محسن علی محمدی 96440296

موضوع: banana

دیتا ست: استفاده از عکس های گیت <https://github.com/techbless/banana-or-not/tree/master/data> برای آموزش و تست که بصورت رندوم 64 عکس آموزش(32 تا موز و 32 تا غیر موز) و برای 30 عکس برای تست(15 تا موز و 15 تا غیر موز) انتخاب شدن.

نمونه داده آموزشی:

موز:





غیر موز:

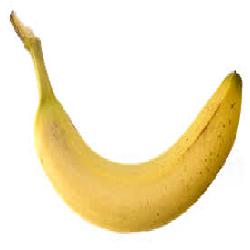
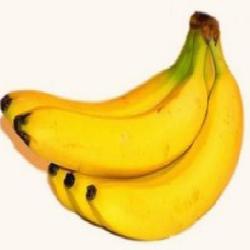
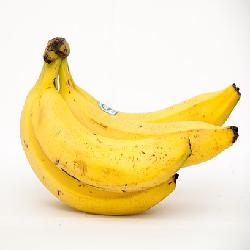






نمونه داده تست:

موز:



غیر موز:



توابع استفاده شده:

def histogram\_equalization(img\_in):

*# segregate color streams*

    b, g, r = cv2.split(img\_in)

    h\_b, \_ = np.histogram(b, 256, [0, 256])

    h\_g, \_ = np.histogram(g, 256, [0, 256])

    h\_r, \_ = np.histogram(r, 256, [0, 256])

*# Cumulative distribution function*

    cdf\_b = np.cumsum(h\_b)

    cdf\_g = np.cumsum(h\_g)

    cdf\_r = np.cumsum(h\_r)

*# Normalize the cumulative distribution function*

    cdf\_norm\_b = np.ma.masked\_equal(cdf\_b, 0)

    cdf\_norm\_b = (

        (cdf\_norm\_b - cdf\_norm\_b.min()) \* 255 / (cdf\_norm\_b.max() - cdf\_norm\_b.min())

    )

    cdf\_final\_b = np.ma.filled(cdf\_norm\_b, 0).astype("uint8")

    cdf\_norm\_g = np.ma.masked\_equal(cdf\_g, 0)

    cdf\_norm\_g = (

        (cdf\_norm\_g - cdf\_norm\_g.min()) \* 255 / (cdf\_norm\_g.max() - cdf\_norm\_g.min())

    )

    cdf\_final\_g = np.ma.filled(cdf\_norm\_g, 0).astype("uint8")

    cdf\_norm\_r = np.ma.masked\_equal(cdf\_r, 0)

    cdf\_norm\_r = (

        (cdf\_norm\_r - cdf\_norm\_r.min()) \* 255 / (cdf\_norm\_r.max() - cdf\_norm\_r.min())

    )

    cdf\_final\_r = np.ma.filled(cdf\_norm\_r, 0).astype("uint8")

*# merge the images in the three channels*

    img\_b = cdf\_final\_b[b]

    img\_g = cdf\_final\_g[g]

    img\_r = cdf\_final\_r[r]

    img\_out = cv2.merge((img\_b, img\_g, img\_r))

*return* img\_out

ابتداد مولفه های رنگی (b,g,r) به از هم جدا میشوند و هیستوگرام هر کدام محاسبه میشود. با استفاده ازcumsum ، تابع توزیع تجمعی هر کدام از مولفه های رنگی محاسبه شده و عمل همسان سازی هیستوگرام برای تمام آنها انجام میشود و در انتهای این تابع مولفه های رنگی با هم ترکیب میشوند.

def sharpening(img\_in):

    kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])

    sharpened = cv2.filter2D(img\_in, -1, kernel)

*return* sharpened

اعمال فیلتر sharpning روی تصویر.

def edge\_detection(channel):

    sobelX = cv2.Sobel(channel, cv2.CV\_16S, 1, 0)

    sobelY = cv2.Sobel(channel, cv2.CV\_16S, 0, 1)

    sobel = np.hypot(sobelX, sobelY)

    sobel[sobel > 255] = 255

*return* sobel

def segmentation\_by\_edge(img\_in):

*# step1: remove noise*

    img = cv2.GaussianBlur(img\_in, (5, 5), 0)

*# step2: edge detection*

*# finding the max intesity from among the R,G and B edges.*

    img\_edge = np.max(

        np.array(

            [

                edge\_detection(img[:, :, 0]),

                edge\_detection(img[:, :, 1]),

                edge\_detection(img[:, :, 2]),

            ]

        ),

        axis=0,

    )

*# step3: Remove the false edge*

    mean = np.mean(img\_edge)

    img\_edge[img\_edge <= mean \* 1.5] = 0

*# step4 , step5 : Loss of rupture and find contours*

    img\_edge = np.asarray(img\_edge, np.uint8)

    contours, heirarchy = cv2.findContours(

        img\_edge, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE

    )

    contours\_level1 = []

*for* i, tupl in enumerate(heirarchy[0]):

*if* tupl[3] == -1:

            tupl = np.insert(tupl, 0, [i])

            contours\_level1.append(tupl)

    contours\_significant = []

    contors\_tooSmall = img\_edge.size \* 5 / 100

*for* tupl in contours\_level1:

        contour = contours[tupl[0]]

        area = cv2.contourArea(contour)

*if* area > contors\_tooSmall:

            contours\_significant.append([contour, area])

            cv2.drawContours(img, [contour], 0, (0, 255, 0), 2, cv2.LINE\_AA, maxLevel=1)

    contours\_significant.sort(key=lambda x: x[1])

    significant = [x[0] *for* x in contours\_significant]

*return* (img\_edge, significant)

عمل تقطیع تصویر با استفاده از تشخیص لبه که دقیقا همان مراحل الگوریتمی که در کلاس گفته شد انجام میشود و در کد هم کامنت گذاری شده است.

def remove\_background(img\_in):

    img\_edge, significant = segmentation\_by\_edge(img\_in)

    mask = img\_edge.copy()

    mask[mask > 0] = 0

    cv2.fillPoly(mask, significant, 255)

    mask = np.logical\_not(mask)

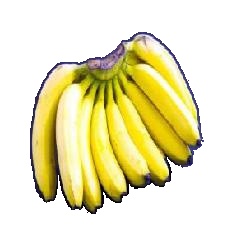
    img\_in[mask] = 255

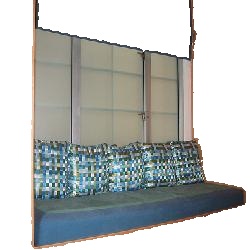
*return* img\_in, mask

ماسکی به اندازه تصویر در نظر گرفته میشود و با استفاده از تابع قبلی نقاطی که درداخل ناحیه شئ نیست 255 (سفید) میشود.

نمونه خروجی پس از حذف بک گراند:







def feature\_extraction(img\_in, mask):

*# mean and median and*

    feature = []

    feature.append(int(np.mean(img\_in[:, :, 0][mask == 0])))

    feature.append(int(np.mean(img\_in[:, :, 1][mask == 0])))

    feature.append(int(np.mean(img\_in[:, :, 2][mask == 0])))

    feature.append(int(np.median(img\_in[:, :, 0][mask == 0])))

    feature.append(int(np.median(img\_in[:, :, 1][mask == 0])))

    feature.append(int(np.median(img\_in[:, :, 2][mask == 0])))

*return* feature

میانگین و میانه مولفه های رنگی عکس که mask در آنها false است محاسبه میشود و در آرایه feature ذخیره میشود.

def nemmodar(x\_train, x\_test):

    x1 = []

    y1 = []

    z1 = []

*for* i in range(32):

        x1.append(x\_train[i][0])

        y1.append(x\_train[i][1])

        z1.append(x\_train[i][2])

    x2 = []

    y2 = []

    z2 = []

*for* i in range(32):

        x2.append(x\_train[i+32][0])

        y2.append(x\_train[i+32][1])

        z2.append(x\_train[i+32][2])

    xt1 = []

    yt1 = []

    zt1 = []

*for* i in range(15):

        xt1.append(x\_test[i][0])

        yt1.append(x\_test[i][1])

        zt1.append(x\_test[i][2])

    xt2 = []

    yt2 = []

    zt2 = []

*for* i in range(15):

        xt2.append(x\_test[i+15][0])

        yt2.append(x\_test[i+15][1])

        zt2.append(x\_test[i+15][2])

    fig = plt.figure()

    ax = fig.add\_subplot(111, projection="3d")

    ax.plot3D(x1, y1, z1, "ob")

    ax.plot3D(x2, y2, z2,'or')

    ax.plot3D(xt1, yt1, zt1,'og')

    ax.plot3D(xt2, yt2, zt2,'oc')

    ax.set\_xlabel('mean b')

    ax.set\_ylabel('mean g')

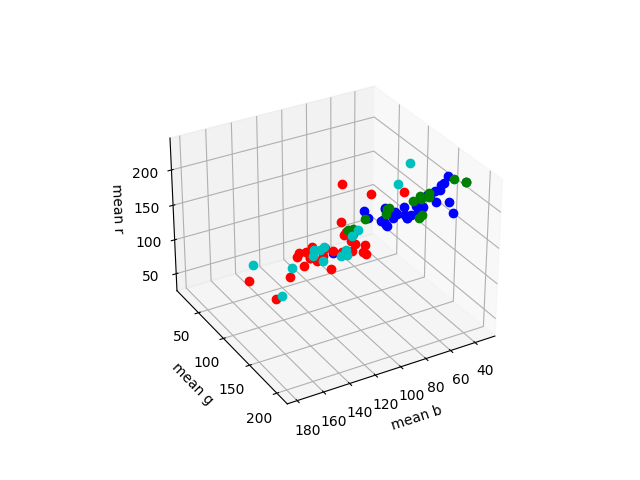
    ax.set\_zlabel('mean r')

    ax.view\_init(30, 60)

    plt.show()

رسم نمودار سه بعدی برای سه ویژگی اول(میانگین سه مولفه رنگی) برای مشاهده وضعیت داده ها.

نمودار:



رنگ آبی برای داده های آموزشی مربوط به موز.

رنگ قرمز برای داده های آموزشی مربوط به غیر موز.

رنگ سبز مربوط به داده های تست مربوط به موز.

رنگ فیروزه ای مربوط به داده ه ای تست مربوط به غیر موز.

def classification(x\_train, y\_train, x\_test, y\_true):

    clf = SVC(kernel="rbf")

    clf.fit(x\_train, y\_train)

    y\_pred = clf.predict(x\_test)

    print(f"accuracy: {accuracy\_score(y\_true, y\_pred)}")

    print(f"y\_pred: {y\_pred}")

    print(f"y\_true: {y\_true}")

طبقه بندی با استفاده از svm و سپس پیش بینی برچسب مربوط به داده های تست.

خروجی این تابع:

accuracy: 0.9666666666666667

y\_pred: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

y\_true: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

def main():

    y\_train = []

    y\_test = []

    dirr = "train/"

    active = True

*while* active:

        featurs = []

*for* i, entry in enumerate(os.scandir(dirr)):

            dir = entry.path

*for* j, entry2 in enumerate(os.scandir(dir)):

                image\_dir = entry2.path

                img\_in = cv2.imread(image\_dir)

                num = 0

*if* num == 1:

                    img\_out = histogram\_equalization(img\_in)

*elif* num == 2:

                    img\_out = sharpening(img\_in)

*elif* num == 3:

                    img\_out = histogram\_equalization(img\_in)

                    img\_out = sharpening(img\_out)

*else*:

                    img\_out = img\_in

                img\_out, mask = remove\_background(img\_out)

                featurs.append(feature\_extraction(img\_out, mask))

*if* dirr == "train/":

*if* i == 0:

                        y\_train.append(1)

*else*:

                        y\_train.append(0)

*else*:

*if* i == 0:

                        y\_test.append(1)

*else*:

                        y\_test.append(0)

*if* dirr == "train/":

            x\_train = featurs.copy()

            dirr = "test/"

*else*:

            x\_test = featurs.copy()

            active = False

    x\_train = np.array(x\_train)

    y\_train = np.array(y\_train)

    x\_test = np.array(x\_test)

    y\_test = np.array(y\_test)

    nemmodar(x\_train, x\_test)

    classification(x\_train, y\_train, x\_test, y\_test)

*if* \_\_name\_\_ == "\_\_main\_\_":

    main()

تابع main که به ترتیب عکس ها را از پوشه train و سپس test میخواند و پس از انجام پیش پردازش ها که در بالا توابع آنها توضیح داده شد عمل استخراج ویژگی انجام میشود و ویژگی های مربوط به train درون x\_train و ویژگی های مربوط به test در x\_test ذخیره میشود.

در انتها هم تابع classification فراخوانی میشود.