off')  # Eksenleri kapatıyoruz

plt.tight_layout()  # Görüntülerin düzenini iyileştiriyoruz

plt.show()
```

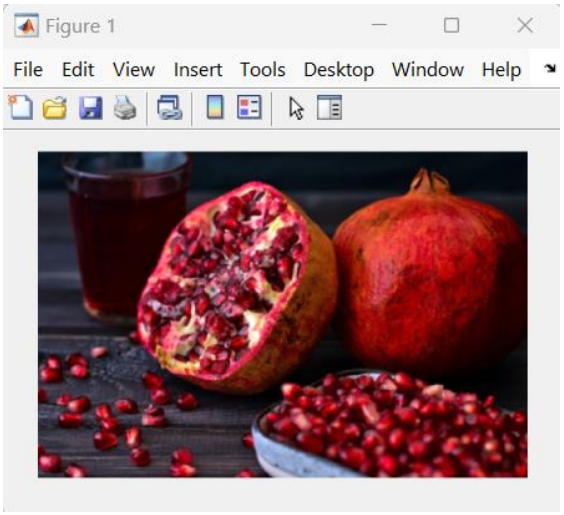
6)Gaussian (Gauss) alçak geçiren filtresi

Gaussian (Gauss) alçak geçiren filtresi, dijital görüntü işleme ve sinyal işleme alanında yaygın olarak kullanılan bir yumuşatma filtresidir. Bu filtre, görüntüdeki yüksek frekans bileşenlerini (örneğin, gürültü ve ani kenar geçişleri) bastırırken düşük frekans bileşenlerini (düzgün ve geniş renk değişimleri) korur.

```
% Görüntünün okunup double tipine veri dönüşümünün yapılması

I1 = im2double(imread('nar.jpg'));
% Filtrenin oluşturulması
h=fspecial('gaussian',10,2)
% Filtrenin uygulanması
I2=imfilter(I1,h,'replicate');
figure,imshow(I2,[]);
```



Şekil 6-Orijinal Görüntü



Şekil 6- Gauss Filtresi Uygulanmış Görüntü

Python kodu:

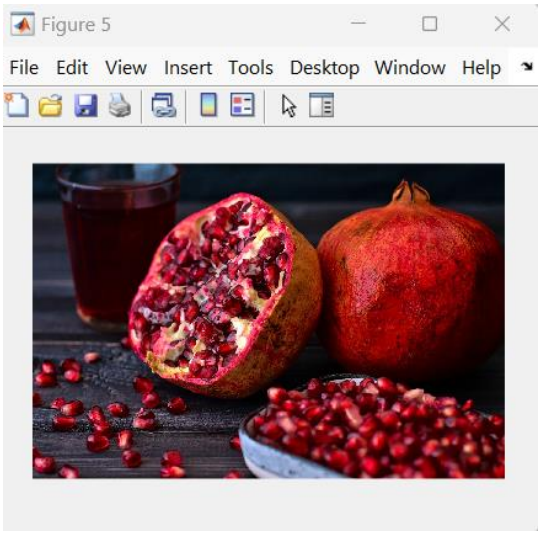
```
 import cv2

import matplotlib.pyplot as plt

import numpy as np


def apply_gaussian_filter_manual(image, sigma):

  # Gaussian kernel oluşturma

  kernel_size = int(6 * sigma + 1)

  if kernel_size % 2 == 0:

    kernel_size += 1


  ax = np.arange(-(kernel_size // 2), kernel_size // 2 + 1)

  xx, yy = np.meshgrid(ax, ax)

  kernel = np.exp(-(xx**2 + yy**2) / (2 * sigma**2))

  kernel = kernel / np.sum(kernel)


  # Pad işlemi (kenarlarda taşmayı önlemek için)

  pad_h = kernel_size // 2

  pad_w = kernel_size // 2

  padded_image = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w), (0, 0)), mode='constant', constant_values=0)


  # Çıkış görüntüsü

  filtered_image = np.zeros_like(image, dtype=np.uint8)


  # Her kanal için filtreleme işlemi

  for c in range(image.shape[2]):  # R, G, B kanalları

    for i in range(image.shape[0]):

      for j in range(image.shape[1]):
```

```
            region = padded_image[i:i+kernel_size, j:j+kernel_size, c]
            filtered_value = np.sum(region * kernel)
            filtered_image[i, j, c] = min(255, max(0, int(filtered_value)))  # Piksel değerlerini sınırla

    return filtered_image

def display_images_with_filter(original, filtered, sigma):
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(original, cv2.COLOR_BGR2RGB))  # Orijinal renkli resmi göster
    plt.title('Orjinal görsel')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(filtered, cv2.COLOR_BGR2RGB))  # Filtrelenmiş renkli resmi göster
    plt.title('Gaus filitreli görsel ')
    plt.axis('off')

    plt.show()

# Kullanım
image_path = 'nar.jpg'  # Resim dosyasının yolu
original_image = cv2.imread(image_path)  # OpenCV kullanılarak görüntüyü oku

# Gaussian filtresi
sigma = 3
filtered_image_manual = apply_gaussian_filter_manual(original_image, sigma)  # Renkli resme Gaussian filtreleme

display_images_with_filter(original_image, filtered_image_manual, sigma)
```

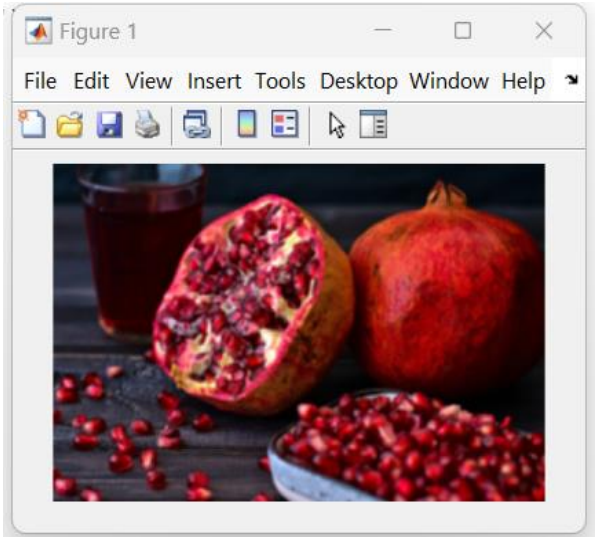
7)Mean Filtresi

Mean filtresi, görüntü işleme uygulamalarında kullanılan basit bir **alçak geçiren (low-pass)** filtredir. Bu filtre, bir pikselin değerini, o pikselin komşularıyla birlikte **aritmetik ortalamasını alarak** yeniden hesaplar. Bu işlem, görüntüdeki yüksek frekanslı bileşenleri (gürültü ve keskin kenar geçişleri) bastırırken, düşük frekanslı bileşenleri (genel renk tonları) korur.

```
% Görüntünün okunup double tipine veri dönüşümünün yapılması
I1 = im2double(imread('nar.jpg'));
% Filtrenin oluşturulması
g=fspecial('average',10)
% Filtrenin uygulanması
I2=imfilter(I1,g,'replicate');
figure,imshow(I2,[]);
```



Şekil 7-Orijinal Görüntü



Şekil 7- Mean Filtresi uygulanmış

Python kodu:

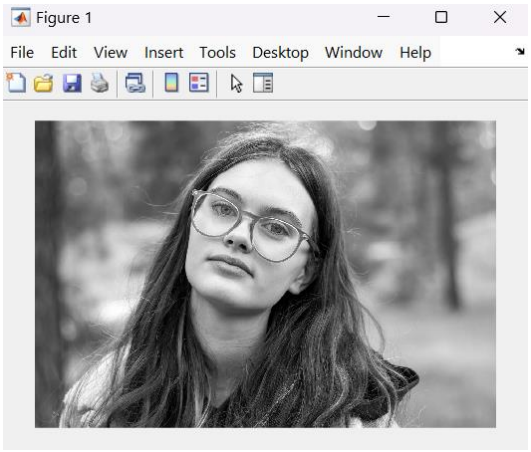
```
import cv2

import matplotlib.pyplot as plt

import numpy as np


def apply_mean_filter_manual_color(image, kernel_size):

  # Pad işlemi (kenarlarda taşmayı önlemek için)

  pad_h = kernel_size // 2

  pad_w = kernel_size // 2

  padded_image = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w), (0, 0)), mode='constant', constant_values=0)


  # Çıkış görüntüsü

  filtered_image = np.zeros_like(image, dtype=np.uint8)


  # Ortalama (mean) filtre kernel oluşturma

  kernel = np.ones((kernel_size, kernel_size), dtype=np.float32) / (kernel_size**2)


  # Her kanal için filtreleme

  for c in range(3):  # Renk kanalları (B, G, R)

    for i in range(image.shape[0]):

      for j in range(image.shape[1]):

        region = padded_image[i:i+kernel_size, j:j+kernel_size, c]

        filtered_value = np.sum(region * kernel)

        filtered_image[i, j, c] = min(255, max(0, int(filtered_value)))  # Piksel değerlerini sınırla


  return filtered_image
```

```
def display_original_and_filtered(original, filtered):
    # Görüntüleri yan yana göster
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(original, cv2.COLOR_BGR2RGB))
    plt.title('Original Image')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(filtered, cv2.COLOR_BGR2RGB))
    plt.title('Mean Filtered Image')
    plt.axis('off')

    plt.show()

# Kullanım
image_path = 'nar.jpg'
original_image = cv2.imread(image_path)  # OpenCV kullanılarak görüntüyü oku

# Ortalama filtreyi renkli görüntüye uygula
kernel_size = 5
filtered_image_color = apply_mean_filter_manual_color(original_image, kernel_size)

# Görüntüleri göster
display_original_and_filtered(original_image, filtered_image_color)
```

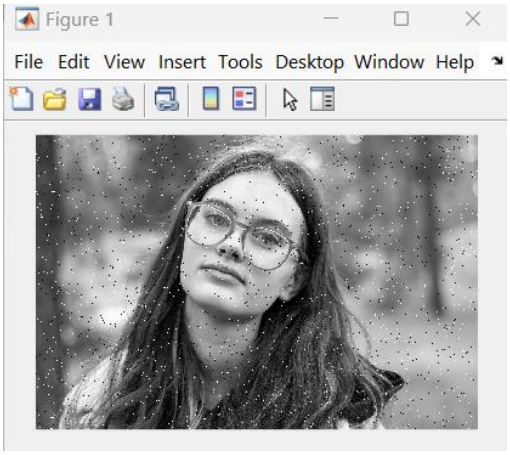
8)Medyan filtresi

Medyan filtresi ,görüntü işleme alanında kullanılan bir **alçak geçiren (low-pass)** filtredir. Görüntüdeki rastgele gürültüyü azaltmak için özellikle etkilidir. Bu filtre, bir pikselin değerini, o pikselin komşu piksellerinin **ortanca (median)** değeri ile değiştirir. Bu işlem, ani parlaklık değişimlerini ve özellikle "tuz ve biber gürültüsü" olarak bilinen rastgele beyaz ve siyah pikselleri etkili bir şekilde ortadan kaldırır.

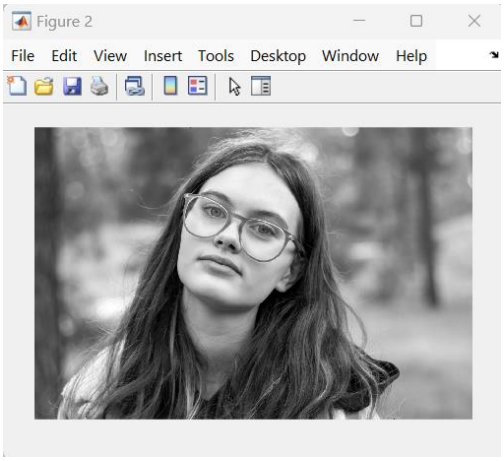
```
 % Görüntünün okunması

I1 = imread('womann.jpg');

% Resmi gri tonlamaya çevirme

I1_gray = rgb2gray(I1);

% Resim üzerinde siyah beyaz noktaların oluşturulması

I2 = imnoise(I1_gray, 'salt & pepper', 0.02);

% Filtrenin uygulanması

I3 = medfilt2(I2);

% Yeni görüntülerin gösterilmesi

imshow(I2)

figure, imshow(I3)
```



Şekil 8-Orijinal Görüntü



Şekil 8-Salt-and-pepper noise Uyg.



Şekil 8-Median Filtresi Uyg. Görüntü

Python kodu:

Python:

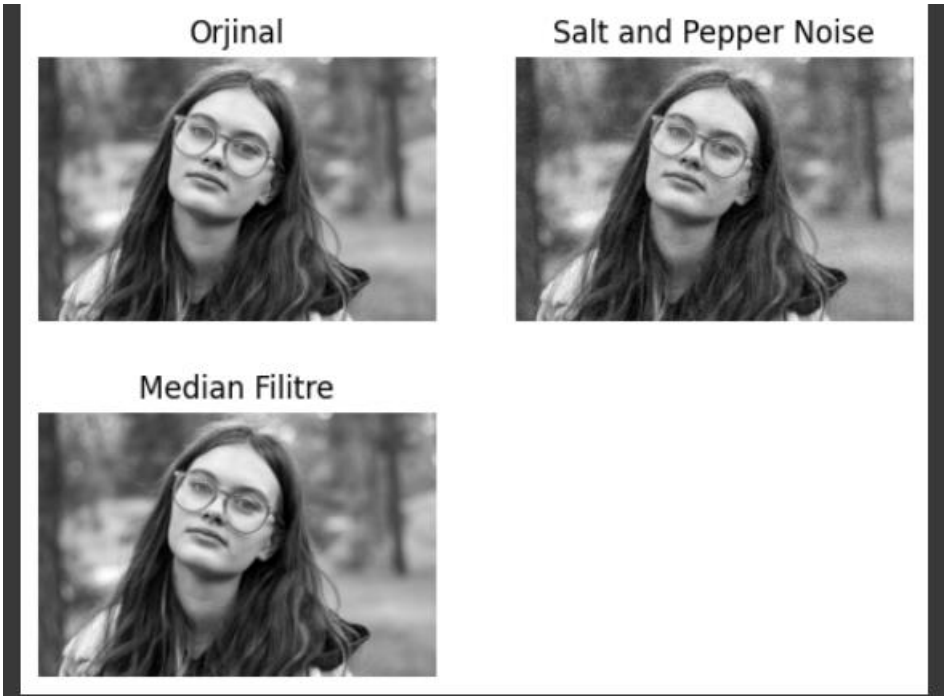
```
import cv2

import matplotlib.pyplot as plt

import numpy as np

import random
```

```
def rgb_to_grayscale_manual(image):
    # Görüntüyü gri tonlamaya dönüştürme (manuel yöntem)
    height, width, channels = image.shape
    gray_image = np.zeros((height, width), dtype=np.uint8)

    for i in range(height):
        for j in range(width):
            r, g, b = image[i, j]  # R, G, B bileşenlerini al
            # Gri tonlama formülü uygula
            gray_value = int(0.2989 * r + 0.5870 * g + 0.1140 * b)
            gray_image[i, j] = gray_value

    return gray_image

def apply_mean_filter_manual(image, kernel_size):
    # Ortalama (mean) filtre kernel oluşturma
    kernel = np.ones((kernel_size, kernel_size), dtype=np.float32) / (kernel_size**2)

    # Pad işlemi (kenarlarda taşmayı önlemek için)
    pad_h = kernel_size // 2
    pad_w = kernel_size // 2
    padded_image = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)), mode='constant',
constant_values=0)

    # Çıkış görüntüsü
```

```
    filtered_image = np.zeros_like(image, dtype=np.uint8)

    # Filtreleme işlemi
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            region = padded_image[i:i+kernel_size, j:j+kernel_size]
            filtered_value = np.sum(region * kernel)
            filtered_image[i, j] = min(255, max(0, int(filtered_value)))  # Piksel değerlerini sınırla

    return filtered_image

def salt_and_pepper_noise(image, prob):
    # Salt and Pepper noise ekleme
    output = np.zeros(image.shape, np.uint8)
    thres = 1 - prob
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            rdn = random.random()
            if rdn < prob:
                output[i][j] = 0
            elif rdn > thres:
                output[i][j] = 255
            else:
                output[i][j] = image[i][j]
    return output
```

```
def display_images(original, noisy, filtered):
    plt.subplot(2, 2, 1)
    plt.imshow(original, cmap='gray')
    plt.title('Orjinal')
    plt.axis('off')

    plt.subplot(2, 2, 2)
    plt.imshow(noisy, cmap='gray')
    plt.title('Salt and Pepper Noise')
    plt.axis('off')

    plt.subplot(2, 2, 3)
    plt.imshow(filtered, cmap='gray')
    plt.title('Median Filitre')
    plt.axis('off')

    plt.show()

# Kullanım
image_path = 'womann.jpg'
original_image = cv2.imread(image_path)  # OpenCV kullanılarak görüntüyü oku
grayscale_image_manual = rgb_to_grayscale_manual(original_image)  # Manuel gri tonlama

# Salt and Pepper Noise ekle
noise_probability = 0.02
noisy_image = salt_and_pepper_noise(grayscale_image_manual, noise_probability)
```

```
# Medyan filtresi ile gürültüyü giderme

median_filtered = cv2.medianBlur(noisy_image, 5)


# Sonuçları göster

display_images(grayscale_image_manual, noisy_image, median_filtered)
```

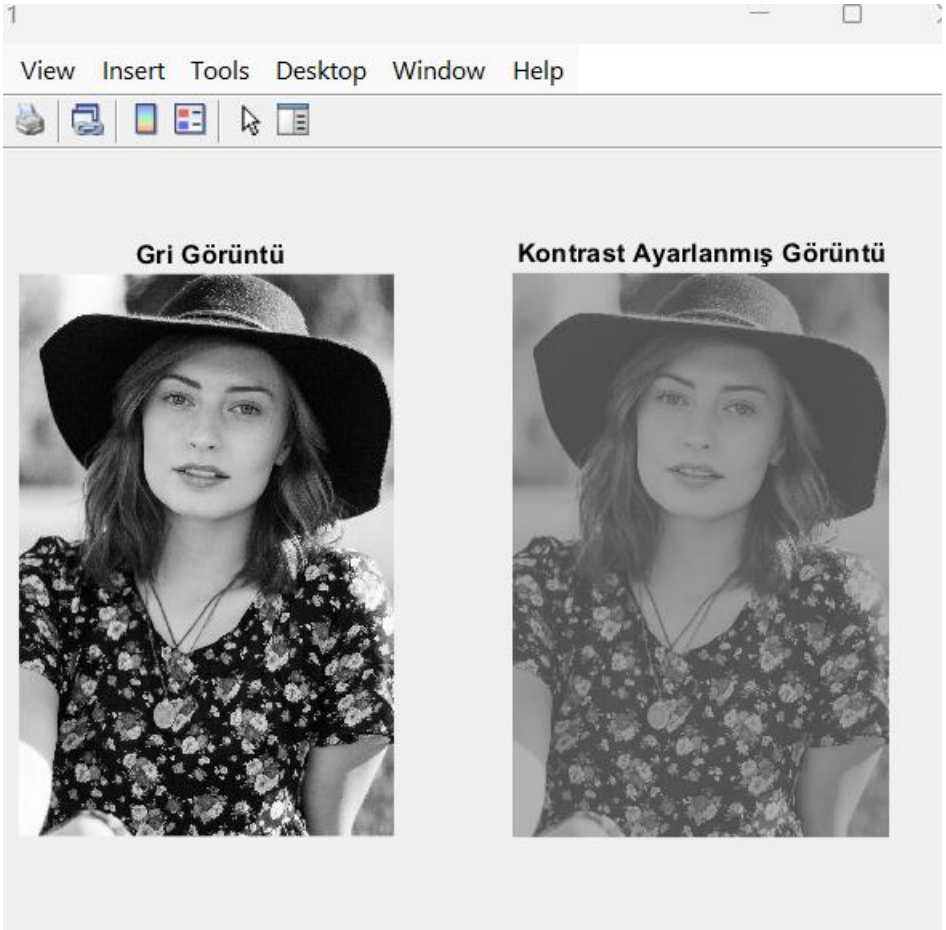


9) Kontrastı Ayarlama

Kontrast ayarlama, bir görüntüdeki en parlak ve en koyu bölgeler arasındaki farkı artırarak veya azaltarak görüntünün netliğini ve görsel etkisini değiştirme işlemidir. Yüksek kontrast, görüntüdeki parlak ve koyu alanların belirgin bir şekilde ayrılmasını sağlar, bu da daha keskin bir görüntüye yol açar. Düşük kontrast ise, parlak ve koyu alanlar arasındaki farkı azaltır, böylece daha düz ve yumuşak bir görüntü oluşturur. Kontrast ayarlama, genellikle görüntülerin daha iyi görünmesi veya belirli detayların daha belirgin hale getirilmesi için kullanılır.

```
orjinalresim = imread('woman.jpg');

griresim = rgb2gray(orjinalresim);

yeni_min = 0.3;
```

```
yeni_max = 0.7;

kontrastliresim = imadjust(griresim, [min(griresim(:))/255, max(griresim(:))/255],
[yeni_min, yeni_max]);

figure; subplot(1, 2, 1); imshow(griresim); title(' Gri Görüntü');

subplot(1, 2, 2); imshow(kontrastliresim); title('Kontrast Ayarlanmış Görüntü');
```



Matlab Kodu:

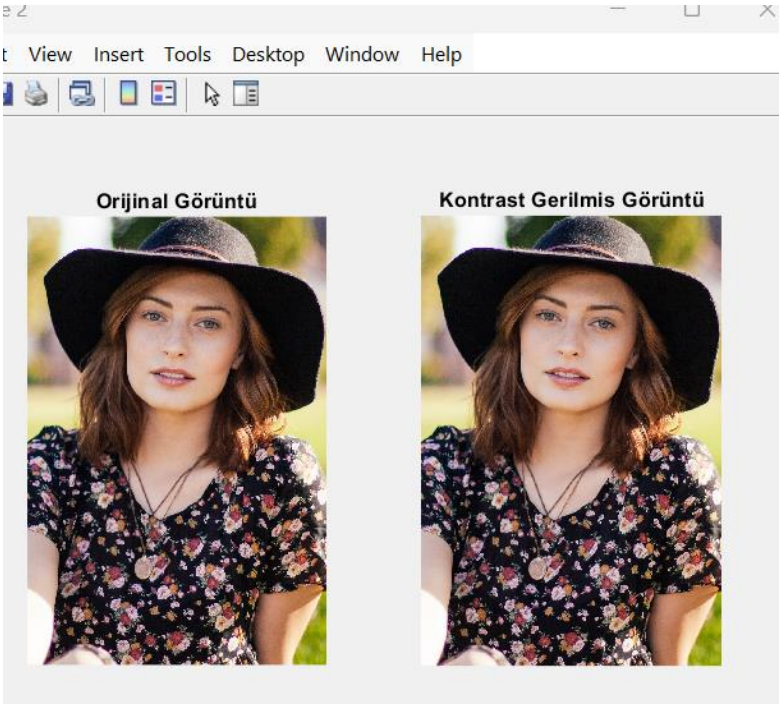
```
from PIL import Image

import matplotlib.pyplot as plt

def kontrast_ayarla(resim_yolu, faktor):

  giris_resmi = Image.open(resim_yolu).convert('L')
```

```
    kontrastli_resim = giris_resmi.copy()

    genislik, yukseklik = kontrastli_resim.size

    for x in range(genislik):
        for y in range(yukseklik):
            piksel_degeri = kontrastli_resim.getpixel((x, y))
            yeni_deger = int(piksel_degeri * faktor)
            kontrastli_resim.putpixel((x, y), yeni_deger)

    pencere_genisligi = genislik * 2
    pencere_yuksekligi = yukseklik

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(giris_resmi, cmap='gray')
    plt.title("Orijinal Resim")
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.imshow(kontrastli_resim, cmap='gray')
    plt.title("Kontrast Ayarlanmış Resim")
    plt.axis('off')

    plt.show()
goruntuyolu = 'woman.jpg'
kontrast_faktoru = 1.5
kontrast_ayarla(goruntuyolu, kontrast_faktoru)
```

10)Kontrast Germe

Kontrast germe, bir görüntüdeki parlaklık değerlerinin daha geniş bir aralığa yayılmasını sağlayarak görüntünün kontrastını artırma işlemidir. Bu işlem, görüntüdeki en düşük ve en yüksek parlaklık değerlerinin (gölge ve aydınlık) daha belirgin hale gelmesini sağlar. Kontrast germe ile, düşük parlaklık değerleri daha koyu, yüksek parlaklık değerleri ise daha parlak hale gelir, böylece görüntüdeki detaylar daha net bir şekilde görünür. Genellikle, düşük kontrastlı görüntüleri daha canlı ve keskin hale getirmek için kullanılır.

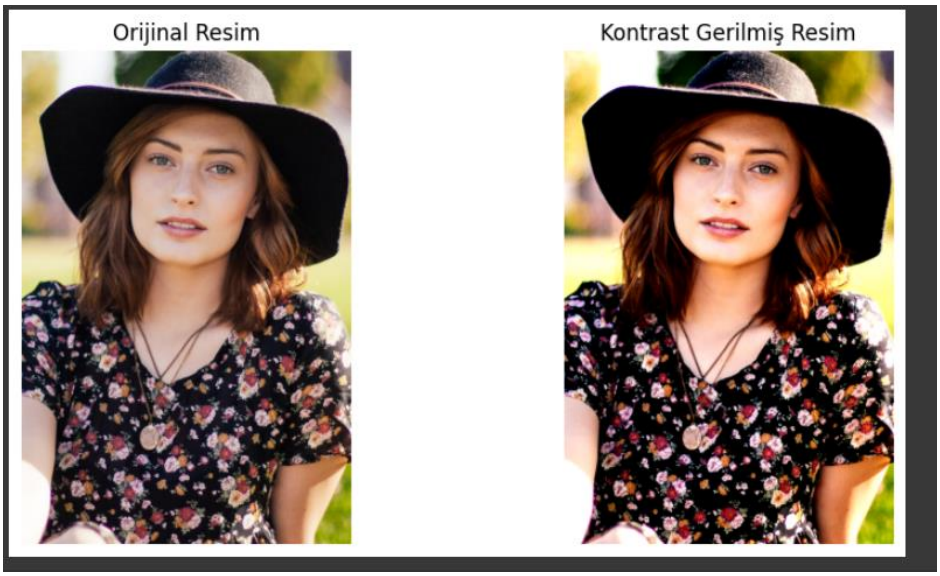
```
I1 = imread('woman.jpg');

min_val = double(min(I1(:)));

max_val = double(max(I1(:)));

I2 = (double(I1) - min_val) * (255 / (max_val - min_val));

figure;

subplot(1,2,1); imshow(I1); title('Orijinal Görüntü');

subplot(1,2,2); imshow(uint8(I2)); title('Kontrast Gerilmis Görüntü');
```



Python kodu:

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
def artir_kontrast(piksel, faktor):
    return int(np.clip(faktor * (piksel - 128) + 128, 0, 255))
def kontrastli_resim_olustur(orijinal_resim, faktor):
    genislik, yukseklik = orijinal_resim.size
    kontrastli_resim = Image.new('RGB', (genislik, yukseklik))

    for x in range(genislik):
        for y in range(yukseklik):
            r, g, b = orijinal_resim.getpixel((x, y))
            r = artir_kontrast(r, faktor)
```

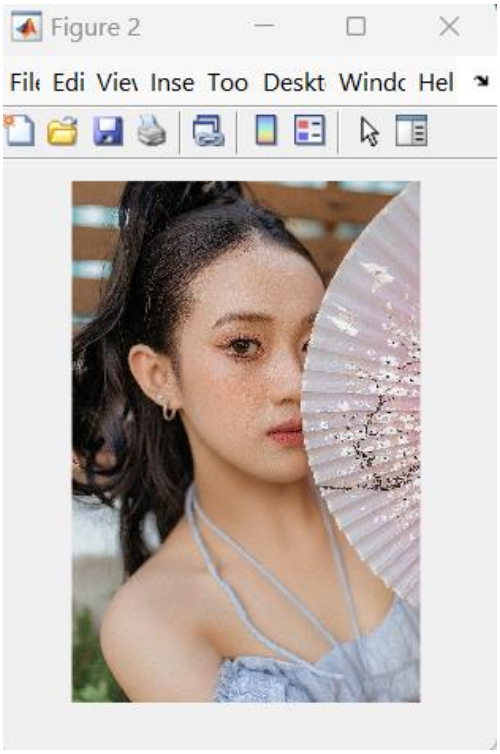
```
        g = artir_kontrast(g, faktor)
        b = artir_kontrast(b, faktor)
        kontrastli_resim.putpixel((x, y), (r, g, b))

    return kontrastli_resim
goruntuyolu = "woman.jpg"
orijinal_resim = Image.open(goruntuyolu)
kontrast_faktoru = 1.5
kontrastli_resim = kontrastli_resim_olustur(orijinal_resim, kontrast_faktoru)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(orijinal_resim)
plt.title("Orijinal Resim")
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(kontrastli_resim)
plt.title("Kontrast Gerilmiş Resim")
plt.axis('off')
plt.show()
```



10) Histogram eşitleme/dengeleme

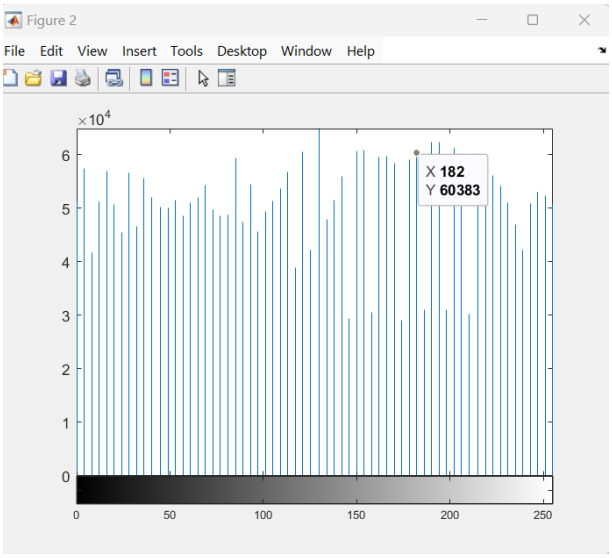
Histogram eşitleme , bir görüntünün parlaklık dağılımını iyileştirerek kontrastı artırmayı amaçlayan bir görüntü işleme tekniğidir. Bu işlem, görüntünün parlaklık değerlerini (yoğunluk seviyelerini) yeniden dağıtarak, tüm parlaklık aralığının (örneğin 0–255) daha etkin bir şekilde kullanılmasını sağlar. Sonuç olarak, görüntünün daha fazla detayı ortaya çıkar ve daha net bir hale gelir.

```
% Görüntünün okunması
I1 = imread('kadin.jpg');
% Histogram eşitlemenin uygulanması
I2 = histeq(I1);
% Histogram eşitlenmiş görüntü
figure, imshow(I2)
% Yeni görüntünün histogramı
figure; imhist(I2)
```



Şekil 10-Orijinal görüntü

Şekil 10-Histogram eşitlemesi uygulamış görüntü



Şekil 10-Yeni görüntünün histogramı

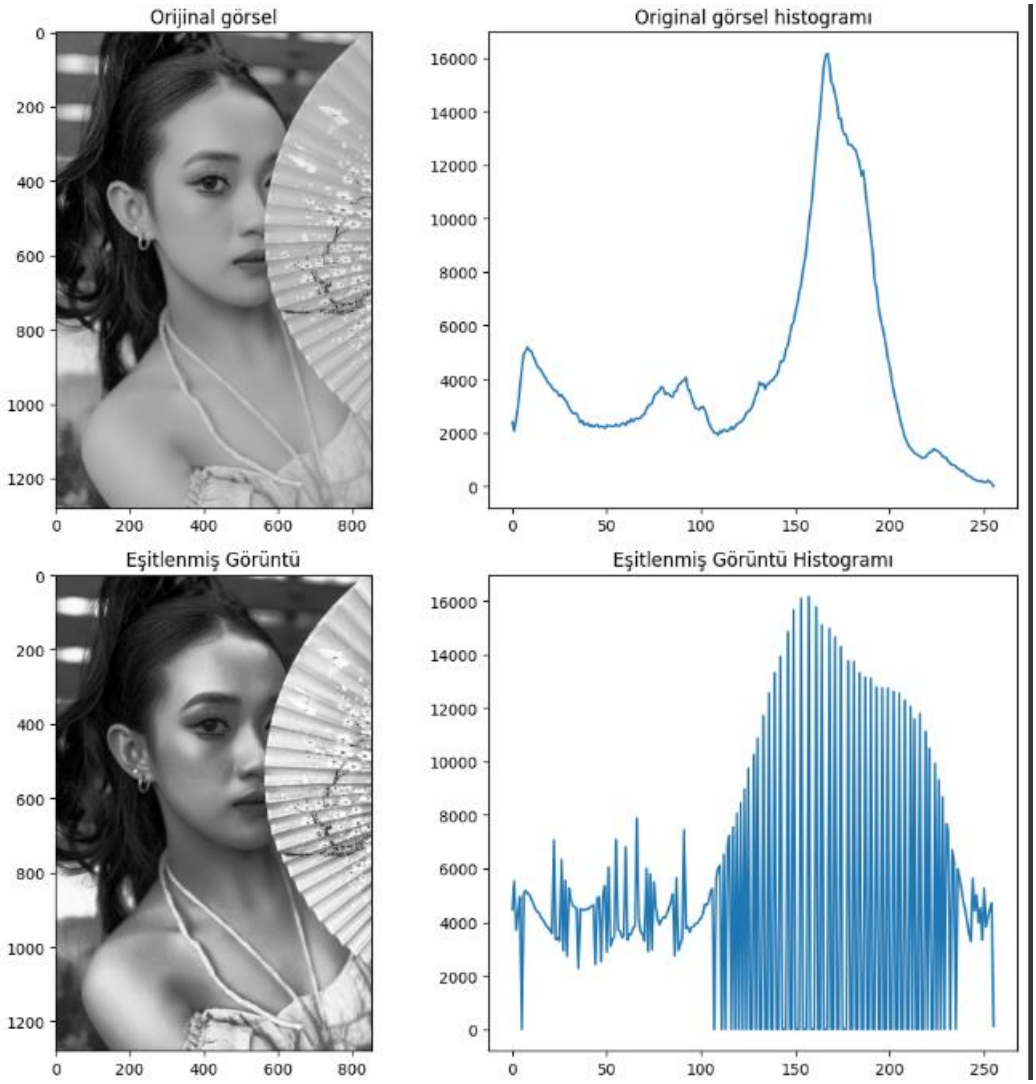
Python Kodu:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

```
# Resmi yükle
img = cv2.imread('kadin.jpg')  # OpenCV ile resmi yükle
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # OpenCV, resmi BGR formatında okur, RGB'ye dönüştürüyoruz

# 1. Gri tonlamaya dönüştürme (RGB'den gri tonlama)
def rgb_to_gray(img):
    # RGB'den gri tonlamaya dönüştürme (0.2989 * R + 0.5870 * G + 0.1140 * B)
    gray_img = np.dot(img[...,:3], [0.2989, 0.5870, 0.1140])
    return gray_img.astype(np.uint8)

gray_image = rgb_to_gray(img)

# 2. Histogram hesaplama (manüel)
def calculate_histogram(image):
    # Gri tonlama değerlerinin sıklığını hesapla (0-255 arası)
    hist = np.zeros(256, dtype=int)
    for pixel in image.flatten():
        hist[pixel] += 1
    return hist

img_hist = calculate_histogram(gray_image)

# 3. Histogram eşitleme (kontrast arttırma)
def histogram_equalization(image):
    # Görüntüdeki her pikselin yoğunluğunu histogram eşitlemesi ile değiştir
    hist = np.zeros(256, dtype=int)
    for pixel in image.flatten():
        hist[pixel] += 1

    # Kümülatif Dağılım Fonksiyonu (CDF) hesapla
```

```
    cdf = hist.cumsum()

    # CDF'yi normalize et (0-255 aralığında)
    cdf_normalized = (cdf - cdf.min()) * 255 / (cdf.max() - cdf.min())
    cdf_normalized = cdf_normalized.astype(np.uint8)

    # Yeni görüntüyü oluştur
    img_equalized = cdf_normalized[image]

    return img_equalized

img_contrast_hist_equal = histogram_equalization(gray_image)

# 4. Yeni histogramı hesapla (kontrastlı görüntü için)
img_contrast_hist = calculate_histogram(img_contrast_hist_equal)

# 5. Sonuçları görselleştir
plt.figure(figsize=(10, 10))

# Orijinal gri tonlamalı görüntü
plt.subplot(2, 2, 1)
plt.imshow(gray_image, cmap='gray')
plt.title('Orijinal görsel')

# Orijinal histogram
plt.subplot(2, 2, 2)
plt.plot(img_hist)
plt.title('Original görsel histogramı')

# Kontrastlı gri tonlamalı görüntü
plt.subplot(2, 2, 3)
```

```
plt.imshow(img_contrast_hist_equal, cmap='gray')
plt.title('Eşitlenmiş Görüntü')

# Kontrastlı görüntü histogramı
plt.subplot(2, 2, 4)
plt.plot(img_contrast_hist)
plt.title('Eşitlenmiş Görüntü Histogramı')

plt.tight_layout()
plt.show()
```

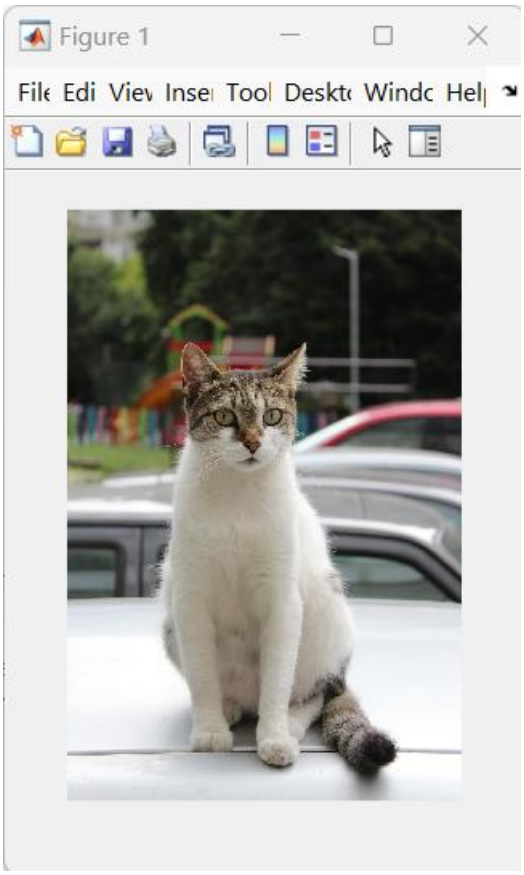


Şekil 11- Orijinal görüntü ve histogram eşitlemesi uygulanmış görüntü

11) Laplacian (Laplas) Filtresi Nedir?

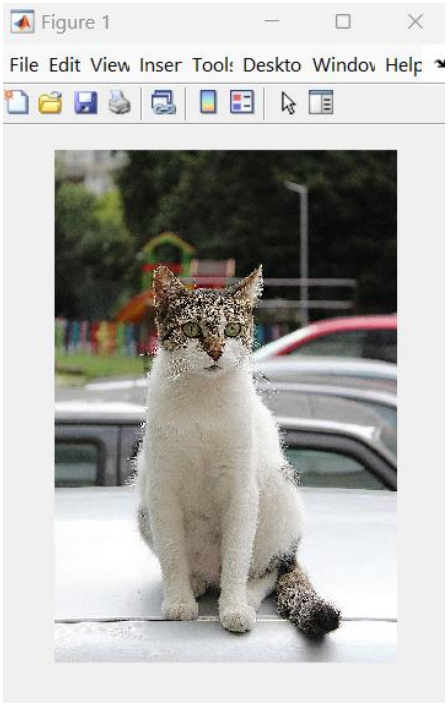
Laplacian filtresi, görüntü işleme ve kenar algılama tekniklerinde kullanılan bir **ikinci dereceden türevsel** filtredir. Görüntüdeki **parlaklık değişimlerini** analiz ederek kenarları ve ince detayları vurgular. Bu filtre, **laplas operatörü** adı verilen matematiksel bir işlemle gerçekleştirilir ve bir pikselin parlaklık değerini, komşu piksellerle olan farkına göre hesaplar.

```
% Görüntünün okunup double tipine veri dönüşümünün yapılması
I1 = im2double(imread('cat.jpg'));
% Filtrelerin oluşturulması
w4=fspecial('laplacian',0);
w8 = [1 1 1; 1 -8 1; 1 1 1];
% Filtrelerin uygulanması
g=imfilter(I1,w4,'replicate');
g4=I1-imfilter(I1,w4,'replicate');
g8=I1-imfilter(I1,w8,'replicate');
% Yeni görüntülerin gösterilmesi

imshow(I1);

imshow(g);

imshow(g4);

imshow(g8);
```

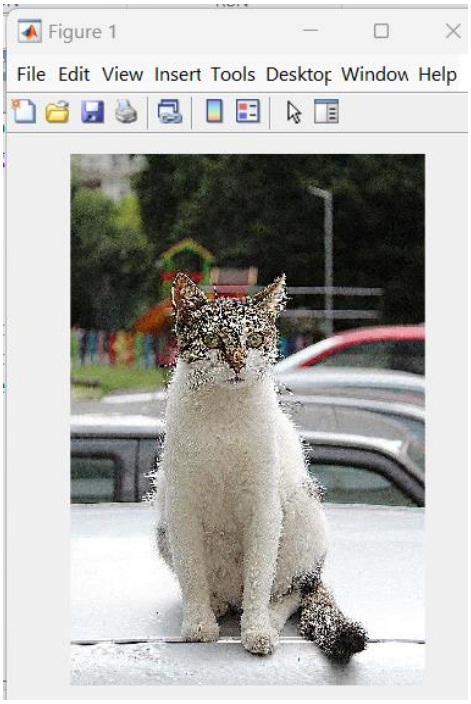


Şekil 11-Orijinal görüntü

Şekil 11-laplace filtresi uygulanmış görüntü



Şekil 11-w4 laplace filtresi



Şekil 11-w8 laplace filtresi

Python Kodu:

```
from PIL import Image
import matplotlib.pyplot as plt

def laplace_filtre(resim):

    genislik, yukseklik = resim.size
    gri_resim = resim.convert("L")  # Resmi gri tonlamaya dönüştürüyoruz (L: grayscale)

    # Laplacian filtre matrisi (3x3)
    laplace_cerceve = [[0, 1, 0],
                       [1, -4, 1],
                       [0, 1, 0]]

    sonuc_resim = Image.new("L", (genislik, yukseklik))

    # Resmin her pikseli için Laplacian filtresini uyguluyoruz
    for x in range(1, genislik-1):  # İlk ve son pikselleri dışlıyoruz, çünkü onlar için çevre pikseller yok
        for y in range(1, yukseklik-1):
            # Filtresi uygulamak için 3x3'lük pencere kullanıyoruz
            pixel = sum([gri_resim.getpixel((x + i, y + j)) * laplace_cerceve[i+1][j+1]
                         for i in range(-1, 2) for j in range(-1, 2)])
            # Hesaplanan değeri 0-255 aralığında sınırlı bir piksel değeri olarak koyuyoruz
            sonuc_resim.putpixel((x, y), int(pixel))

    return sonuc_resim  # Filtrelenmiş resmi döndürüyoruz

goruntuyolu = "cat.jpg"
giris_resmi = Image.open(goruntuyolu)

laplace_resim = laplace_filtre(giris_resmi)
```

```
plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)
plt.imshow(giris_resmi, cmap='gray')  # Orijinal resmi gri tonlamada gösteriyoruz
plt.title('Orijinal Görüntü')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(laplace_resim, cmap='gray')  # Filtre uygulanmış resmi gri tonlamada gösteriyoruz
plt.title('Laplas Filtreli Görüntü')
plt.axis('off')

plt.show()
```

12)Sobel Filitresi

Sobel filtresi, görüntü işleme alanında kullanılan bir kenar algılama yöntemidir. Görüntülerdeki kenarları (yani parlaklık değişimlerinin yoğun olduğu bölgeleri) tespit etmek için yatay ve dikey doğrultulardaki parlaklık değişimlerini hesaplar. Sobel filtresi, genellikle görüntüdeki nesnelerin sınırlarını belirlemek, şekilleri çıkarmak veya nesne algılama gibi uygulamalarda kullanılır.

Nasıl Çalışır?

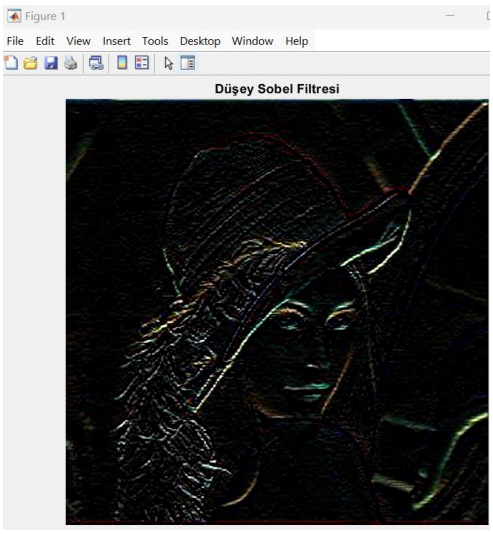
Sobel filtresi, bir görüntü üzerinde iki ayrı filtre uygular:

1. **Gx (Yatay filtre):** Görüntünün yatay kenarlarını algılar.
2. **Gy (Dikey filtre):** Görüntünün dikey kenarlarını algılar.

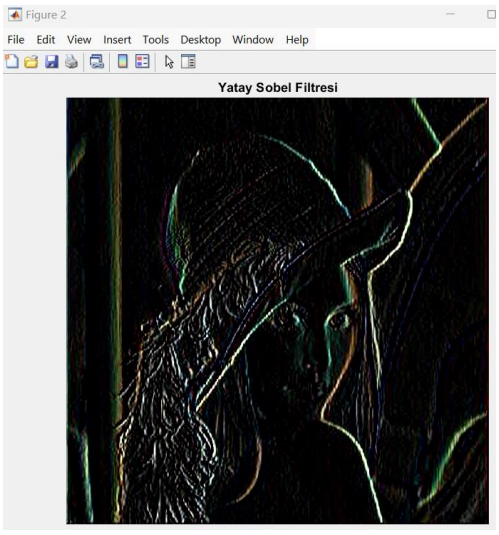
Bu iki filtre, bir görüntü üzerindeki parlaklık değişimlerini (gradyanları) belirlemek için kullanılır.

Yatay ve düşey sobel filtrenin uygulanmasının kodu:

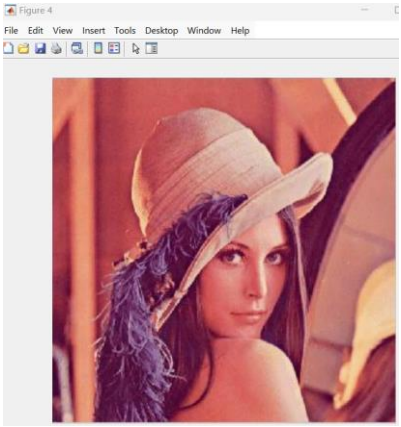
```
% Görüntünün okunup double tipine veri dönüşümünün yapılması
I1 = im2double(imread('lena.bmp'));

% Düşey Sobel filtresinin oluşturulması
h_dikey = fspecial('sobel');

% Yatay Sobel filtresinin oluşturulması
h_yatay = h_dikey';

% Düşey Sobel filtresinin uygulanması
I2_dikey = imfilter(I1, h_dikey, 'replicate');

% Yatay Sobel filtresinin uygulanması
I2_yatay = imfilter(I1, h_yatay, 'replicate');

% Sonuçların gösterilmesi
figure, imshow(I2_dikey, []), title('Düşey Sobel Filtresi');
figure, imshow(I2_yatay, []), title('Yatay Sobel Filtresi');
```



Şekil 12-Düşey Sobel Filtresi



Şekil 12-Yatay Sobel Filtres



Şekil 12-Orijinal Görüntü

Python Kodu:

```
 from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
def sobel_flt(grt_yolu):
   giris_resmi = Image.open(grt_yolu)
   resim_genisligi, resim_yuksekligi = giris_resmi.size
   cikis_resmi_xy = Image.new("RGB", (resim_genisligi, resim_yuksekligi))
   sablon_boyutu = 3  # 3x3 boyutunda bir filtre
   matris_x = [-1, 0, 1, -2, 0, 2, -1, 0, 1]  # Düşey Sobel filtresi (X yönü)
   matris_y = [1, 2, 1, 0, 0, 0, -1, -2, -1]  # Yatay Sobel filtresi (Y yönü
   # Resmin her pikseli için Sobel filtrelerini uygula
   for x in range((sablon_boyutu - 1) // 2, resim_genisligi - (sablon_boyutu - 1) // 2):
      for y in range((sablon_boyutu - 1) // 2, resim_yuksekligi - (sablon_boyutu - 1) // 2):
         toplam_gri_x, toplam_gri_y = 0, 0  # Sobel filtrelerinin geçişiyle hesaplanan griler
         k = 0  # Matrisin indeksi
         # 3x3'lük pencerede hareket et
         for i in range(-((sablon_boyutu - 1) // 2), (sablon_boyutu - 1) // 2 + 1):
            for j in range(-((sablon_boyutu - 1) // 2), (sablon_boyutu - 1) // 2 + 1):
               # Pikseli oku ve gri tonlamaya dönüştür
               okunan_renk = giris_resmi.getpixel((x + i, y + j))
               if isinstance(okunan_renk, int):
                  gri = okunan_renk  # Gri resim için direkt değeri al
               else:
                  # Renkli resim için gri tonlama dönüşümü
```

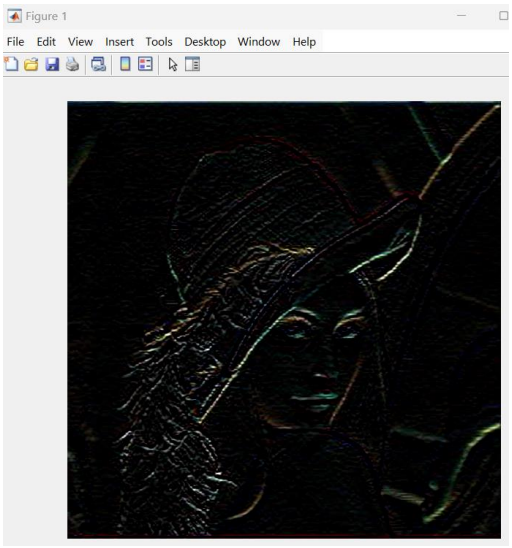
```
                gri = int(0.299 * okunan_renk[0] + 0.587 * okunan_renk[1] + 0.114 * okunan_renk[2])

                # Sobel filtrelerinin x ve y yönlerindeki geçişlerini hesapla
                toplam_gri_x += gri * matris_x[k]
                toplam_gri_y += gri * matris_y[k]
                k += 1
            # Sobel filtrelerinin birleşimiyle yeni piksel değeri hesapla
            renk_xy = abs(toplam_gri_x) + abs(toplam_gri_y)
            # Pikselin değerini sınırlıyoruz
            if renk_xy > 255:
                renk_xy = 255
            elif renk_xy < 0:
                renk_xy = 0
            # Hesaplanan yeni pikseli çıkış resmine yerleştir
            cikis_resmi_xy.putpixel((x, y), (renk_xy, renk_xy, renk_xy))
    # Sonuçları görselleştirme için grafik oluşturuyoruz
    plt.figure(figsize=(10, 5))
    # Orijinal resmi göster
    plt.subplot(1, 2, 1)
    plt.imshow(giris_resmi, cmap='gray')
    plt.title("Orijinal Resim")
    plt.axis('off')
    # Sobel filtreli resmi göster
    plt.subplot(1, 2, 2)
    plt.imshow(cikis_resmi_xy, cmap='gray')
    plt.title("Sobel Filtreli Resim")
    plt.axis('off')
    # Görselleri ekranda göster
    plt.show()
# Fonksiyonu çağırıyoruz
sobel_flt('lena.bmp')
```

13) Prewitt Filtresi

Prewitt filtresi, görüntü işleme alanında kenar tespiti için kullanılan bir yöntemdir. Görüntüdeki parlaklık değişimlerini algılayarak kenarların bulunduğu bölgeleri belirlemek amacıyla tasarlanmıştır. Bu filtre, Sobel filtresi ile benzer bir mantıkta çalışır ancak daha basit bir çekirdek (kernel) yapısına sahiptir.

Prewitt yatay matlab kodu:

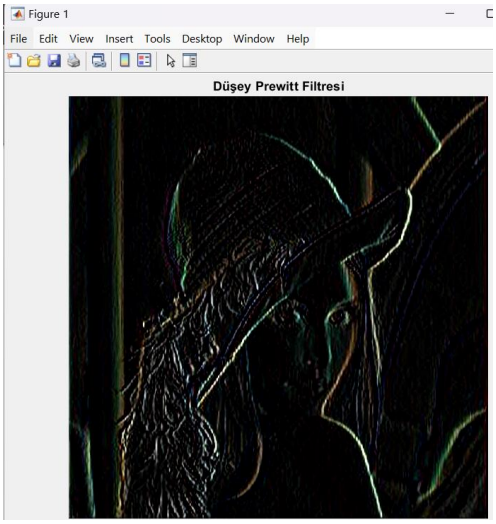
```
% Görüntünün okunup double tipine veri dönüşümünün yapılması

I1 = im2double(imread('lena.bmp'));
% Filtrenin oluşturulması
h=fspecial('prewitt');
% Filtrenin uygulanması
I2=imfilter(I1,h,'replicate');
figure,imshow(I2,[]);
```



Şekil 13-Prewitt Yatay Filitresi Uyg. Görüntü

Prewitt vertically (düşey) matlab kodu:

```
% Görüntünün okunup double tipine veri dönüşümünün yapılması

I1 = im2double(imread('lena.bmp'));

% Filtrenin oluşturulması
h = fspecial('prewitt');

% Filtrenin uygulanması (Düşey filtre için)
I2 = imfilter(I1, h', 'replicate'); % Transpoz alındı
figure, imshow(I2, []), title('Düşey Prewitt Filtresi');
```



Şekil 13-Prewitt Düşey Filtresi Uyg. Görüntü

Python Kodu:

```
from PIL import Image

import numpy as np

import matplotlib.pyplot as plt

def prewitt_filtresi(grt_yolu):

  giris_resmi = Image.open(grt_yolu)

  if giris_resmi.mode == 'L':

    resim_genisligi, resim_yuksekligi = giris_resmi.size

  else:

    giris_resmi = giris_resmi.convert('L')

    resim_genisligi, resim_yuksekligi = giris_resmi.size

  cikis_resmi = Image.new("RGB", (resim_genisligi, resim_yuksekligi))

  sablon_boyutu = 3

  for x in range((sablon_boyutu - 1) // 2, resim_genisligi - (sablon_boyutu - 1) // 2):

    for y in range((sablon_boyutu - 1) // 2, resim_yuksekligi - (sablon_boyutu - 1) // 2):

      P1 = giris_resmi.getpixel((x - 1, y - 1)) // 3
```

```
            P2 = giris_resmi.getpixel((x, y - 1)) // 3
            P3 = giris_resmi.getpixel((x + 1, y - 1)) // 3
            P4 = giris_resmi.getpixel((x - 1, y)) // 3
            P5 = giris_resmi.getpixel((x, y)) // 3
            P6 = giris_resmi.getpixel((x + 1, y)) // 3
            P7 = giris_resmi.getpixel((x - 1, y + 1)) // 3
            P8 = giris_resmi.getpixel((x, y + 1)) // 3
            P9 = giris_resmi.getpixel((x + 1, y + 1)) // 3

            Gx = abs(-P1 + P3 - P4 + P6 - P7 + P9)
            Gy = abs(P1 + P2 + P3 - P7 - P8 - P9)

            PrewittDegeri = Gx + Gy

            if PrewittDegeri > 255:
                PrewittDegeri = 255
            cikis_resmi.putpixel((x, y), (PrewittDegeri, PrewittDegeri, PrewittDegeri)
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(giris_resmi, cmap='gray')
    plt.title("Orijinal Resim")
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(cikis_resmi, cmap='gray')
    plt.title("Prewitt Filtreli Resim")
    plt.axis('off')
    plt.show()
prewitt_filtresi('lena.bmp')
```

14)MATLAB'da Geometrik İşlemler

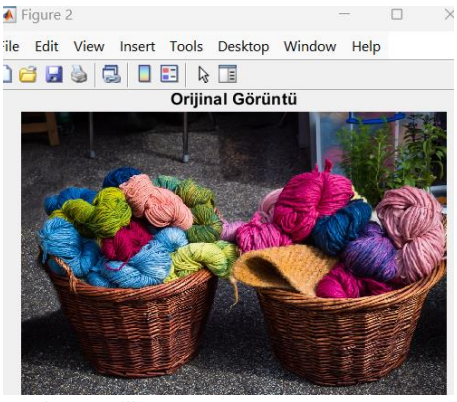
1)Açılı Döndürme

Açılı döndürme, bir görüntünün belirli bir açı etrafında saat yönünde veya saat yönünün tersine dönmesini sağlayan temel bir geometrik dönüşüm işlemidir. Bu işlem, özellikle görüntü işleme ve bilgisayarla görme uygulamalarında yaygın olarak kullanılır. MATLAB, bu işlem için kullanıcıya kolaylık sağlayan **imrotate** fonksiyonunu sunar.

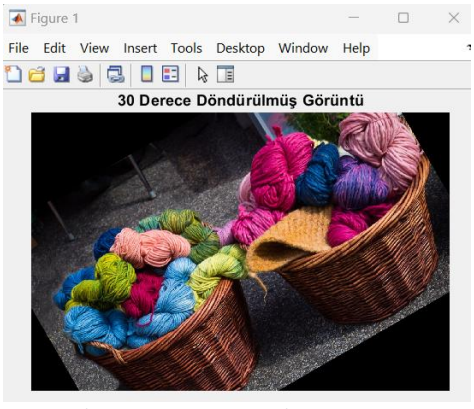
1. Açılı Döndürme İşleminin Mantığı

Açılı döndürme işlemi, görüntüdeki her bir pikselin, bir döndürme matrisi kullanılarak yeni bir konuma taşınmasıyla gerçekleştirilir.

```
% Görüntünün okunması
I = imread('ip.jpg');

% Görüntünün 30 derece döndürülmesi
rotatedImage = imrotate(I, 30, 'bilinear', 'crop');

% Orijinal ve döndürülmüş görüntünün gösterilmesi
figure;
subplot(1, 2, 1);
imshow(I);
title('Orijinal Görüntü');

subplot(1, 2, 2);
imshow(rotatedImage);
title('30 Derece Döndürülmüş Görüntü');
```



Şekil 14-Orijinal Görüntü



Şekil 14-Saat Yönü tersine 30 derece döndürme

Python:

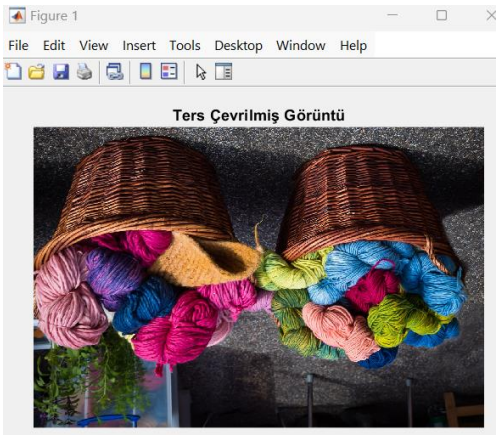
```
 import cv2
import numpy as np
import matplotlib.pyplot as plt
# Görüntüyü okuma
img = cv2.imread('ip.jpg')
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# 30 derece döndürme işlemi
rows, cols, channels = img.shape
angle_30 = np.radians(30)  # 30 dereceyi radiana çevirme
cos30 = np.cos(angle_30)
sin30 = np.sin(angle_30)
center_x, center_y = cols / 2, rows / 2
rotation_matrix_30 = np.array([
   [cos30, -sin30, (1 - cos30) * center_x + sin30 * center_y],
   [sin30, cos30, (1 - cos30) * center_y - sin30 * center_x]
])
img_rotate_30 = np.zeros_like(img_rgb)
for i in range(rows):
   for j in range(cols):
      new_coords = np.dot(rotation_matrix_30, [j, i, 1])
      new_x, new_y = int(new_coords[0]), int(new_coords[1])
      if 0 <= new_x < cols and 0 <= new_y < rows:
         img_rotate_30[new_y, new_x] = img_rgb[i, j]
# Görseli gösterme
plt.figure(figsize=(8, 8))
plt.subplot(1, 1, 1)
plt.imshow(img_rotate_30)
plt.title('30 Derece Çevrilmiş Görüntü')
plt.axis('off')
plt.tight_layout()
plt.show()
```

2)Görüntünün ters çevrilmesi

Görüntünün ters çevrilmesi için en üstteki satır en alta, en alttaki satır ise en üste gelecek şekilde görüntü matrisinin tüm elemanları karşılıklı olarak yer değiştirmesi ile sağlanır.

```
% Orijinal görüntünün yüklenmesi
I = imread('ip.jpg'); % Görüntü dosyanızın adı 'lena.bmp' olmalıdır

% Görüntünün ters çevrilmesi
I_flipped = flipud(fliplr(I)); % Hem yatay hem dikey eksende ters çevirme

% Görüntülerin gösterilmesi
figure;
subplot(1, 2, 1), imshow(I), title('Orijinal Görüntü');
subplot(1, 2, 2), imshow(I_flipped), title('Ters Çevrilmiş Görüntü');
```

Şekil 15- Orijinal Görüntü



Şekil 15-Ters Çevrilmiş Görüntü

Python:

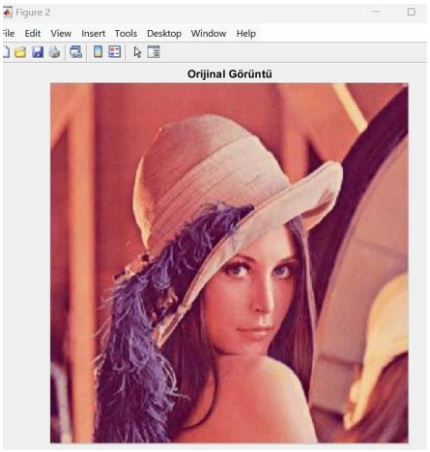
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Görüntüyü okuma
img = cv2.imread('ip.jpg')
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# 180 derece döndürme işlemi
rows, cols, channels = img.shape
angle_180 = np.radians(180)  # 180 dereceyi radiana çevirme
cos180 = np.cos(angle_180)
sin180 = np.sin(angle_180)
center_x, center_y = cols / 2, rows / 2

rotation_matrix_180 = np.array([
  [cos180, -sin180, (1 - cos180) * center_x + sin180 * center_y],
  [sin180, cos180, (1 - cos180) * center_y - sin180 * center_x]
])
img_ters = np.zeros_like(img_rgb)
for i in range(rows):
  for j in range(cols):
    new_coords = np.dot(rotation_matrix_180, [j, i, 1])
    new_x, new_y = int(new_coords[0]), int(new_coords[1])
    if 0 <= new_x < cols and 0 <= new_y < rows:
      img_ters[new_y, new_x] = img_rgb[i, j]
# Görseli gösterme
plt.figure(figsize=(8, 8))
```

```
plt.subplot(1, 1, 1)

plt.imshow(img_ters)

plt.title('Ters Çevrilmiş Görüntü')

plt.axis('off')

plt.tight_layout()

plt.show()
```

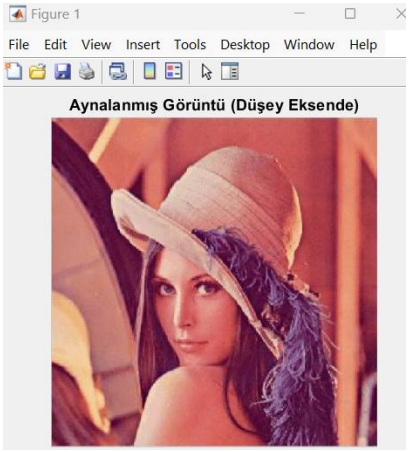
3)Görüntü Aynalama

Aynalama işlemi, ters çevirme işleminin düşey eksende yapılmış halidir. Birinci sutun elemanları ile son sutun elemanları sırası ile yer değitirir.

```
% Orijinal görüntünün yüklenmesi
I = imread('lena.bmp');

% Görüntünün aynalanması (düşey eksende ters çevirme)
I_mirrored = fliplr(I); % Sütunlar ters çevrilir, sağ ve sol yer değiştirir.

% Görüntülerin gösterilmesi
figure;
subplot(1, 2, 1), imshow(I), title('Orijinal Görüntü');
subplot(1, 2, 2), imshow(I_mirrored), title('Aynalanmış Görüntü (Düşey Eksende)');
```



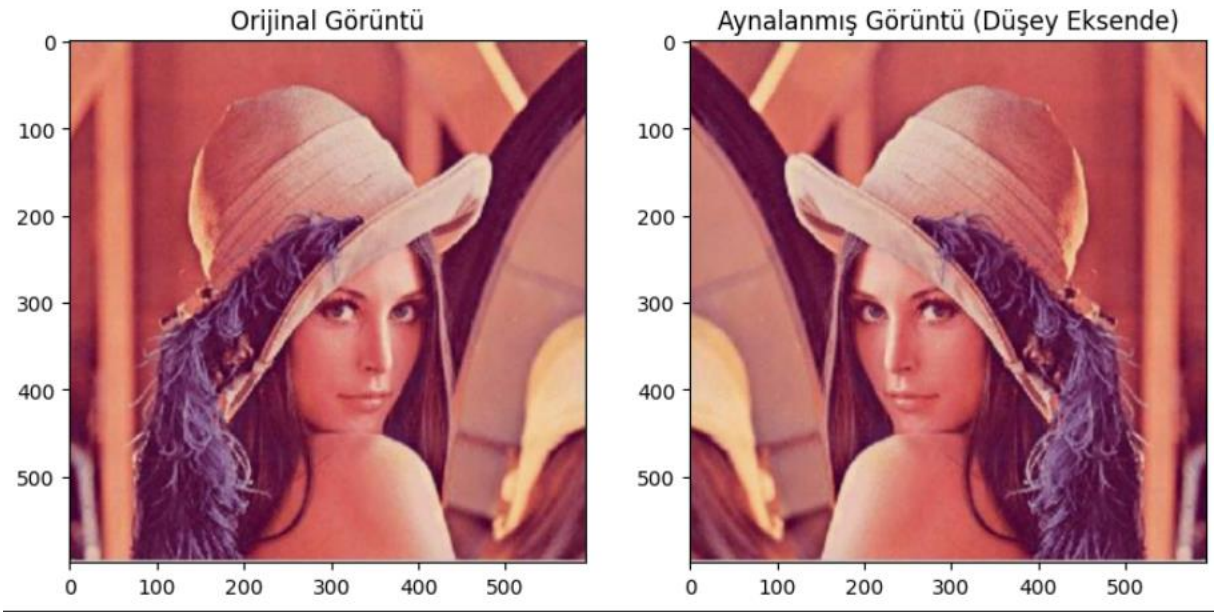
Şekil 16-Orijinal Görüntü



Şekil 16-Aynalanmış Görüntü

Python Kodu:

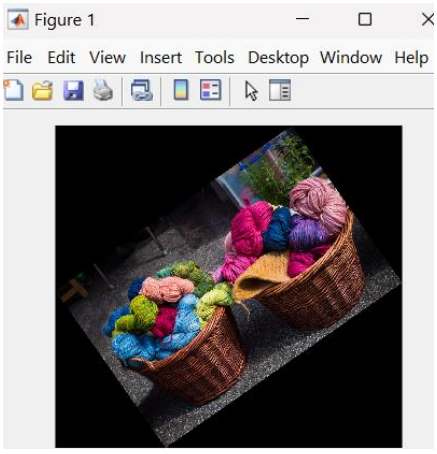
```
 import cv2
import numpy as np
import matplotlib.pyplot as plt
# Görüntüyü okuma
img = cv2.imread('lena.bmp')
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# Görüntüyü aynalama işlemi (yatayda)
rows, cols, channels = img.shape
img_mirror = np.zeros_like(img_rgb)
for i in range(rows):
   for j in range(cols):
      # Yatayda aynalama
      img_mirror[i, cols - j - 1] = img_rgb[i, j]
# Görselleri yan yana gösterme
plt.figure(figsize=(12, 6))
# Orijinal Görüntü
plt.subplot(1, 2, 1)
plt.imshow(img_rgb)
plt.title('Orijinal Görüntü')
plt.axis('off')

# Aynalanmış Görüntü
plt.subplot(1, 2, 2)
plt.imshow(img_mirror)
plt.title('Aynalanmış Görüntü')
plt.axis('off')

plt.tight_layout()
plt.show()
```



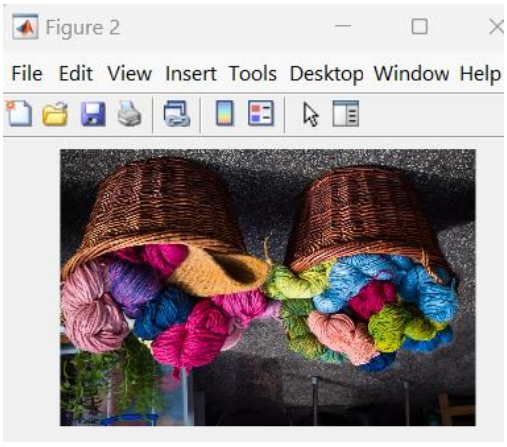
4)Görüntünün Çevrilmesi

Bir görüntünün çevrilmesi verilen açıya göre saat yönünde ya da saat yönünün tersi yönünde görüntü düzleminin döndürülmesidir.

```
% Görüntünün okunması
I1 = imread('ip.jpg');
% Görüntünün verilen açı kadar çevrilmesi
I2 = imrotate(I1,35,'bilinear');
% Yeni görüntünün gösterilmesi
figure, imshow(I2)
% Görüntünün verilen açı kadar çevrilmesi
I3 = imrotate(I1,180,'bicubic');
figure, imshow(I3)
```



Şekil 17-Görüntünün Yan Çevrilmesi



Şekil 17-Görüntünün Ters Çevrilmesi

Python:

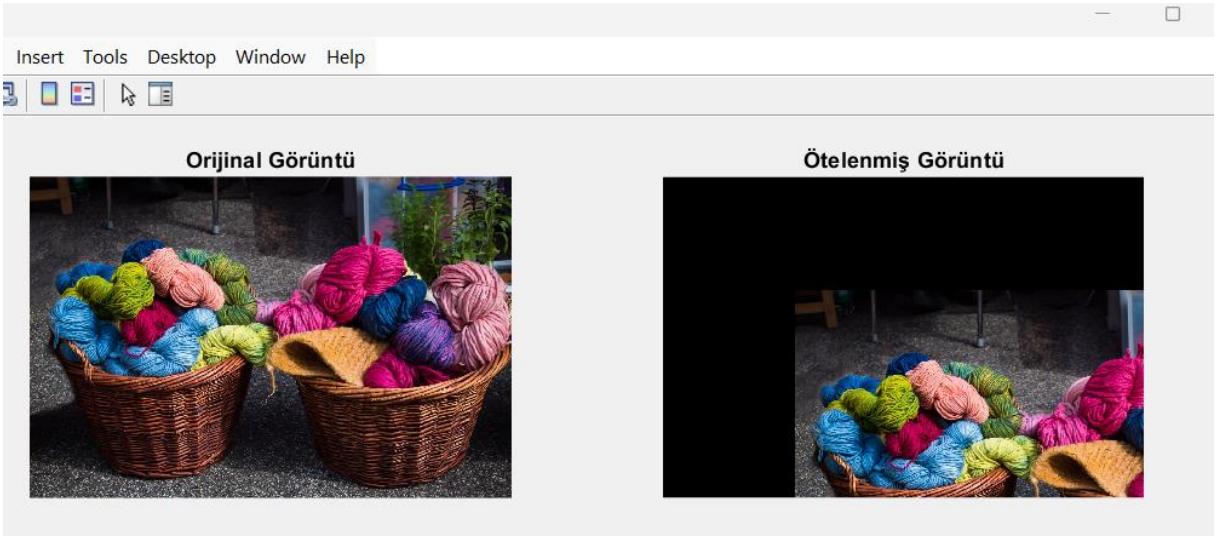
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
I1 = cv2.imread('ip.jpg')  # BGR formatında okunur.


# BGR formatını RGB'ye manuel olarak çevirme
def bgr_to_rgb(image):
    h, w, c = image.shape
    rgb_image = np.zeros_like(image)
    for y in range(h):
        for x in range(w):
            rgb_image[y, x, 0] = image[y, x, 2]  # R
            rgb_image[y, x, 1] = image[y, x, 1]  # G
            rgb_image[y, x, 2] = image[y, x, 0]  # B
    return rgb_image
# Görüntü döndürme işlemi (manuel olarak)
def rotate_image(image, angle):
    h, w, c = image.shape
    center = (w // 2, h // 2)
    theta = np.radians(angle)
    rotated_image = np.zeros_like(image)
    for y in range(h):
        for x in range(w):
            x_prime = int((x - center[0]) * np.cos(theta) - (y - center[1]) * np.sin(theta) + center[0])
            y_prime = int((x - center[0]) * np.sin(theta) + (y - center[1]) * np.cos(theta) + center[1])
            if 0 <= x_prime < w and 0 <= y_prime < h:
                rotated_image[y_prime, x_prime] = image[y, x]
    return rotated_image


I1_rgb = bgr_to_rgb(I1)  # Orijinal BGR görüntüyü RGB'ye çevir
I2 = rotate_image(I1_rgb, 35) # 35 derece döndürülmüş görüntü
I3 = rotate_image(I1_rgb, 180)  # 180 derece döndürülmüş görüntü
plt.figure(figsize=(15, 5))  # Görüntülerin gösterilmesi
```

```
plt.subplot(1, 3, 1)

plt.imshow(I1_rgb)  # RGB olarak gösteriyoruz

plt.title('Orijinal Görüntü')

plt.axis('off')

plt.subplot(1, 3, 2)

plt.imshow(I2)

plt.title('35 Derece Döndürülmüş Görüntü')

plt.axis('off')

plt.subplot(1, 3, 3)

plt.imshow(I3)

plt.title('180 Derece Döndürülmüş Görüntü')

plt.axis('off')

plt.tight_layout()

plt.show()
```

5) Görüntü öteleme

Görüntü öteleme, bir görüntünün belirli bir doğrultuda (yatay veya dikey) yer değiştirilmesi işlemidir. Bu işlem, görüntüdeki piksellerin belirli bir sayı kadar kaydırılmasıyla gerçekleştirilir.

```
% Orijinal görüntünün yüklenmesi
I = imread('ip.jpg'); % Görüntü dosyanızın adı 'lena.bmp' olmalıdır

% Öteleme vektörünün tanımlanması (x ve y yönünde)
dx = 350; % Yatay yönde 50 piksel sağa kaydırma
dy = 300; % Dikey yönde 30 piksel aşağı kaydırma

% Görüntünün ötelenmesi
I_translated = imtranslate(I, [dx, dy]);

% Görüntülerin gösterilmesi
figure;
subplot(1, 2, 1), imshow(I), title('Orijinal Görüntü');
subplot(1, 2, 2), imshow(I_translated), title('Ötelenmiş Görüntü');
```



Şekil 18-Görüntü Öteleme

Python Kodu:

```
from PIL import Image
import matplotlib.pyplot as plt
def otele(goruntu_yolu, x, y):
  try:
    orijinal_grt = Image.open(goruntu_yolu)
    # Görüntüyü ötele
    oteleme_grt = orijinal_grt.transform(
      orijinal_grt.size,
      Image.AFFINE,
      (1, 0, x, 0, 1, y)
    )
    # Görüntüleri göster
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(orijinal_grt)
    plt.title("Orijinal Görüntü")
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.imshow(oteleme_grt)
    plt.title("Öteleme Sonrası Görüntü")
    plt.axis('off')
    plt.show()
```

```
    except Exception as e:
        print(f"Hata: {e}")


# Kullanıcıdan giriş al
print("Not: Pozitif X değeri sağa, pozitif Y değeri aşağı öteleme yapar.")
goruntu_yolu = "ip.jpg"
x_oteleme = float(input("X ekseninde öteleme miktarını girin (sağa kaydırma için pozitif değer girin): "))
y_oteleme = float(input("Y ekseninde öteleme miktarını girin (aşağı kaydırma için pozitif değer girin): "))
otele(goruntu_yolu, x_oteleme, y_oteleme)
```

15)Görüntünün Tekrar Boyutlandırılması

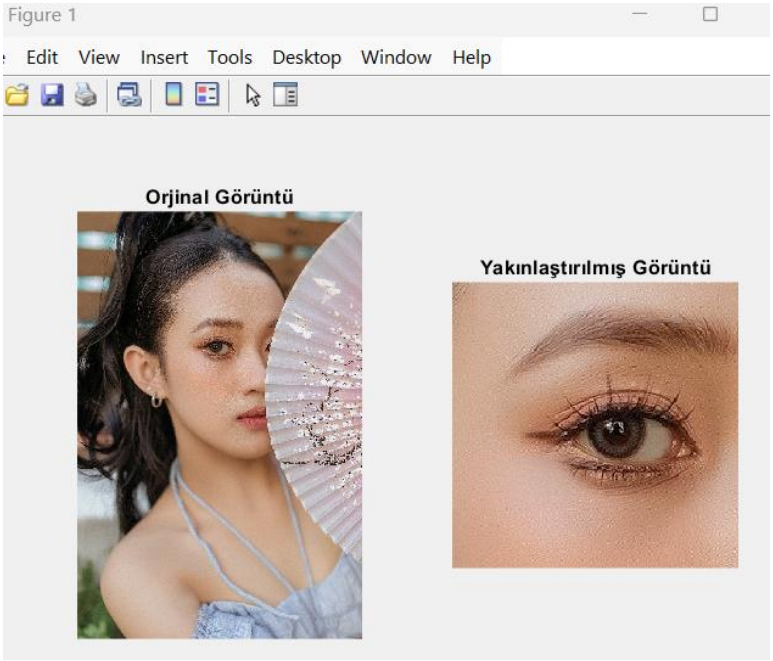
MATLAB'da görüntü boyutlandırması için **imresize** fonksiyonu kullanılır. Bu fonksiyon ile görüntü istenilen ölçek faktörü veya boyutlarla yeniden şekillendirilebilir.

1) Yakınlaştırma

Bir görüntüyü daha büyük hale getirme işlemidir. Bu işlem, görüntünün piksel boyutlarını artırarak, görüntünün belirli bir kısmının daha ayrıntılı bir şekilde görülmesini sağlar. MATLAB'da yakınlaştırma işlemi, **imresize** fonksiyonu kullanılarak yapılır. Bu fonksiyon, bir görüntüyü belirtilen ölçek faktörü kadar büyütür ve büyütülen görüntüyü oluştururken, yeni piksel değerlerini interpolasyon yöntemleriyle hesaplar.

```
% Orijinal görüntünün yüklenmesi
orjinalresim = imread('kadin.jpg');

% Seçilen bölgenin koordinatları: [sol üst x, sol üst y, genişlik, yükseklik]
secilibolge = [300, 300, 200, 200];

% Seçilen bölgenin kesilmesi
yakinlastirilmisbolge = imcrop(orjinalresim, secilibolge);

% Kesilen bölgenin boyutunun büyütülmesi (yaklaşık %200 büyütme)
yakinlastirilmisbolge_buyutulmus = imresize(yakinlastirilmisbolge, 2);
```

```
% Görüntülerin gösterilmesi
figure;
subplot(1, 2, 1);
imshow(orjinalresim);
title('Orjinal Görüntü');

subplot(1, 2, 2);
imshow(yakinlastirilmisbolge_buyutulmus);
title('Yakınlaştırılmış Görüntü');
% Orijinal görüntünün yüklenmesi
orjinalresim = imread('kadin.jpg');

% Seçilen bölgenin koordinatları: [sol üst x, sol üst y, genişlik, yükseklik]
secilibolge = [300, 300, 200, 200];

% Seçilen bölgenin kesilmesi
yakinlastirilmisbolge = imcrop(orjinalresim, secilibolge);

% Kesilen bölgenin boyutunun büyütülmesi (yaklaşık %200 büyütme)
yakinlastirilmisbolge_buyutulmus = imresize(yakinlastirilmisbolge, 2);

% Görüntülerin gösterilmesi
figure;
subplot(1, 2, 1);
imshow(orjinalresim);
title('Orjinal Görüntü');

subplot(1, 2, 2);
imshow(yakinlastirilmisbolge_buyutulmus);
title('Yakınlaştırılmış Görüntü');
```

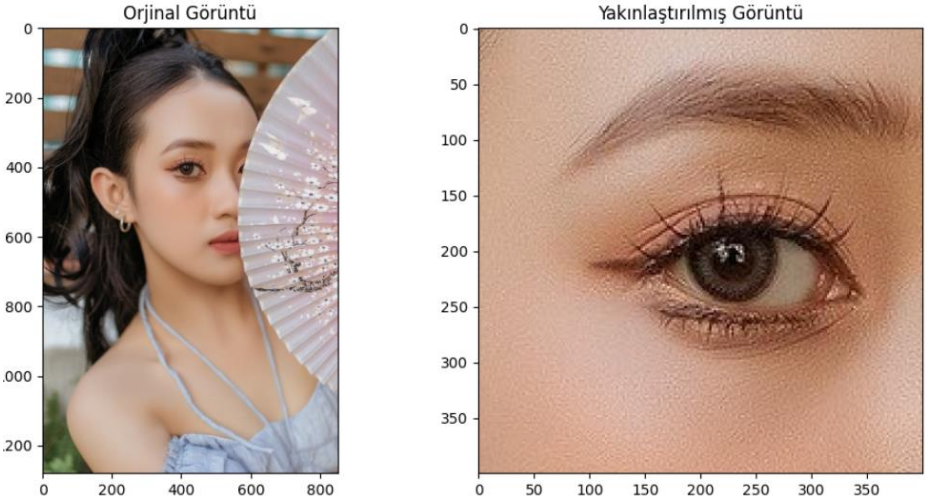


Şekil 19-Görüntü Yakınlaştırılması

Python Kodu:

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

orjinalresim = cv2.imread('kadin.jpg')

orjinalresim_rgb = cv2.cvtColor(orjinalresim, cv2.COLOR_BGR2RGB)  # BGR'den RGB'ye dönüştür

# Seçilen bölgenin koordinatları: [sol üst x, sol üst y, genişlik, yükseklik]

secilibolge = (300, 300, 200, 200)  # (x, y, genişlik, yükseklik)

# Seçilen bölgenin kesilmesi

yakinlastirilmisbolge = orjinalresim_rgb[secilibolge[1]:secilibolge[1]+secilibolge[3],
secilibolge[0]:secilibolge[0]+secilibolge[2]]

# Yakınlaştırma işlemi için manuel çözüm (2x büyütme)

def zoom_in(image, scale=2):

  height, width, channels = image.shape

  new_height = height * scale

  new_width = width * scale

  # Yeni görüntü için boş bir liste oluşturuyoruz

  zoomed_image = []

  for i in range(new_height):

    # Yeni satır için boş bir liste oluşturuluyor

    row = []

    for j in range(new_width):

      # Orijinal görüntüdeki karşılık gelen pikseli bulmak için

      orig_i = int(i / scale)

      orig_j = int(j / scale)

      # Orijinal görüntüdeki piksel değerini alıyoruz

      pixel = image[orig_i, orig_j]

      # Satır listesine pikseli ekliyoruz

      row.append(pixel)

    # Yeni satırı zoomed_image listesine ekliyoruz

    zoomed_image.append(row)

  return zoomed_image

# Kesilen bölgeyi 2x büyütme

yakinlastirilmisbolge_buyutulmus = zoom_in(yakinlastirilmisbolge, scale=2)

# Görüntülerin gösterilmesi
```

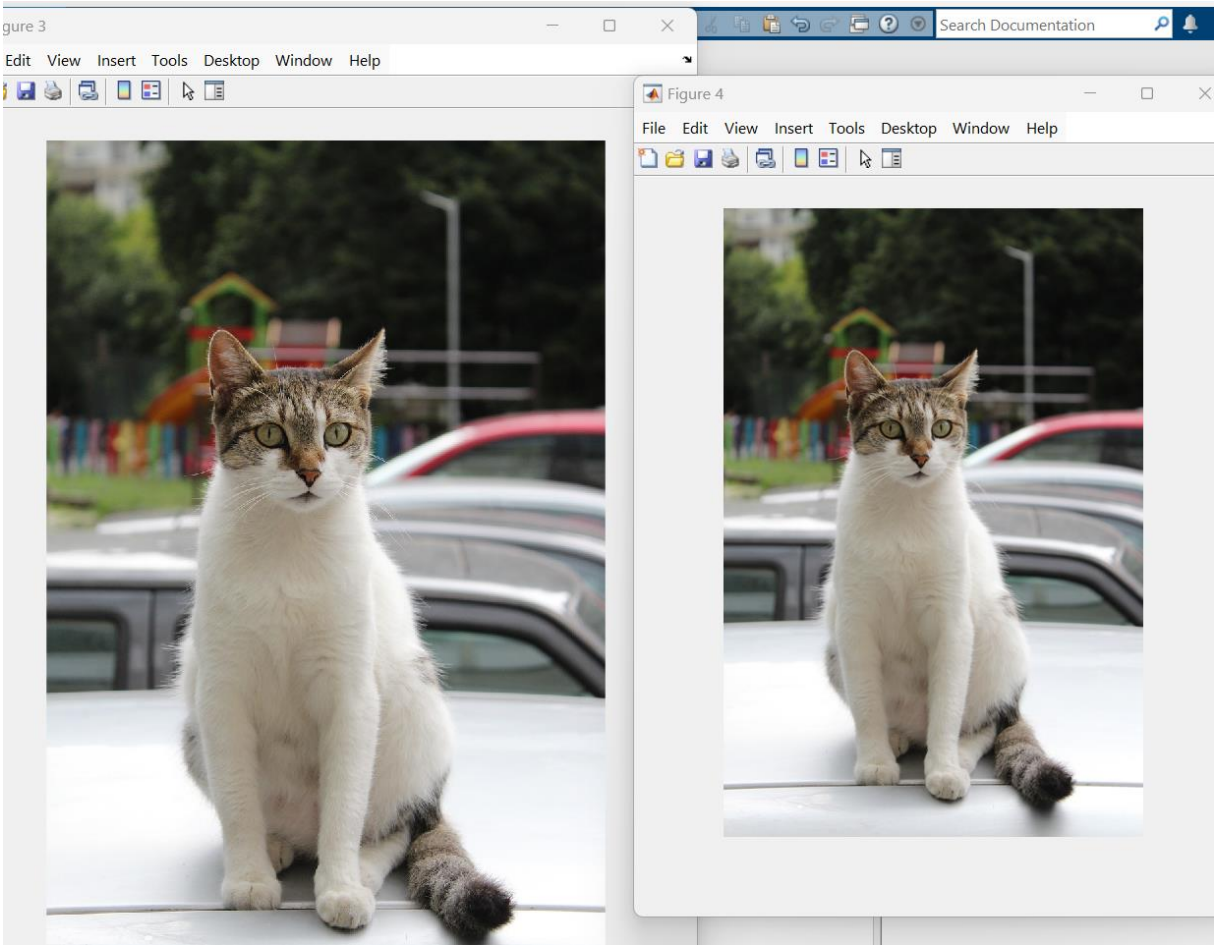
```
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(orjinalresim_rgb)
plt.title('Orjinal Görüntü')

plt.subplot(1, 2, 2)
plt.imshow(yakinlastirilmisbolge_buyutulmus)
plt.title('Yakınlaştırılmış Görüntü')

plt.tight_layout()
plt.show()
```



2)Uzaklaştırma

Uzaklaştırma, bir görüntünün boyutlarını küçültme işlemidir. Bu işlem, genellikle bir görüntüyü belirli bir oranda küçültmek ve böylece daha düşük çözünürlükte bir versiyonunu elde etmek için kullanılır. Uzaklaştırma işlemi, görüntüdeki piksel sayısını azaltır ve daha küçük bir boyutta daha az detay içerir. MATLAB'da bu işlem **imresize** fonksiyonu ile yapılır.

```
% Görüntünün okunması

I1 = imread('cat.jpg');


% Görüntünün uzaklaştırılması

I2 = imresize(I1, 0.5); % Görüntüyü yarıya küçült


% İlk görüntünün gösterilmesi

figure1 = figure; imshow(I1);

truesize(figure1); % İlk figure penceresinin boyutunu sabit tut


% Uzaklaştırılmış görüntünün gösterilmesi

figure2 = figure; imshow(I2);

truesize(figure2); % İkinci figure penceresinin boyutunu sabit tut
```



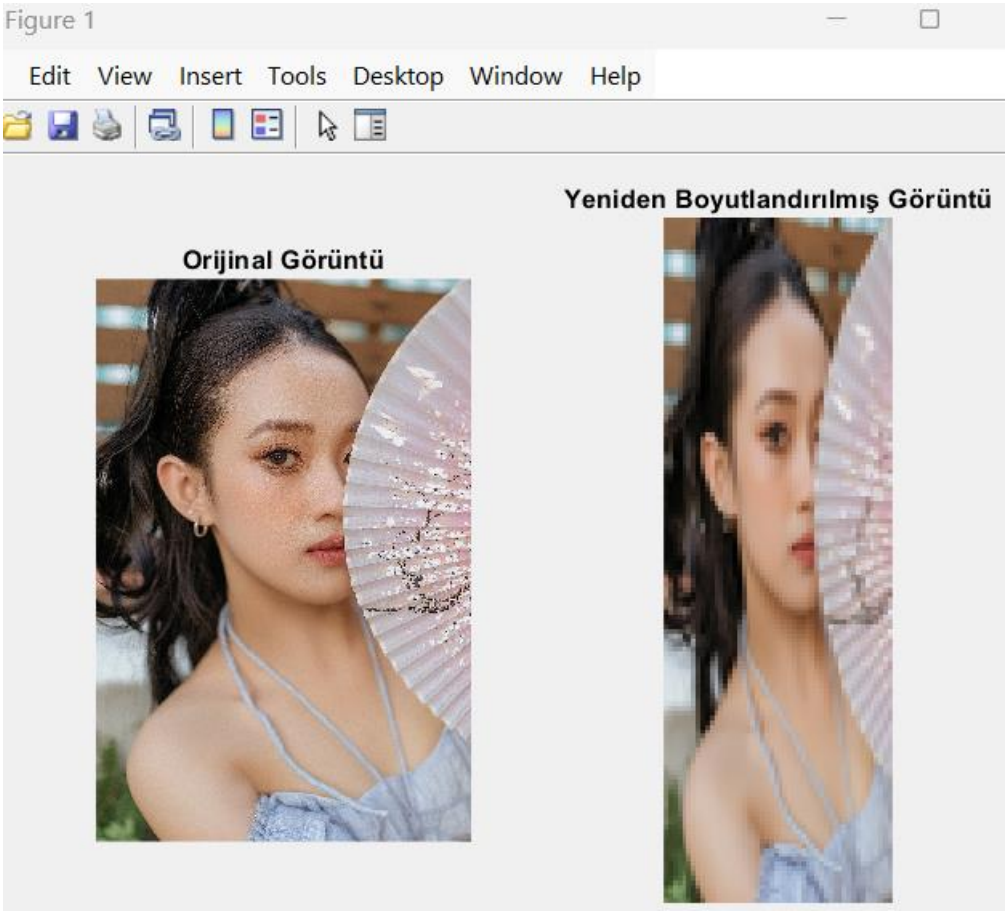
Python Kodu:

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

from PIL import Image


def uzaklastir_ve_goster(goruntu_yolu, oran):

  try:

    # Görüntüyü okuma

    orijinal_goruntu = Image.open(goruntu_yolu)


    # Görüntü boyutlarını alma

    genislik, yukseklik = orijinal_goruntu.size


    # Yeni boyutları hesaplama
```

```
        yeni_genislik = int(genislik / oran)
        yeni_yukseklik = int(yukseklik / oran)

        # Görüntüyü yeniden boyutlandırma
        uzaklastirilmis_goruntu = orijinal_goruntu.resize((yeni_genislik, yeni_yukseklik))

        # Görselleri yan yana gösterme
        plt.figure(figsize=(12, 6))

        # Orijinal Görüntü
        plt.subplot(1, 2, 1)
        plt.imshow(orijinal_goruntu)
        plt.title('Orijinal Görüntü')
        plt.axis('off')

        # Uzaklaştırılmış Görüntü
        plt.subplot(1, 2, 2)
        plt.imshow(uzaklastirilmis_goruntu)
        plt.title('Uzaklaştırılmış Görüntü')
        plt.axis('off')

        plt.tight_layout()
        plt.show()

    except Exception as e:
        print(f"Hata: {e}")
# Kullanıcıdan giriş alma
goruntu_yolu = 'cat.jpg'  # Görüntü dosyasının yolu
uzaklastirma_orani = float(input("Uzaklaştırma oranını girin (örneğin, 1.5): "))
uzaklastir_ve_goster(goruntu_yolu, uzaklastirma_orani)
```

Bir görüntünün tekrar boyutlandırılması verilen boyutlara göre resmin oluşturulmasıdır.

```
% Görüntünün okunması
I1 = imread('kadin.jpg');

% Görüntünün yeniden boyutlandırılması
I2 = imresize(I1, [150 50]);

% Görüntülerin yan yana gösterilmesi
figure;
subplot(1, 2, 1);
imshow(I1);
title('Orijinal Görüntü');

subplot(1, 2, 2);
imshow(I2);
title('Yeniden Boyutlandırılmış Görüntü');
```

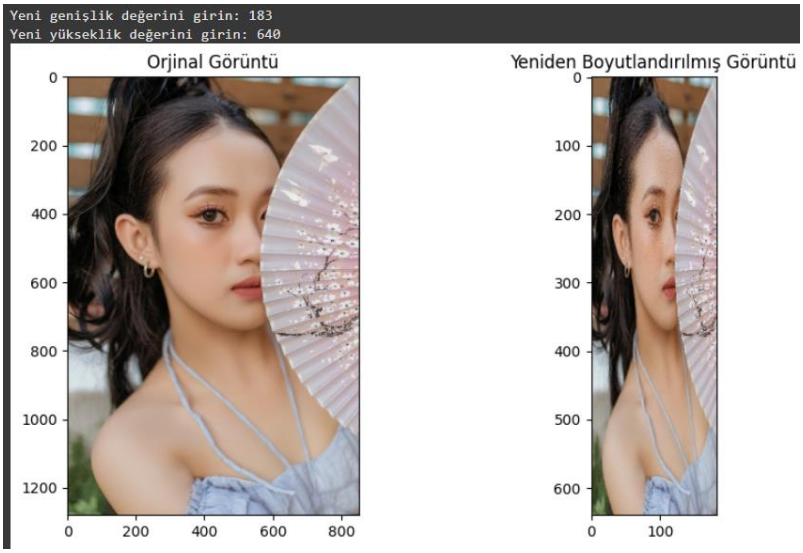


Şekil 21-Görüntünün tekrar boyutlandırılması

Python Kodu:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

# Renkli resmi yükle
img = cv2.imread('kadin.jpg')
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # OpenCV, resmi BGR formatında okur, RGB'ye dönüştürüyoruz

# 1. Gri tonlamaya dönüştürme (RGB'den gri tonlama)
def rgb_to_gray(img):
    # RGB'den gri tonlamaya dönüştürme (0.2989 * R + 0.5870 * G + 0.1140 * B)
    gray_img = np.dot(img[...,:3], [0.2989, 0.5870, 0.1140])
    return gray_img.astype(np.uint8)

gray_image = rgb_to_gray(img)

# 4. Yeni boyutta görüntü (belirli boyutlara yeniden boyutlandırma)
def resize_to_new_size(image, new_width, new_height):
    resized_image3 = np.zeros((new_height, new_width, image.shape[2]), dtype=np.uint8)

    scale_x = new_width / image.shape[1]
    scale_y = new_height / image.shape[0]

    for i in range(new_height):
        for j in range(new_width):
            orig_i = int(i / scale_y)
            orig_j = int(j / scale_x)
            resized_image3[i, j] = image[orig_i, orig_j]

    return resized_image3

# Yeni boyutlara yeniden boyutlandırma için kullanıcıdan değer alma
new_width = int(input("Yeni genişlik değerini girin: "))
```

```
new_height = int(input("Yeni yükseklik değerini girin: "))


resized_image3 = resize_to_new_size(img_rgb, new_width, new_height)
# 5. Sonuçları görselleştirme
plt.figure(figsize=(10, 5))


plt.subplot(1, 2, 1)
plt.imshow(img_rgb)
plt.title('Orjinal Görüntü')
plt.subplot(1, 2, 2)
plt.imshow(resized_image3)
plt.title('Yeniden Boyutlandırılmış Görüntü')


plt.tight_layout()
plt.show()
```



16) Morfolojik işlemler

Morfolojik işlemler, görüntü işleme alanında, özellikle ikili (binary) görüntüler üzerinde yapılan temel analiz ve işlem yöntemleridir. Bu işlemler, genellikle şekil, yapı ve nesnelerin büyüklüklerini değiştirmek veya nesneleri birleştirmek veya ayırmak gibi amaçlarla kullanılır. Morfolojik işlemler, görüntülerin yapısal özelliklerini analiz etmek için çok faydalıdır.

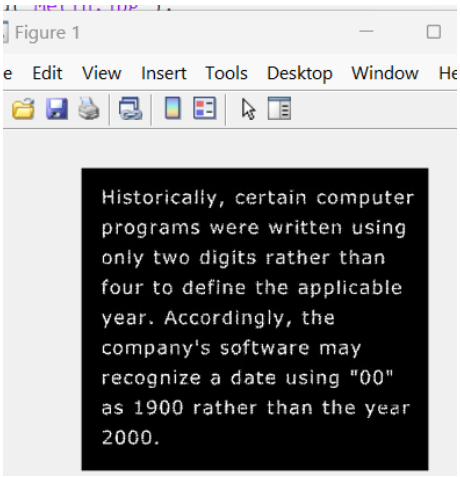
Morfolojik imge işlemede temel olarak kullanılan iki işlem vardır:

- Yayma (dilation)
- Aşındırma (erosion)

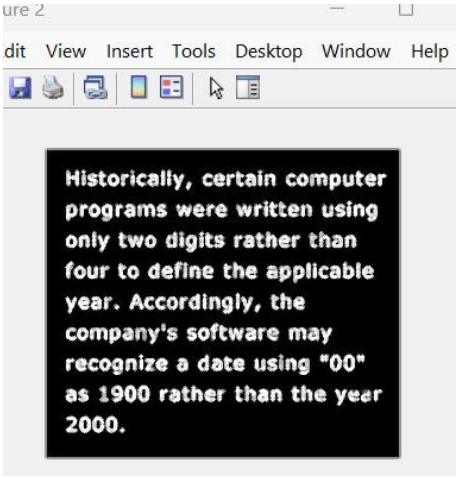
1)Yayma

Yayma (Dilation), morfolojik bir görüntü işleme işlemidir ve genellikle ikili (binary) görüntülerde kullanılır. Bu işlem, görüntüdeki beyaz (ön plan) piksellerin çevresindeki komşuları genişleterek, nesnelerin boyutlarını büyütür veya birbirine yakın nesneleri birleştirir. Yayma, bir yapısal elemanın (örneğin, disk, kare veya çapraz şekli) görüntüdeki her pikselin etrafında kaydırılmasıyla yapılır.

```
% Görüntünün okunması

I1=imread('Metin.jpg');
% Kernelin hazırlanması
h=[0 1 0;1 1 1;0 1 0];
% Dialation işleminin uygulanması
I2=imdilate(I1,h);
% Yeni görüntülerin gösterilmesi
imshow(I1) ;figure, imshow(I2)
```



Şekil 27-Orijinal Görüntü

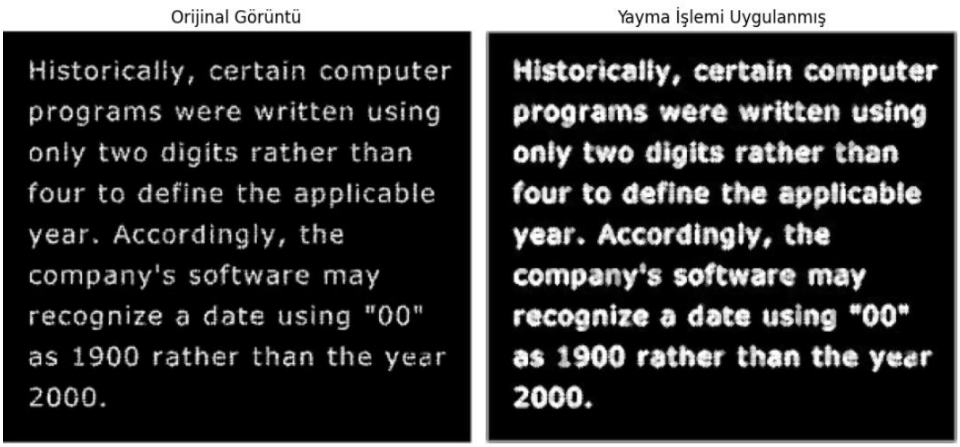


Şekil 27-Yayma İşlemi Uyg. Görüntü

Python Kodu:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
# Resmin okunması (gri tonlamalı)
img = cv2.imread('Metin.jpg', 0)

# Kernel'in hazırlanması
kernel = np.array([[0, 1, 0],
                   [1, 1, 1],
                   [0, 1, 0]], dtype=np.uint8)

# Görüntü boyutları
height, width = img.shape

# Dilatasyon işlemini gerçekleştirecek fonksiyon
def dilate_image(image, kernel):
  kernel_height, kernel_width = kernel.shape
  pad_h = kernel_height // 2
  pad_w = kernel_width // 2

  # Görüntüye padding ekleyelim
  padded_image = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)), mode='constant', constant_values=0)

  dilated_image = np.zeros_like(image)

  # Görüntü üzerinde kaydırma yaparak dilatasyon işlemi
  for i in range(height):
    for j in range(width):
      # Kernel'ı görüntü üzerine kaydırıyoruz
      region = padded_image[i:i+kernel_height, j:j+kernel_width]
      dilated_image[i, j] = np.max(region * kernel)  # Kernel ile örtüşen piksellerin maksimum değeri alınır
```

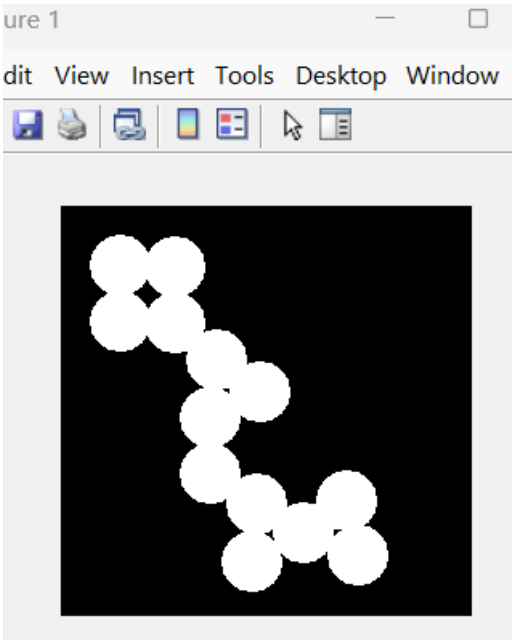
```
    return dilated_image

# Dilatasyon işlemini uygulayalım
dilated_img = dilate_image(img, kernel)

# Görüntüleri yan yana koyup başlık ekleme
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Orijinal Görüntü')
plt.axis('off')  # Eksenleri kaldırıyoruz
plt.subplot(1, 2, 2)
plt.imshow(dilated_img, cmap='gray')
plt.title('Yayma İşlemi Uygulanmış')
plt.axis('off')  # Eksenleri kaldırıyoruz

# Görselleştirme
plt.tight_layout()
plt.show()
```



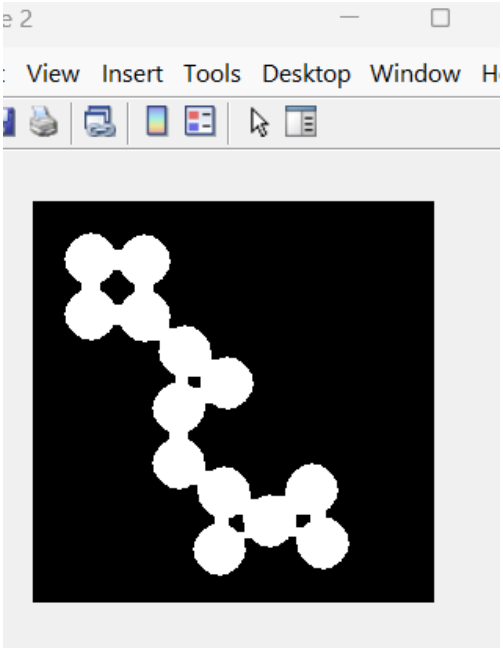
2) Morfolojik İmge İşleme – Aşındırma(erosion)

Aşındırma (erosion), **morfolojik görüntü işleme** yöntemlerinden biridir ve genellikle ikili (binary) veya gri tonlamalı görüntüler üzerinde uygulanır. Aşındırma işlemi, bir görüntünün belirli bir yapısal eleman (**structuring element**) ile analiz edilerek nesnelerin sınırlarının küçültülmesini sağlar.

```
% Görüntünün okunması
I1=imread('circles.png');
% Kernelin hazırlanması
h=ones(5,5);
% Erosion işleminin uygulanması
I2=imerode(I1,h);
% Yeni görüntülerin gösterilmesi
figure;imshow(I1)
figure; imshow(I2)
```



Şekil 28-Orijinal Görüntü

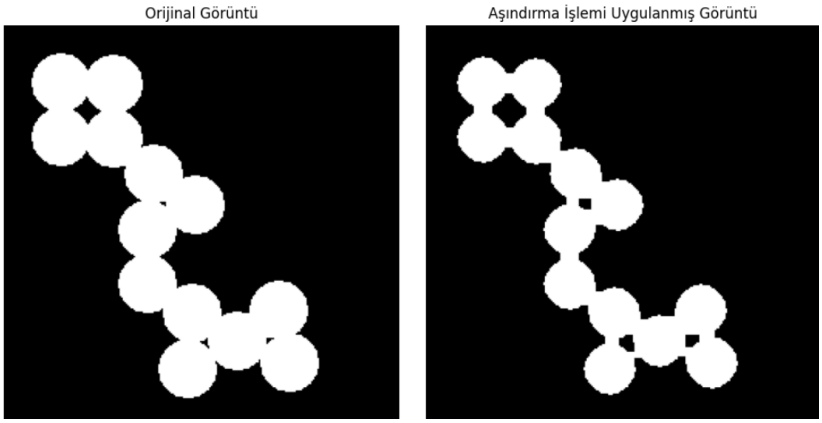


Şekil 28- Aşındırma işlemi uygulanmış görüntü

Python Kodu:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('circles.png', 0)  # Gri tonlamalı olarak okuma
kernel = np.ones((5, 5), np.uint8)  # Kernel'in hazırlanması (5x5 boyutunda bir kare kernel)
height, width = img.shape   # Görüntü boyutları

# Erozyon işlemini gerçekleştirecek fonksiyon
def erode_image(image, kernel):
  kernel_height, kernel_width = kernel.shape
  pad_h = kernel_height // 2
  pad_w = kernel_width // 2

  # Görüntüye padding ekleyelim
  padded_image = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)), mode='constant', constant_values=255)
  eroded_image = np.zeros_like(image)

  # Görüntü üzerinde kaydırma yaparak erozyon işlemi
  for i in range(height):
    for j in range(width):
      # Kernel'ı görüntü üzerine kaydırıyoruz
      region = padded_image[i:i+kernel_height, j:j+kernel_width]
      # Kernel'in örtüştüğü bölgedeki minimum değeri alıyoruz
      eroded_image[i, j] = np.min(region * kernel)

  return eroded_image

eroded_img = erode_image(img, kernel)   # Erozyon işlemini uygulayalım

plt.figure(figsize=(10, 5))    # Görüntüleri gösterme

plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
```

```
plt.title('Orijinal Görüntü')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(eroded_img, cmap='gray')
plt.title('Aşındırma İşlemi Uygulanmış Görüntü')
plt.axis('off')

plt.tight_layout()
plt.show()
```



Morfolojik İmge İşleme – Açma(opening) ve Kapama(closing)

Açma ve kapama, yayma ve aşındırma işlemlerinin iki değerli imgeye ardışıl uygulanmasıyla yapılan işlemlerdir.

Open(Açma)

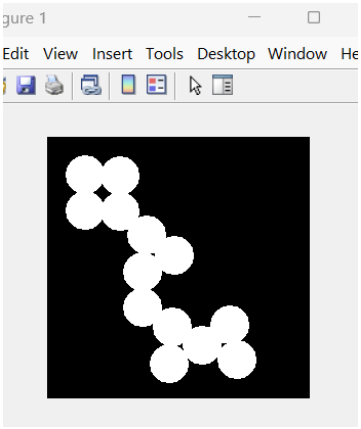
Açma (Opening), morfolojik görüntü işlemede kullanılan bir yöntemdir ve genellikle gürültüyü temizlemek veya bir görüntüdeki nesneleri ayrıştırmak için kullanılır. **Açma işlemi**, **aşındırma (erosion)** işleminin hemen ardından **yayma (dilation)** işleminin uygulanmasıyla gerçekleştirilir.

```
% Görüntünün okunması
I1 = imread('circles.png');
% Kernelin hazırlanması
se = strel('disk',5);
% Açma işleminin uygulanması
I2 = imopen(I1,se);
```

% Yeni görüntülerin gösterilmesi

figure;imshow(I1)

figure; imshow(I2)



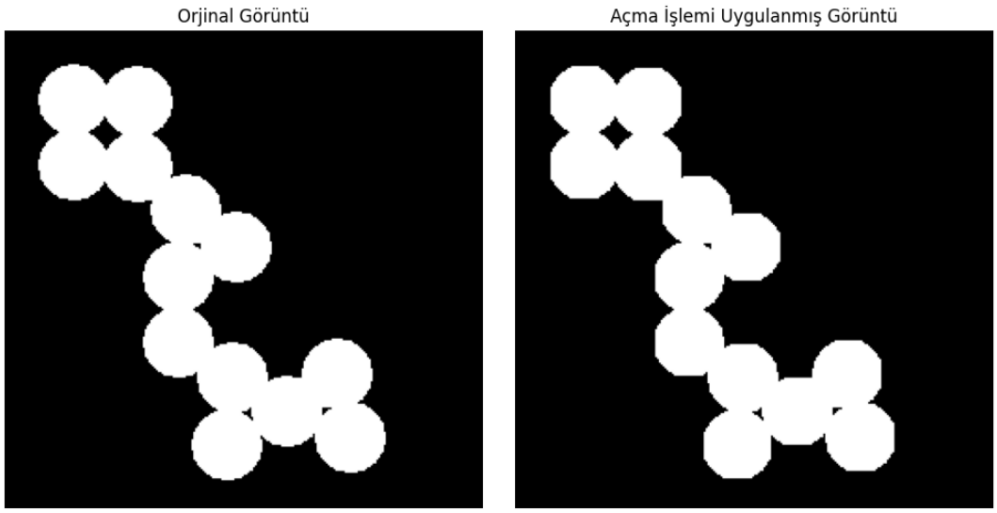
Şekil 29-Açma uygulanmış grt

Python Kodu:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
image = cv2.imread('circles.png')
# Erozyon fonksiyonu
def apply_erosion(img, kernel):
  eroded_img = np.zeros_like(img)
  kernel_height, kernel_width = kernel.shape
  img_height, img_width, _ = img.shape
  # Çekirdek boyutunda pencereleri kaydırarak işlemi yapıyoruz
  for i in range(kernel_height//2, img_height - kernel_height//2):
    for j in range(kernel_width//2, img_width - kernel_width//2):
      region = img[i - kernel_height//2:i + kernel_height//2 + 1, j - kernel_width//2:j + kernel_width//2 + 1]
      for c in range(3):
        if np.all(region[:, :, c] == 255):
          eroded_img[i, j, c] = 255
        else:
          eroded_img[i, j, c] = 0
  return eroded_img
```

```
# Dilasyon fonksiyonu
def apply_dilation(img, kernel):
    dilated_img = np.zeros_like(img)
    kernel_height, kernel_width = kernel.shape
    img_height, img_width, _ = img.shape

    for i in range(kernel_height//2, img_height - kernel_height//2):
        for j in range(kernel_width//2, img_width - kernel_width//2):
            region = img[i - kernel_height//2:i + kernel_height//2 + 1, j - kernel_width//2:j + kernel_width//2 + 1]
            for c in range(3):
                if np.any(region[:, :, c] == 255):
                    dilated_img[i, j, c] = 255
                else:
                    dilated_img[i, j, c] = 0
    return dilated_img

# Açma (Opening) işlemi
def apply_opening(img, kernel):
    eroded = apply_erosion(img, kernel)
    opened_img = apply_dilation(eroded, kernel)
    return opened_img

# 9x9 boyutunda bir çekirdek
kernel = np.ones((9, 9), np.uint8)

# Açma işlemi uygula
opened_image_result = apply_opening(image, kernel)
plt.figure(figsize=(10, 5))    # Sonuçları görselleştirme

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))  # Görüntüyü RGB formatına dönüştürerek gösteriyoruz
plt.title('Orjinal Görüntü')
plt.axis('off')
plt.subplot(1, 2, 2)
```

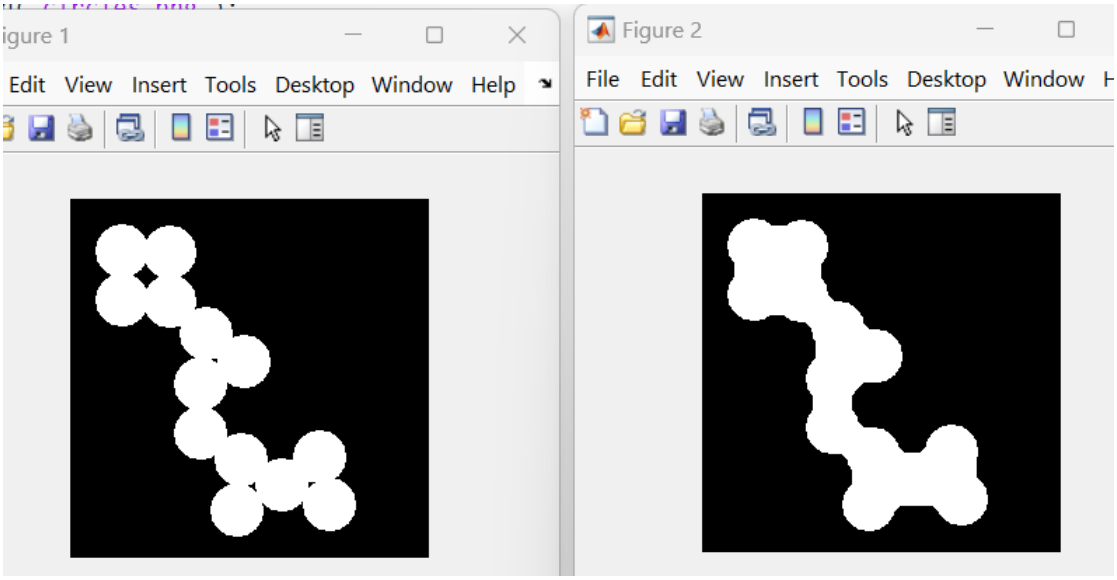
```
plt.imshow(cv2.cvtColor(opened_image_result, cv2.COLOR_BGR2RGB))  # Görüntüyü RGB formatına dönüştürerek gösteriyoruz

plt.title('Açma İşlemi Uygulanmış Görüntü')

plt.axis('off')

plt.tight_layout()

plt.show()
```



Close(Kapama)

Kapama, yayma ve aşındırma işlemlerinin görüntüye ardışıl uygulanmasıyla yapılan işlemlerdir.

```
% Görüntünün okunması

I1 = imread(''circles.png'');

% Kernelin hazırlanması

se = strel('disk',10);

% Kapama işleminin uygulanması

I2 = imclose(I1,se);

% Yeni görüntülerin gösterilmesi

figure;imshow(I1) figure; imshow(I2)
```



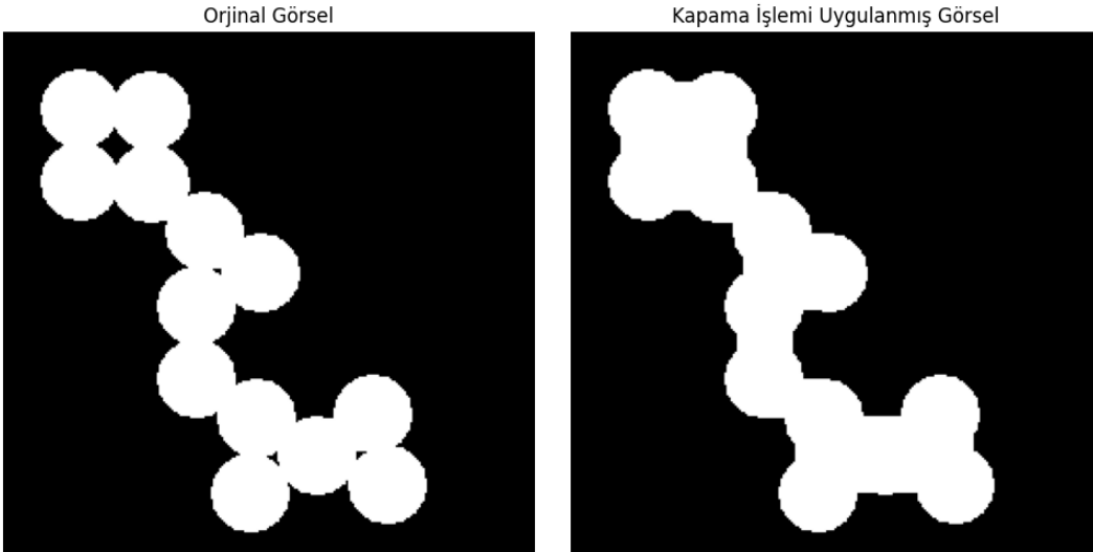
Şekil 30-Orijinal görüntü ve açma İşlemi Uygulanmış Görüntü

Python Kodu:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
img = cv2.imread('circles.png')
# Dilasyon fonksiyonu
def dilation(image, kernel):
  dilated_image = np.zeros_like(image)
  kernel_height, kernel_width = kernel.shape
  image_height, image_width, _ = image.shape
  # Çekirdek boyutunda pencereleri kaydırarak işlemi yapıyoruz
  for i in range(kernel_height//2, image_height - kernel_height//2):
    for j in range(kernel_width//2, image_width - kernel_width//2):
      # Çekirdek boyutunda bir bölgeyi inceliyoruz
      region = image[i - kernel_height//2:i + kernel_height//2 + 1, j - kernel_width//2:j + kernel_width//2 + 1]
      # Her renk kanalını kontrol ediyoruz (RGB)
      for c in range(3):
        # Eğer bölgedeki herhangi bir renk kanalı 255 (beyaz) ise, merkez pikseli beyaz yapıyoruz
        if np.any(region[:, :, c] == 255):
          dilated_image[i, j, c] = 255
        else:
          dilated_image[i, j, c] = 0
  return dilated_image
```

```
# Erozyon fonksiyonu
def erosion(image, kernel):
    eroded_image = np.zeros_like(image)
    kernel_height, kernel_width = kernel.shape
    image_height, image_width, _ = image.shape

    # Çekirdek boyutunda pencereleri kaydırarak işlemi yapıyoruz
    for i in range(kernel_height//2, image_height - kernel_height//2):
        for j in range(kernel_width//2, image_width - kernel_width//2):
            # Çekirdek boyutunda bir bölgeyi inceliyoruz
            region = image[i - kernel_height//2:i + kernel_height//2 + 1, j - kernel_width//2:j + kernel_width//2 + 1]
            # Her renk kanalını kontrol ediyoruz (RGB)
            for c in range(3):
                # Eğer bölgedeki tüm renk kanalları 255 (beyaz) ise, merkez pikseli beyaz yapıyoruz
                if np.all(region[:, :, c] == 255):
                    eroded_image[i, j, c] = 255
                else:
                    eroded_image[i, j, c] = 0
    return eroded_image

# Kapama (Closing) işlemi: Dilasyon ve erozyonun sırasıyla uygulanması
def closing(image, kernel):
    # Önce dilasyon uygula
    dilated_image = dilation(image, kernel)
    # Sonra erozyon uygula
    closed_image = erosion(dilated_image, kernel)
    return closed_image

kernel = np.ones((9, 9), np.uint8)     # 3x3 boyutunda bir çekirdek
closed_image = closing(img, kernel)   # Kapama işlemi uygulama
plt.figure(figsize=(10, 5))     # Sonuçları görselleştirme
```

```
plt.subplot(1, 2, 1)

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))  # Görüntüyü RGB formatına dönüştürerek gösteriyoruz

plt.title('Orjinal Görsel')

plt.axis('off')


plt.subplot(1, 2, 2)

plt.imshow(cv2.cvtColor(closed_image, cv2.COLOR_BGR2RGB))  # Görüntüyü RGB formatına dönüştürerek gösteriyoruz

plt.title('Kapama İşlemi Uygulanmış Görsel')

plt.axis('off')

plt.tight_layout()

plt.show()
```



16) Görüntü Netleştirme Algoritmaları

1)Kenar Görüntüsü Kullanarak Resmi Netleştirme

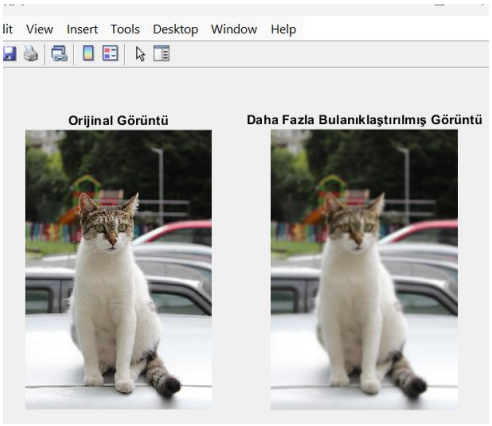
"Kenar Görüntüsü Kullanarak Resmi Netleştirme" (Edge-Based Image Sharpening), bir görüntüyü netleştirmek amacıyla kenar bilgilerini kullanarak görselin daha belirgin hale getirilmesi işlemidir. Bu işlemde, görüntüdeki kenarları vurgulamak için kenar algılama algoritmaları (örneğin, Sobel, Canny veya Prewitt) kullanılır. Kenarlar tespit edildikten sonra, bu kenar bilgisi, görüntüye eklenerek detaylar ve keskinlik artırılır.

Matlab'da genellikle şu adımlar izlenir:

1. **Kenar Algılama**: Görüntüdeki kenarlar tespit edilir. Bu genellikle bir kenar algılama filtresi (örneğin, Sobel filtresi) ile yapılır.
2. **Görüntü Netleştirme**: Tespit edilen kenar bilgileri, görüntüye eklenir. Bu, görüntüdeki düşük frekanslı (bulanık) bileşenleri yüksek frekanslı (keskin) bileşenlerle harmanlamak yoluyla yapılır.

Sonuç olarak, bu işlemle, özellikle görüntüdeki detayların daha net görülmesi sağlanır.

```
% Görüntüyü yükle
img = imread('cat.jpg'); % Görüntü dosyasını burada değiştirin

% 9x9 ortalama (mean) filtre matrisini oluştur
mean_filter = ones(9, 9) / 81; % 9x9'luk matris ve toplamı 1 olacak şekilde
normalize edilmiştir

% Görüntüyü bir kez bulanıklaştır
blurred_img = imfilter(img, mean_filter, 'replicate');

% Aynı filtreyi bir kez daha uygula
blurred_img = imfilter(blurred_img, mean_filter, 'replicate');

% Sonuçları göster
subplot(1,2,1), imshow(img), title('Orijinal Görüntü');
subplot(1,2,2), imshow(blurred_img), title('Daha Fazla Bulanıklaştırılmış
Görüntü');
```

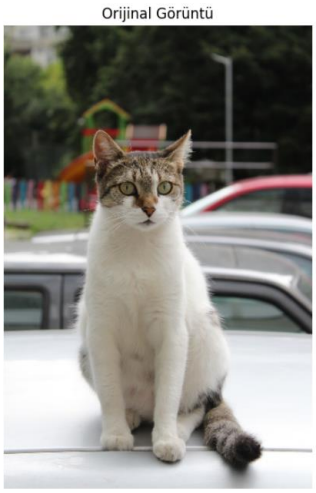


Şekil 31-İki defa mean 9x9 matris ile bulanıklaştırma işlemi uygulanmış görüntü

Python kodu:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('cat.jpg')  # Görüntü dosyasını burada değiştirin

# Görüntü boyutlarını küçültme
scale_percent = 50  # Boyutları yüzde 50 oranında küçült
new_width = int(img.shape[1] * scale_percent / 100)
new_height = int(img.shape[0] * scale_percent / 100)
resized_img = np.zeros((new_height, new_width, img.shape[2]), dtype=np.uint8)

for i in range(new_height):
   for j in range(new_width):
      orig_x = int(i / (new_height / img.shape[0]))
      orig_y = int(j / (new_width / img.shape[1]))
      resized_img[i, j] = img[orig_x, orig_y]

# Görüntüyü BGR'den RGB'ye dönüştürme (manuelleştirilmiş)
rgb_img = np.zeros_like(resized_img)
rgb_img[..., 0] = resized_img[..., 2]  # B'yi R'ye taşı
rgb_img[..., 1] = resized_img[..., 1]  # G aynı kalır
rgb_img[..., 2] = resized_img[..., 0]  # R'yi B'ye taşı

# Görüntü boyutlarını al
height, width, channels = rgb_img.shape

# 9x9'luk ortalama filtreyi manuel olarak oluştur
filter_size = 9
padding = filter_size // 2

# Padding uygulanmış yeni görüntü matrisi oluştur
padded_height = height + 2 * padding
padded_width = width + 2 * padding
padded_img = np.zeros((padded_height, padded_width, channels), dtype=np.uint8)
```

```
# Orijinal görüntüyü yeni matrisin ortasına yerleştir
for i in range(height):
    for j in range(width):
        for c in range(channels):
            padded_img[i + padding, j + padding, c] = rgb_img[i, j, c]
# Görüntü boyutunda sıfırlarla dolu bir matris oluşturma
blurred_img = np.zeros((height, width, channels), dtype=np.uint8)

# Görüntüdeki her piksel için 9x9'luk bölgeyi al ve ortalama filtreyi uygula
for i in range(height):
    for j in range(width):
        # 9x9'luk penceredeki bölgeyi al
        region = padded_img[i:i + filter_size, j:j + filter_size]

        # Ortalama hesaplama
        for c in range(channels):  # R, G, B kanalları
            total = 0
            for m in range(filter_size):
                for n in range(filter_size):
                    total += region[m, n, c]
            blurred_img[i, j, c] = total // (filter_size * filter_size)

# Matplotlib ile görüntüleri yan yana gösterme
plt.figure(figsize=(12, 6))  # Görüntü boyutları
plt.subplot(1, 2, 1)
plt.imshow(rgb_img)
plt.title("Orijinal Görüntü")
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(blurred_img)
plt.title("Bulanıklaştırılmış Görüntü")
plt.axis('off')

plt.tight_layout()  #grt göster
```



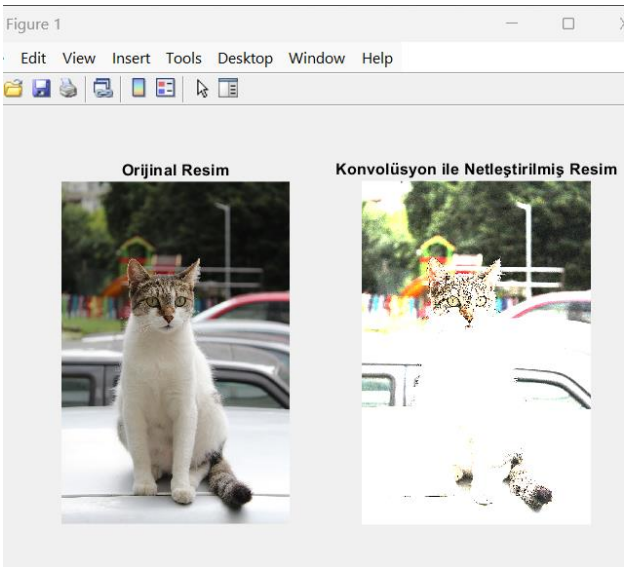
Matlab Kodu:

```
% Görüntüyü yükle
img = imread('cat.jpg');  % Görüntü dosyasını buraya ekleyin
img = im2double(img);    % Görüntüyü double formata dönüştürme
% 5x5 Gauss çekirdeği oluşturma
sigma = 1;  % Standart sapma değeri
filter_size = 5;
[x, y] = meshgrid(-floor(filter_size/2):floor(filter_size/2), -floor(filter_size/2):floor(filter_size/2));
gauss_filter = exp(-(x.^2 + y.^2) / (2 * sigma^2));
gauss_filter = gauss_filter / sum(gauss_filter(:));  % Normalize etme
% Görüntüye Gauss filtresi uygulama
blurred_img = imfilter(img, gauss_filter, 'same');
% Keskinleştirilmiş görüntüyü elde etme
sharpened_img = img + (img - blurred_img);
% Sonuçları gösterme
figure;
subplot(1, 2, 1), imshow(img), title('Orijinal Görüntü');
subplot(1, 2, 2), imshow(sharpened_img), title('Netleştirilmiş Görüntü');
```

2)Konvolüsyon yöntemi (çekirdek matris) ile netleştirme

Konvolüsyon yöntemiyle netleştirme, bir görüntüyü keskinleştirmek için kullanılan bir tekniktir. Bu işlemde, bir çekirdek matris (filtre) görüntüye uygulanarak her pikselin etrafındaki değerlerle çarpılır ve toplanarak yeni bir piksel değeri oluşturulur. Netleştirme, özellikle kenarları ve detayları belirginleştirir, böylece görüntü daha keskin hale gelir. Genellikle "sharpening" filtresi kullanılır.

```
% Giriş resmini yükleyin
GirisResmi = imread('cat.jpg'); % Kendi resim dosyanız ile değiştirin

% Resmin boyutlarını alın
[ResimYuksekligi, ResimGenisligi, ~] = size(GirisResmi);

% Çekirdek matrisi (3x3)
Matris = [0, -2, 0; -2, 11, -2; 0, -2, 0];

% Çekirdek matrisi ile konvolüsyon uygulayın
% Renkli resim üzerinde işlemi yapıyoruz
CikisResmiR = GirisResmi(:,:,1); % Kırmızı kanal
CikisResmiG = GirisResmi(:,:,2); % Yeşil kanal
CikisResmiB = GirisResmi(:,:,3); % Mavi kanal

CikisResmiR = uint8(conv2(double(CikisResmiR), Matris, 'same'));
CikisResmiG = uint8(conv2(double(CikisResmiG), Matris, 'same'));
CikisResmiB = uint8(conv2(double(CikisResmiB), Matris, 'same'));

% Piksel değerlerini [0, 255] aralığına sıkıştırın
CikisResmiR(CikisResmiR > 255) = 255;
CikisResmiR(CikisResmiR < 0) = 0;

CikisResmiG(CikisResmiG > 255) = 255;
CikisResmiG(CikisResmiG < 0) = 0;

CikisResmiB(CikisResmiB > 255) = 255;
CikisResmiB(CikisResmiB < 0) = 0;

% Netleştirilmiş resmi yeniden birleştirin
CikisResmi = cat(3, CikisResmiR, CikisResmiG, CikisResmiB);

% Orijinal ve netleştirilmiş resmi yan yana görselleştirin
figure;
subplot(1, 2, 1);
imshow(GirisResmi);
title('Orijinal Resim');

subplot(1, 2, 2);
imshow(CikisResmi);
title('Konvolüsyon ile Netleştirilmiş Resim');
```



Python Kodu:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Resmi yükle
giris_resmi = cv2.imread("cat.jpg")  # Orijinal renkli resim

# Çekirdek matrisi
matris = [
  [0, -2, 0],
  [-2, 11, -2],
  [0, -2, 0]
]

# Çekirdek boyutu
kernel_size = len(matris)
pad = kernel_size // 2  # Kenarlarda sıfır ekleme miktarı (padding)

# Resmin boyutlarını alın
resim_yuksekligi, resim_genisligi, kanal_sayisi = giris_resmi.shape

# Kenarları sıfırlarla doldur

def pad_image(image, pad, channel):
  padded_image = np.zeros((resim_yuksekligi + 2 * pad, resim_genisligi + 2 * pad), dtype=np.uint8)
  padded_image[pad:pad + resim_yuksekligi, pad:pad + resim_genisligi] = image[:, :, channel]
  return padded_image

# Konvolüsyon işlemini uygula

def apply_convolution(image, kernel):
  output = np.zeros_like(image, dtype=np.uint8)
  for channel in range(kanal_sayisi):
    padded_image = pad_image(image, pad, channel)
    channel_output = np.zeros((resim_yuksekligi, resim_genisligi), dtype=np.uint8)
    for i in range(pad, resim_yuksekligi + pad):
      for j in range(pad, resim_genisligi + pad):
        toplam = 0
        for m in range(kernel_size):
          for n in range(kernel_size):
            toplam += padded_image[i - pad + m, j - pad + n] * kernel[m][n]
```

```
            # Değerleri [0, 255] aralığına sınırla
            channel_output[i - pad, j - pad] = max(0, min(255, toplam))
        output[:, :, channel] = channel_output
    return output

# Konvolüsyon işlemini uygula
netlestirilmis_resim = apply_convolution(giris_resmi, matris)

# Orijinal ve netleştirilmiş resmi göster
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(giris_resmi, cv2.COLOR_BGR2RGB))
plt.title("Orijinal Resim")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(netlestirilmis_resim, cv2.COLOR_BGR2RGB))
plt.title("Konvolüsyon ile Netleştirilmiş Resim")
plt.axis("off")

plt.tight_layout()
plt.show()
```

17) **Perspektif düzeltme**

Perspektif düzeltme, bir görüntüdeki perspektif bozulmalarını giderme işlemidir. Perspektif bozulma, özellikle bir nesneye farklı açılardan bakıldığında nesnelerin boyutlarının değişmesiyle oluşur. Örneğin, yüksek bir binanın alt kısmı büyük, üst kısmı ise küçük görünür. Perspektif düzeltme, bu bozulmaları düzeltmek için dijital araçlar kullanılarak yapılan bir işlemdir. Bu işlem, özellikle fotoğrafçılık, mimarlık ve harita yapımı gibi alanlarda nesnelerin doğru oranlarla görünmesini sağlar.

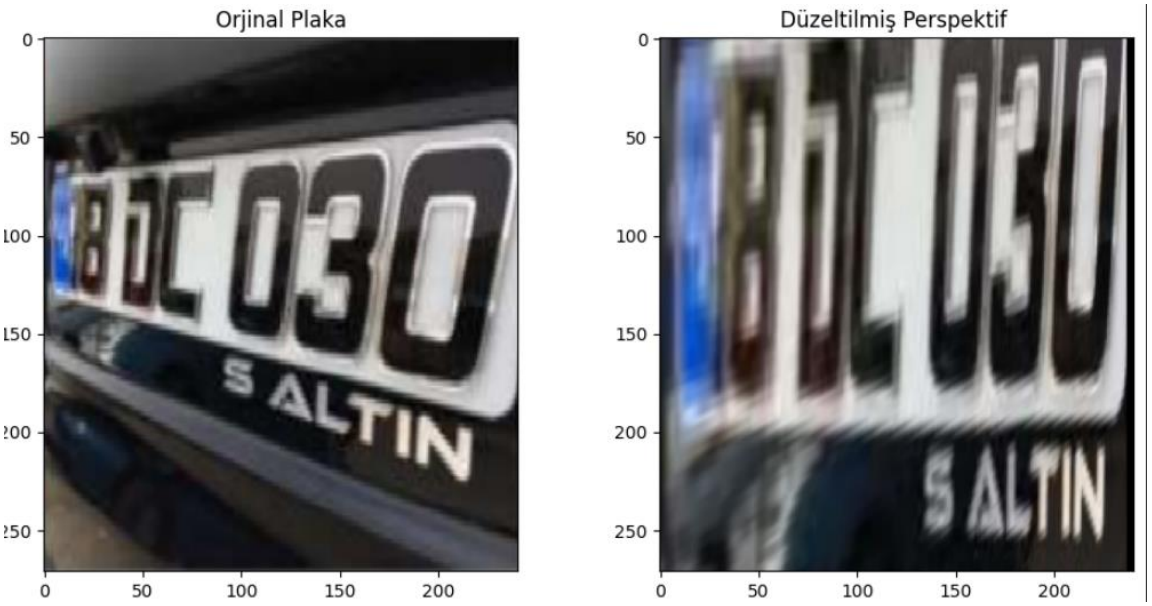
```
plaka_gorseli = imread('Plaka.jpg');

% Kullanıcıdan plakanın dört köşe noktasını seçmesini iste
figure;
imshow(plaka_gorseli);
title('plakanın dört köşesini sırasıyla seçin');
[x_koordinat, y_koordinat] = ginput(4);

% Seçilen köşe noktalarını orijinal görsel üzerinde işaretle
hold on;
plot(x_koordinat, y_koordinat, 'ro-', 'LineWidth', 2);
hold off;

% Düzleştirilmiş plaka için hedef köşe koordinatlarını tanımla
% Bu hedef, plakanın düzleştirilmiş haline göre belirlediğimiz yeni boyutları içeriyor
hedef_genislik = 400; % Hedef genişlik
hedef_yukseklik = 100; % Hedef yükseklik
hedef_koordinatlar = [
    0, 0; % Sol üst köşe
```

```
    hedef_genislik, 0; % Sağ üst köşe
    hedef_genislik, hedef_yukseklik; % Sağ alt köşe
    0, hedef_yukseklik % Sol alt köşe
];

% Seçilen orijinal köşe koordinatları ile hedef koordinatlar arasındaki
% perspektif dönüşüm matrisini hesapla
giris_koordinat = [x_koordinat, y_koordinat];
perspektif_donusum = fitgeotrans(giris_koordinat, hedef_koordinatlar, 'projective');

% Hesaplanan perspektif dönüşümünü kullanarak plakanın düzleştirilmiş
% versiyonunu oluştur
duzeltilmis_plaka_gorseli = imwarp(plaka_gorseli, perspektif_donusum, 'OutputView',
imref2d([hedef_yukseklik, hedef_genislik]));

figure;
subplot(1,2,1);
imshow(plaka_gorseli);
title('Orijinal Resim');

subplot(1,2,2);
imshow(duzeltilmis_plaka_gorseli);
title('Düzeltilmiş Perspektif');
```

Python Kodu:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('Plaka.jpg')

# Orijinal görüntünün boyutlarını al
height, width, _ = img.shape

# Dört köşe noktasını tanımla (Bu noktalar kullanıcının belirlemesi gereken noktalardır)
pts1 = np.float32([[0, 70], [250, 75], [250, 250], [0, 150]])

# Perspektif düzeltme yapmak istediğimiz yeni koordinatlar
pts2 = np.float32([[0, 0], [width, 0], [width, height], [0, height]])

# 2D homografi matrisini hesaplamak için denklem kuruyoruz.
def compute_homography(pts1, pts2):
  A = []
  for i in range(4):
    x, y = pts1[i][0], pts1[i][1]
    x_prime, y_prime = pts2[i][0], pts2[i][1]
    A.append([-x, -y, -1, 0, 0, 0, x*x_prime, y*x_prime, x_prime])
    A.append([0, 0, 0, -x, -y, -1, x*y_prime, y*y_prime, y_prime])

  # Denklemi çözmek için A matrisinin SVD (singular value decomposition) kullanarak çözümünü yapıyoruz
  A = np.array(A)
  _, _, VT = np.linalg.svd(A)
  H = VT[-1].reshape(3, 3)  # Son satırdan homografi matrisini alıyoruz
  return H

# Homografi matrisini hesapla
H = compute_homography(pts1, pts2)

# Perspektif dönüşüm için bu matrisi kullanıyoruz
def warp_perspective(img, H, output_size):
  height, width = output_size
  result = np.zeros((height, width, 3), dtype=np.uint8)

  # Ters homografi matrisini al
  H_inv = np.linalg.inv(H)

  # Görüntüdeki her piksel için ters dönüşüm hesaplanır
  for y in range(height):
    for x in range(width):
      # Pikselin yeni koordinatını almak için H^-1'ı kullan
      coords = np.array([x, y, 1])
```

```
        orig_coords = np.dot(H_inv, coords)
        orig_coords /= orig_coords[2]  # Normalize et
        orig_x, orig_y = int(orig_coords[0]), int(orig_coords[1])

        if 0 <= orig_x < img.shape[1] and 0 <= orig_y < img.shape[0]:
            result[y, x] = img[orig_y, orig_x]

    return result

# Perspektif dönüşümünü uygula
perspective_corrected_img = warp_perspective(img, H, (height, width))

# Sonuçları görselleştirme
plt.figure(figsize=(10, 5))


plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))  # Görüntüyü RGB formatına dönüştürerek gösteriyoruz
plt.title('Orjinal Plaka')

# Perspektif düzeltme yapılmış Görüntü
plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(perspective_corrected_img, cv2.COLOR_BGR2RGB))  # Görüntüyü RGB formatına
dönüştürerek gösteriyoruz
plt.title('Düzeltilmiş Perspektif')

plt.tight_layout()
plt.show()
```



18) Çapraz Korelasyon (Cross Corelation)

Çapraz Korelasyon (Cross-Correlation), görüntü işleme alanında, iki sinyal veya görüntü arasındaki benzerliği ölçmek için kullanılan bir tekniktir. Genellikle bir görüntüyü veya sinyali başka bir görüntü veya sinyal üzerinde kaydırarak (shifting) her iki veri setinin örtüşen bölgelerinde matematiksel bir ilişki hesaplanır. Bu işlem, özellikle şablon eşleşme (template matching) gibi uygulamalarda, bir görüntüde belirli bir desenin yerini tespit etmek için kullanılır.

Çapraz korelasyon, genellikle şu şekilde hesaplanır:

1. Bir görüntü ve bir şablon (diğer görüntü) seçilir.
2. Şablon, görüntü üzerinde farklı konumlarda kaydırılır.
3. Her kaydırma adımında, şablon ile görüntü arasındaki benzerlik ölçülür (genellikle noktasal çarpanların toplamı alınır).
4. En yüksek benzerlik değeri, şablonun görüntüdeki en iyi eşleştiği konumu gösterir.

Bu yöntem, görüntüdeki arama işlemleri, nesne tanıma ve analiz gibi görevlerde yaygın olarak kullanılır.

Python:

```
import cv2

import numpy as np

import matplotlib.pyplot as plt


# Görsellerin gri tonlamalı olarak okunması

main_image = cv2.imread('matkap.jpg', cv2.IMREAD_GRAYSCALE)  # Ana görüntü

drill_tip_image = cv2.imread('saglam_matkap.jpg', cv2.IMREAD_GRAYSCALE)  # Şablon görüntü (matkap ucu)


# Normalleştirilmiş çapraz korelasyon işlemi (manuel hesaplama)

def normalized_cross_correlation(template, image):

  # Şablon ve ana görüntü boyutlarını al

  template_height, template_width = template.shape

  image_height, image_width = image.shape


  # Çıkış korelasyon matrisi oluştur

  correlation_matrix = np.zeros((image_height - template_height + 1, image_width - template_width + 1))


  # Şablonun ortalamasını ve standart sapmasını hesapla

  template_mean = np.mean(template)

  template_std = np.std(template)
```

```
    # Şablonu normalize et
    normalized_template = (template - template_mean) / template_std

    # Görüntü üzerinde her bir konum için çapraz korelasyonu hesapla
    for i in range(correlation_matrix.shape[0]):
        for j in range(correlation_matrix.shape[1]):
            # Şablon boyutunda bir bölge seç
            region = image[i:i + template_height, j:j + template_width]
            # Bölgenin ortalama ve standart sapmasını hesapla
            region_mean = np.mean(region)
            region_std = np.std(region)

            # Eğer bölgenin standart sapması sıfır değilse, normalize et ve korelasyonu hesapla
            if region_std != 0:
                normalized_region = (region - region_mean) / region_std
                correlation_matrix[i, j] = np.sum(normalized_template * normalized_region)

    return correlation_matrix

# Şablon ile ana görüntü arasındaki çapraz korelasyonu hesapla
correlation_result = normalized_cross_correlation(drill_tip_image, main_image)

# Korelasyon sonucunda en yüksek değeri ve konumunu bul
max_correlation = np.max(correlation_result)
max_index = np.unravel_index(np.argmax(correlation_result), correlation_result.shape)
top_y, top_x = max_index

# Şablonun ana görüntüdeki konumunu belirle
drill_tip_height, drill_tip_width = drill_tip_image.shape
top_left_x = top_x
top_left_y = top_y

# Ana görüntü üzerine tespit edilen bölgeyi dikdörtgenle çiz
plt.figure()
```

```
plt.imshow(main_image, cmap='gray')
plt.title('Sağlam Matkap Ucu Tespit Edildi')
plt.axis('off')
rectangle = plt.Rectangle((top_left_x, top_left_y), drill_tip_width, drill_tip_height, edgecolor='r', linewidth=2, fill=False)
plt.gca().add_patch(rectangle)
plt.show()

# Tespit edilen sonuçları yazdır
print(f"En yüksek korelasyon değeri: {max_correlation:.2f}")
print(f"Matkap ucu sol üst köşesi: (X: {top_left_x}, Y: {top_left_y})")
```