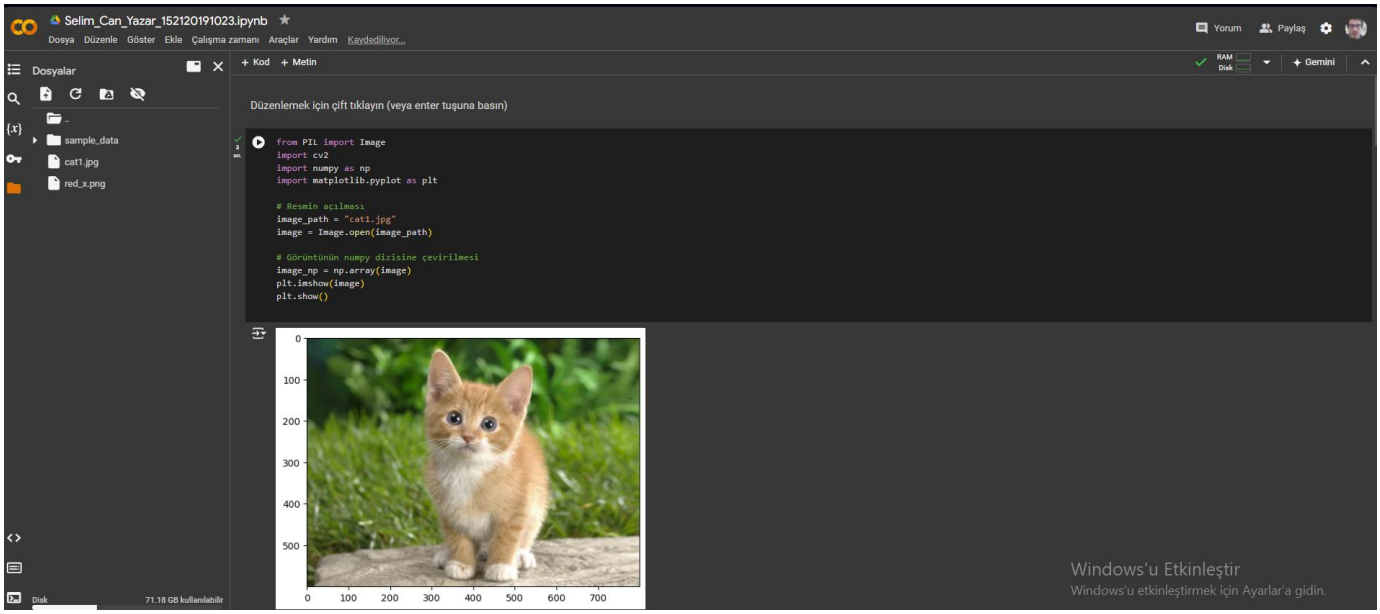
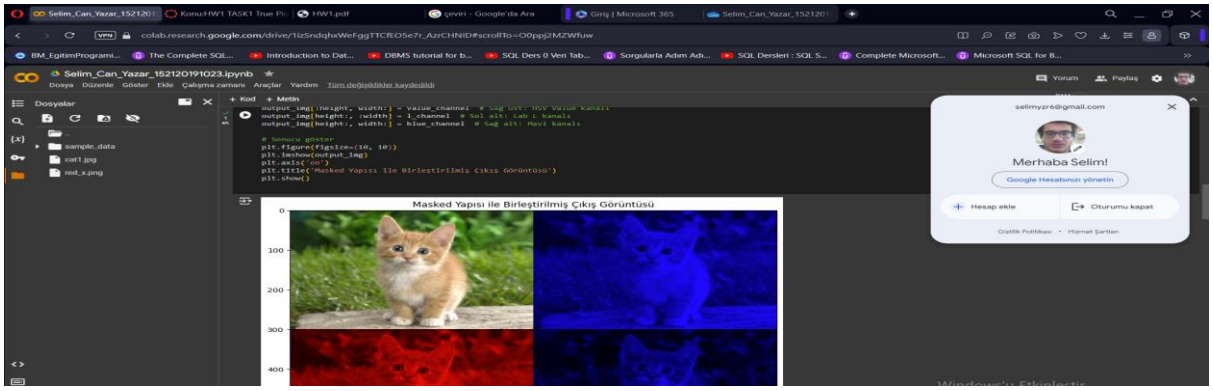
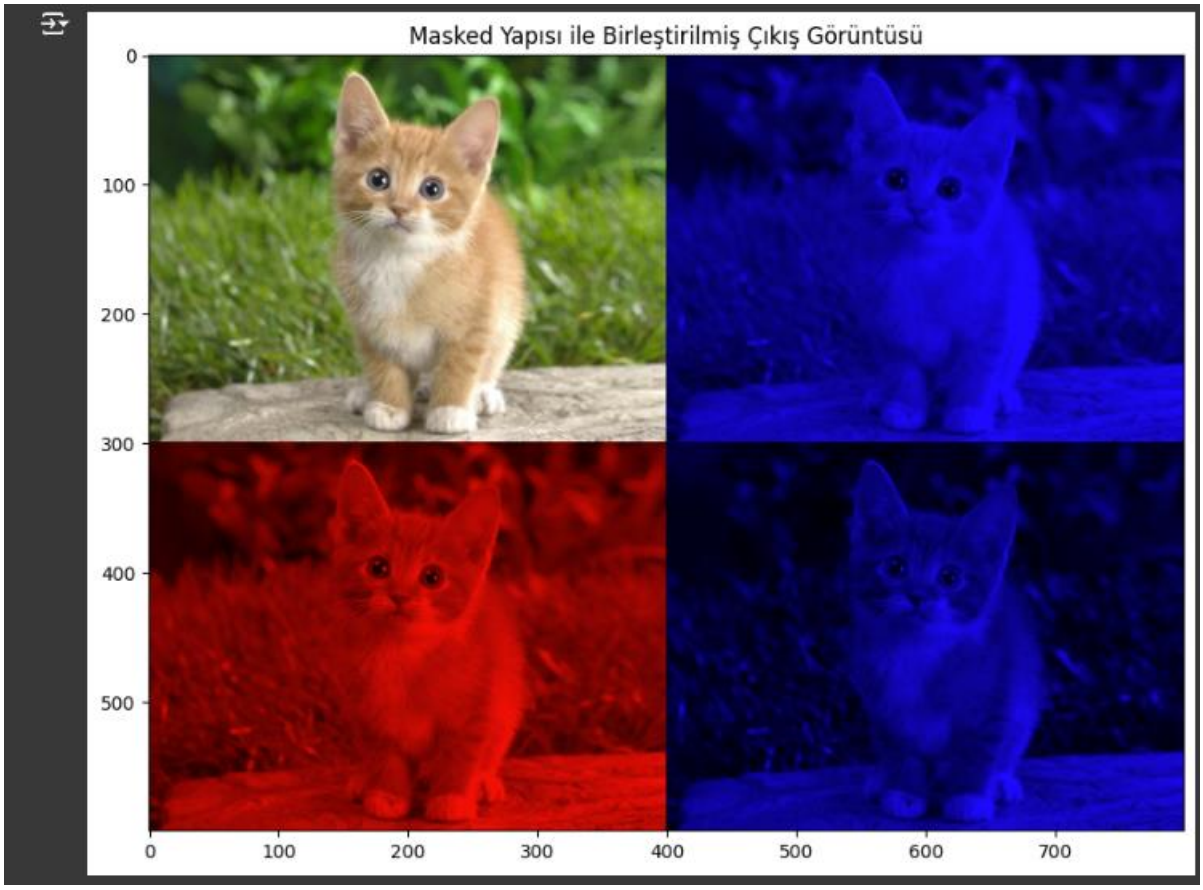


Ödeve başlamadan önce öncelikle ödevin bu kısmında bir resim dosyasını açıp üzerinde işlem yapabilmek için gerekli hazırlıkları yaptım daha sonra resmi ekranda gösterdim. İlk olarak, ödevde kullanılan kütüphaneler olan: `PIL` kütüphanesinden `Image` sınıfı, OpenCV (`cv2`), `numpy`, ve `matplotlib.pyplot` ('plt' olarak kısaltılmış) içe aktardım. Bu kütüphanelerin görüntü işleme ve görselleştirme işlemleri için geniş bir araç seti sunduğunu öğrendim. `PIL`, resim dosyalarını açmak ve temel işlemler yapmak için kullanılırken, `numpy` görüntü verilerini bir dizi olarak işleyebilmemizi sağlar böylece görüntü matematiksel olarak analiz edilebiliyor. `matplotlib` ise bu verileri görselleştirmek, yani ekranda göstermek için kullanılıyor. Ödevde kodun ana kısmında, resmin yolu belirtilerek `Image.open(image_path)` fonksiyonu ile resim dosyası açılıyor. Bu resim, `image` adında bir değişkende saklanıyor. Ardından, bu görüntü `numpy` dizisine çevriliyor. Bu dönüşüm sayesinde resim, piksellerden oluşan sayısal bir matris haline geliyor ve bu şekilde üzerinde daha rahat işlem yapılabiliyor. Son olarak, `plt.imshow(image)` ve `plt.show()` fonksiyonları ile resim ekranda gösteriliyor. Bu görselleştirme, resmi adım adım inceleyebilmek ve işleme sürecinde neler olduğunu gözlemleyebilmemizi sağlıyor. Bu aşamalar genellikle görüntü işleme projelerinin başlangıç adımlarını oluşturur. (Sağ üst köşede profil resmi gözükmektedir.)



Task1

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

# Orijinal resmi yüklenmesi
image_path = '/content/cat1.jpg'
original_img = cv2.imread(image_path)

# cv2 kütüphanesi BGR formatında yüklendiği için RGB'ye çevirilmesi
original_img_rgb = cv2.cvtColor(original_img, cv2.COLOR_BGR2RGB)

# Resmi yarı boyutuna küçültülmesi
resized_img = cv2.resize(original_img_rgb, (0, 0), fx=0.5, fy=0.5)

# Resmin HSV ve Lab renk paletlerine dönüştürülmesi (RGB temelinde)
hsv_img = cv2.cvtColor(resized_img, cv2.COLOR_RGB2HSV)
lab_img = cv2.cvtColor(resized_img, cv2.COLOR_RGB2Lab)

# HSV Value kanalının maske yapısı ile çıkartılması
hsv_value_masked = hsv_img.copy()
hsv_value_masked[:, :, 0] = 0  # H kanalını sıfırla
hsv_value_masked[:, :, 1] = 0  # S kanalını sıfırla
value_channel = hsv_value_masked  # Yalnızca Value kanalını içerir

# Lab'ın L kanalının maske yapısı ile çıkartılması
lab_l_masked = lab_img.copy()
lab_l_masked[:, :, 1] = 0  # a kanalını sıfırla
lab_l_masked[:, :, 2] = 0  # b kanalını sıfırla
l_channel = lab_l_masked

# RGB resminden sadece Mavi kanalın maske ile çıkartılması
blue_channel_masked = resized_img.copy()
blue_channel_masked[:, :, 0] = 0  # G kanalını sıfırla
blue_channel_masked[:, :, 1] = 0  # R kanalını sıfırla
blue_channel = blue_channel_masked

# Orijinal resim ile aynı boyutta boş bir çıkış görüntüsü hazırlanması
height, width, _ = resized_img.shape
output_img = np.zeros((height * 2, width * 2, 3), dtype=np.uint8)

# Her bir çeyreği çıkış görüntüsüne yerleştir
output_img[:height, :width] = resized_img  # Sol üst: Orijinal
output_img[:height, width:] = value_channel  # Sağ üst: HSV Value kanalı
output_img[height:, :width] = l_channel  # Sol alt: Lab L kanalı
output_img[height:, width:] = blue_channel  # Sağ alt: Mavi kanalı

# Sonucu göster
plt.figure(figsize=(10, 10))
plt.imshow(output_img)
plt.axis('on')
plt.title('Masked Yapısı ile Birleştirilmiş Çıkış Görüntüsü')
plt.show()
```



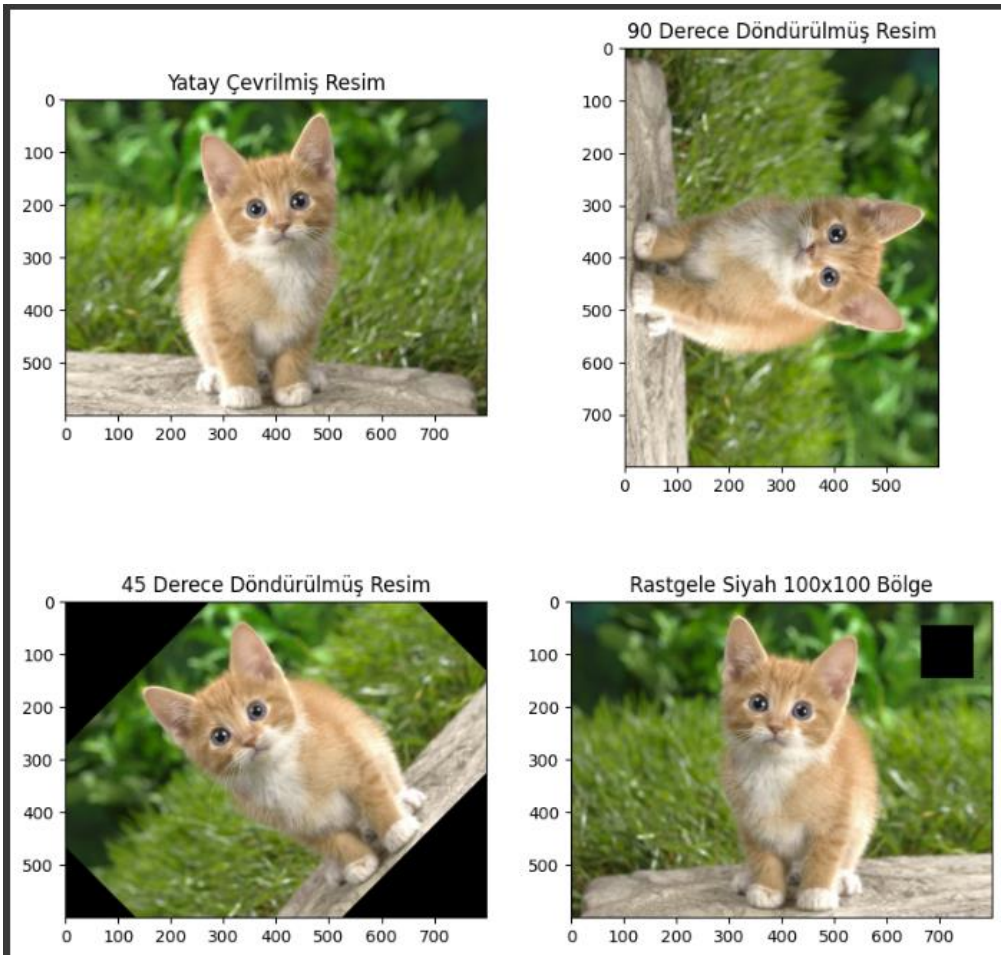
Task 1 de bir kedi görüntüsünü işleyerek farklı renk paletlerinde ayrı çeyreklerde birleştirip tek bir çıktı elde etmeyi amaçladık. İlk adımda, OpenCV kullanılarak belirtilen dosya yolundaki resim yükledim ve ardından RGB formatına çevirdim. Bu dönüşüm OpenCV'nin varsayılan olarak resimleri BGR formatında yüklemesi nedeniyle gereklidir. Daha sonra resim boyut olarak yarıya küçültülerek orijinal resim boyutuna 4 resiminde sığması sağlandı.

Task 1 de devamında görüntüyü farklı renk uzaylarına dönüştürdüm. RGB formatındaki görüntü, sırasıyla HSV ve Lab renk alanlarına çevrildi. HSV renk uzayı, görüntünün Hue (H), Saturation (S) ve Value (V) kanallarına ayrıldığı bir format sunar. Burada yalnızca parlaklık bilgisi olan V kanalını izole edebilmek için H ve S kanalları sıfırlanır. Aynı işlem, Lab renk uzayı için de uygulanır ve yalnızca parlaklık değerini taşıyan L kanalı kalacak şekilde diğer kanallar sıfırlandı. Benzer şekilde, RGB formatındaki görüntüde sadece mavi kanalın görünür kalması sağlandı.

Kodun son kısmında, bu renk kanalları dört çeyreğe ayrılmış tek bir çıktı görüntüsünde birleştirdim. Sol üst çeyrekte orijinal görüntü, sağ üstte HSV'nin yalnızca V kanalı, sol altta Lab'ın L kanalı ve sağ altta ise mavi kanal bulunmasını sağladım. Bu şekilde, görüntünün farklı renk bileşenleri tek bir karede görüntülenir ve `matplotlib` ile başlık altında eksenleriyle birlikte görselleştirilir. Resimlerin farklı renk uzaylarını tek bir görselde bir araya getirmek renk bilgisi analizi için etkili bir seçenek sunmaktadır.

Task 2

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

# 1. Orijinal resmin yüklenmesi
image_np = cv2.imread('cat1.jpg')

# 2. Orijinal resmi kullan
image_resized = image_np  # Boyut değiştirmeden kullanıyoruz

# 3. Orijinal görüntünün yatay çevirilmesi
flipped_image = cv2.flip(image_resized, 1)

# 4. 90 derece döndürülmesi
rotated_90 = cv2.rotate(image_resized, cv2.ROTATE_90_CLOCKWISE)

# 5. 45 derece döndürme
(h, w) = image_resized.shape[:2]
center = (w // 2, h // 2)
M = cv2.getRotationMatrix2D(center, 45, 1.0)
rotated_45 = cv2.warpAffine(image_resized, M, (w, h))

# 6. Rastgele 100x100 bir alanın sıfırlanması
image_with_zeros = image_resized.copy()
x, y = np.random.randint(0, image_resized.shape[1] - 100), np.random.randint(0, image_resized.shape[0] - 100)
image_with_zeros[y:y+100, x:x+100] = 0

# Görüntüleri gösterelim
fig, axs = plt.subplots(2, 2, figsize=(10, 10))

axs[0, 0].imshow(cv2.cvtColor(flipped_image, cv2.COLOR_BGR2RGB))  # BGR'den RGB'ye çevirilmesi
axs[0, 0].set_title("Yatay Çevrilmiş Resim")
axs[0, 1].imshow(cv2.cvtColor(rotated_90, cv2.COLOR_BGR2RGB))  # BGR'den RGB'ye çevirilmesi
axs[0, 1].set_title("90 Derece Döndürülmüş Resim")
axs[1, 0].imshow(cv2.cvtColor(rotated_45, cv2.COLOR_BGR2RGB))  # BGR'den RGB'ye çevirilmesi
axs[1, 0].set_title("45 Derece Döndürülmüş Resim")
axs[1, 1].imshow(cv2.cvtColor(image_with_zeros, cv2.COLOR_BGR2RGB))  # BGR'den RGB'ye çevirilmesi
axs[1, 1].set_title("Rastgele Siyah 100x100 Bölge")

for ax in axs.flat:
    ax.axis('on')

plt.show()
```



Task 2 de ödevdeki kedi resmi çeşitli manipülasyonlarla dönüştürerek farklı versiyonlarını oluşturuyor ve bu dönüşümleri görselleştiriyoruz. İlk olarak, `cv2.imread('cat1.jpg')` kullanılarak belirtilen yol üzerinden resim yükleniyor ve bu yüklenmiş görüntü `image_np` değişkenine atanıyor. Bu resim, OpenCV formatında (BGR renk düzeni) saklanıyor ve üzerinde farklı dönüşümler yapılacak.

Orijinal resim herhangi bir yeniden boyutlandırma işlemi yapılmadan kullanılıyor ve `image_resized` değişkenine atanıyor. Ardından bu resim, `cv2.flip(image_resized, 1)` fonksiyonu kullanılarak yatay çevriliyor. Bu işlem, resmin sağ ve sol taraflarını yer değiştirerek bir ayna görüntüsü oluşturuyor ve `flipped_image` değişkeninde saklanıyor.

Bir sonraki adımda orijinal görüntünün saat yönünde 90 derece döndürülmesi sağlanıyor. Bu işlem, `cv2.rotate(image_resized, cv2.ROTATE_90_CLOCKWISE)` kullanılarak yapılıyor ve bu dönüştürülmüş görüntü `rotated_90` değişkenine atanıyor. Bu dönüşüm sayesinde, görüntünün dikey ve yatay boyutları yer değiştiriyor.

Bir başka dönüştürme işlemi olarak, görüntü 45 derece döndürülüyor. Bunun için öncelikle görüntünün merkez noktaları belirleniyor ve `cv2.getRotationMatrix2D(center, 45, 1.0)` kullanılarak bir dönüş matrisi oluşturuluyor. Bu matris, görüntünün 45 derece dönüştürülmesi için gerekli olan koordinat dönüşümünü tanımlar. `cv2.warpAffine(image_resized, M, (w, h))` kullanılarak bu dönüş matrisi uygulanıyor ve 45 derece döndürülmüş görüntü `rotated_45` değişkeninde saklanıyor. Bu dönüşüm, görüntünün daha çapraz bir perspektifte görülmesini sağlıyor.

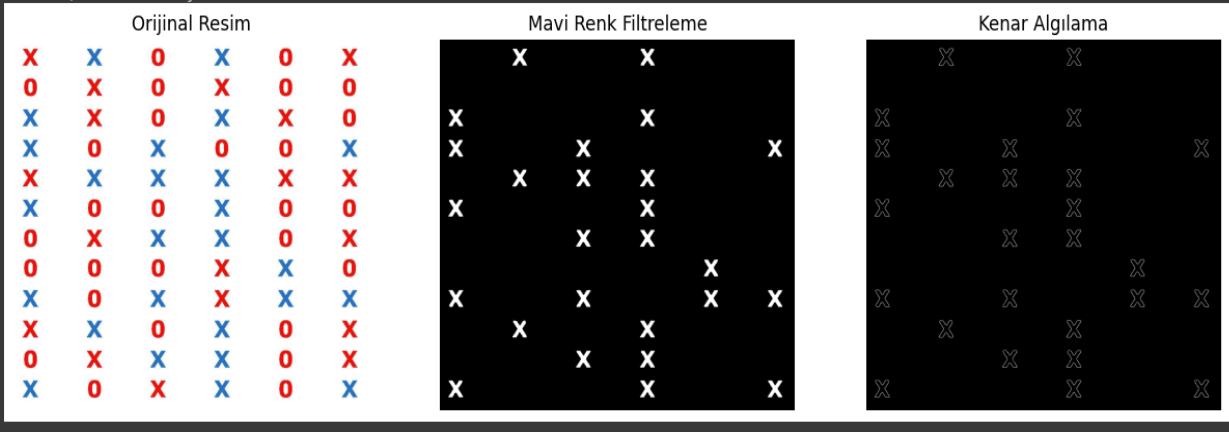
Son olarak, görüntüde rastgele bir alanda boşluk oluşturma işlemi gerçekleştiriliyor. Bunun için öncelikle orijinal görüntünün bir kopyası oluşturuluyor ve bu kopya üzerinde değişiklik yapılıyor. `np.random.randint(0, image_resized.shape[1] - 100)` ve benzeri ifadeler kullanılarak görüntüde rastgele bir koordinat belirleniyor ve bu koordinattan başlayarak 100x100 boyutlarındaki bir alan siyaha boyanıyor. Bu işlem, görüntüde belirli bir bölgenin üzerinde değişiklik yaparak farklı etkilerin incelenmesini sağlıyor ve `image_with_zeros` değişkeninde saklanıyor.

Bütün bu dönüştürülmüş görüntüler, matplotlib kullanılarak bir dizi halinde görselleştiriliyor. `plt.subplots(2, 2, figsize=(10, 10))` kullanılarak 2x2'lik bir alt görsel matrisi oluşturuluyor ve her bir dönüşüm bu alt görsellerde ekranda gösteriliyor. Bu sayede, resmin farklı dönüşümlerinin etkileri yan yana görülebiliyor ve her bir dönüşümün görüntü üzerindeki etkisi detaylıca incelenebiliyor.

Task 3

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

# Yüklenmiş resmin okunması
image = cv2.imread('red_x.png')

# RGB formatına dönüştürülmesi
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Resmin HSV formatına çevirilmesi
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# Mavi renk aralığını HSV'de tanımlanması
lower_blue = np.array([100, 150, 50])  # HSV renk değerlerinin alt sınırı
upper_blue = np.array([140, 255, 255])  # HSV renk değerlerinin üst sınırı

# Mask oluşturulması (mavi olan bölgeleri beyaz, geri kalanları siyah yapacak)
mask = cv2.inRange(hsv, lower_blue, upper_blue)

# Maskeye morfolojik işlemler uygulayarak gürültünün azaltılması
kernel = np.ones((3, 3), np.uint8)
mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

# Kenar algılama (Canny edge detection)
edges = cv2.Canny(mask, 50, 150)

# Şekillerin bulunması (contours)
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Mavi X işaretlerinin sayısının bulunması
x_count = 0
for cnt in contours:
    # Şeklin alanını kontrol edelim (X işareti olduğuna karar vermek için)
    area = cv2.contourArea(cnt)

    # X işaretleri belirli bir alan aralığında olmalı, çok küçük veya çok büyük şekilleri ihmal ediyoruz
    if 100 < area < 5000:  # Bu değerleri X'in boyutuna göre ayarlandı
        # Konturun minimum bounding rectangle (dikdörtgen çerçevesi)
        rect = cv2.minAreaRect(cnt)
        width, height = rect[1]

        # X işaretlerinin şeklinin kapladığı alanın düzgün dörtgen biçiminde olduğunu varsayıyoruz
        if 0.8 < width / height < 1.2:  # Genişlik ve yükseklik oranı yaklaşık 1 olmalı
            x_count += 1

print(f"Mavi X işaretlerinin sayısı: {x_count}")

# Sonuçların görselleştirilmesi (orijinal resim ve maske)
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

axs[0].imshow(image_rgb)
axs[0].set_title("Orijinal Resim")
axs[0].axis('off')

axs[1].imshow(mask, cmap='gray')
axs[1].set_title("Mavi Renk Filtreleme")
axs[1].axis('off')

axs[2].imshow(edges, cmap='gray')
axs[2].set_title("Kenar Algılama")
axs[2].axis('off')

plt.show()
```

```
Mavi X işaretlerinin sayısı: 26
```



Task 3 de verilen bir resim üzerinde mavi renkteki X işaretlerini algılamak ve sayısını belirlemek üzere bir dizi görüntü işleme adımı gerçekleştiriyor. İlk olarak, `cv2.imread('red_x.png')` fonksiyonuyla belirtilen resim yükleniyor ve bu resim OpenCV formatında (BGR renk düzeni) saklanıyor. Daha sonra, `cv2.cvtColor(image, cv2.COLOR_BGR2RGB)` kullanılarak bu görüntü RGB formatına dönüştürülüyor; bu, matplotlib ile görüntüleme işlemlerinde renklerin doğru gösterilmesi için yapılıyor.

Ardından, resim HSV renk uzayına çevriliyor (`cv2.cvtColor(image, cv2.COLOR_BGR2HSV)`), bu renk uzayı özellikle belirli renkleri filtrelemek için uygun bir yapı sunuyor. Kodda, mavi rengin aralığı tanımlanıyor: `lower_blue` ve `upper_blue` değişkenleri ile HSV renk değerlerinin alt ve üst sınırları belirleniyor. Bu sınırlar kullanılarak, `cv2.inRange(hsv, lower_blue, upper_blue)` fonksiyonu ile mask oluşturuluyor. Bu maske, mavi renkli bölgeleri beyaz, geri kalan bölgeleri ise siyah olarak belirliyor.

Maskeyi oluşturduktan sonra, görüntüdeki gürültüyü azaltmak için morfolojik işlemler uygulanıyor. Öncelikle, 3x3 boyutunda bir kernel oluşturuluyor ve bu kernel kullanılarak `cv2.morphologyEx` fonksiyonuyla sırasıyla açılma (görüntüdeki küçük beyaz gözleri temizlemek) ve kapanma (siyah bölgeleri kapatarak çevresindeki beyaz bölgelerle birleştirmek) işlemleri yapılıyor. Bu işlemler, maske üzerindeki gürültülerin azaltılması ve daha temiz bir görüntü elde edilmesi için kullanılıyor.

Daha sonra, kenar algılama işlemi uygulanıyor. `cv2.Canny(mask, 50, 150)` fonksiyonu kullanılarak kenarlar belirleniyor. Bu işlem, maske üzerindeki keskin geçişleri tespit ederek X işaretlerinin kenarlarını belirginleştiriyor. Ardından, `cv2.findContours` fonksiyonu ile bu kenarlar üzerindeki konturlar bulunuyor. Konturlar, görüntüdüki kapalı bölgeleri temsil eden çizgilerdir ve bu durumda X işaretlerini tespit etmek için kullanılıyor.

Bulunan her bir konturun alanı `cv2.contourArea(cnt)` kullanılarak hesaplanıyor ve belirli bir alan aralığında olanlar seçiliyor (çok küçük veya çok büyük konturlar ihmal ediliyor). Bu sayede, belirli bir boyuttaki X işaretleri hedef alınmış oluyor. Ardından, her bir kontur için minimum alan kaplayan dikdörtgen (`cv2.minAreaRect(cnt)`) bulunuyor ve bu dikdörtgenin genişlik ve yükseklik oranı kontrol ediliyor. Oran yaklaşık olarak 1 ise, bu konturun bir X işareti olduğuna karar veriliyor ve sayılıyor.

Son olarak, orijinal resim, mavi renk filtresi uygulanmış maske ve kenar algılama sonucu elde edilen görüntüler matplotlib kullanılarak gösteriliyor. Bu görselleştirme, kullanıcıya işlem adımlarının her birini gözlemleme imkânı tanıyor ve algoritmanın nasıl çalıştığını anlamaya yardımcı oluyor. Kodun sonunda, tespit edilen mavi X işaretlerinin sayısı ekrana yazdırılıyor.