

به نام خدا
مهدی فیروزبخت
تمرین سری چهارم
درس بینایی کامپیوتر

در این تمرین لازم است تا مراحل ایجاد تصویر به کمک GLCM و K-MEANS را پیاده سازی کنیم. برای این کار ابتدا کتابخانه های مورد نیاز را وارد میکنیم:

```
!pip install tqdm-joblib
!python -m pip install -U scikit-image

import numpy as np
import cv2 as cv2
import matplotlib.pyplot as plt
from skimage.feature import graycomatrix, graycoprops
from google.colab.patches import cv2_imshow
from multiprocessing import Pool
import tqdm
from skimage.util.shape import view_as_windows
from joblib import Parallel, delayed
from tqdm_joblib import tqdm_joblib
from sklearn.cluster import KMeans
```

در مرحله بعد تصویر مورد نیاز را وارد کرده و آنرا کوچک میکنیم. برای این تمرین از ۳/۰ تصویر استفاده شده است. این کار را با تابع load_data انجام میدهیم. همچنین سطوح روشنایی را به ۲۵ درصد کاهش میدهیم :

```
def load_data(img):
    img = cv2.imread(img,1)
    img = cv2.resize(img, (0, 0), fx = 0.3, fy = 0.3)
    gray_image = cv2.cvtColor(img , cv2.COLOR_BGR2GRAY)
    scaled_gray_image = (rescale(gray_image, 1, anti_aliasing=False)*25).astype('uint8')
    return scaled_gray_image
```

در ادامه با استفاده از تابع show_hist مقدار هیستوگرام را نمایش میدهیم:

```
def show_hist(image):
    # computing the histogram of the blue channel of the image
    hist = cv2.calcHist([image],[0],[None],[256],[0,256])

    # plot the above computed histogram
    plt.plot(hist, color='b')
    plt.title('Image Histogram For Blue Channel GFG')
    plt.show()
```

در ادامه تابع GLCM را بررسی میکنیم .

در این تابع نیاز است ابتدا با استفاده از graycomatrix مقدار GLCM را محاسبه کرده سپس با استفاده از graycoprops ویژگی های مورد نیاز را استخراج کنیم. برای این تمرین ویژگی های 'dissimilarity', 'contrast', 'homogeneity', 'energy', 'correlation' استخراج میشود. همچنین برای همسایگی ها و درجات گوناگون نیز مقدار آن ها محاسبه میکنیم. کد آن به صورت زیر است. ورودی این تابع پچی است که بررسی ویژگی ها بر روی آن بررسی میشود. همچنین مقدار فاصله را به عنوان neigh و درجه را با عنوان angle دریافت کرده ایم. :

```
def glcm_props(patch , neigh , angle):
    lf = []
    props = ['dissimilarity', 'contrast', 'homogeneity', 'energy', 'correlation']
    glcm = graycomatrix(patch, neigh, angle, levels=26, symmetric=True, normed=True)

    for f in props:
        for item in graycoprops(glcm, f):
            lf.extend( item )

    return np.asarray(lf)
```

در ادامه تابع Compute_props را بررسی میکنیم. در این تابع مقدار های متفاوت پچ ، همسایگی ، درجات تعریف شده و از پایپلین برای محاسبه ویژگی های GLCM استفاده میشود.

```
def compute_props(scaled_gray_image):
    # number of processes
    nprocs = 10
    patch_sizes = [ 11 , 21 , 25 , 31 ]
    angles = [ 0 , np.pi/4 , np.pi/2, 3*np.pi/4]
    neighs = [ 3 , 5 , 7]

    results = []
    for size in patch_sizes:
        PATCH_SIZE = (size , size)
        patches = image.extract_patches_2d(scaled_gray_image, PATCH_SIZE)

        result = []
        title = "Calculation for patch size : " + str(size)
        with tqdm_joblib(desc=title , total=patches.shape[0]) as progress_bar:
            result.extend(Parallel(n_jobs=nprocs)(delayed(glcm_props)(patch,neighs,angles) for patch in patches))
        results.append(result)

    return results
```

در تابع compute_algo با استفاده از k-means و ویژگی های به دست آمده از مراحل قبل تصاویر را میسازیم. در این مرحله از نرمال سازی نیز استفاده شده است :

```
def compute_algo(img , k):
    patch_sizes = [ 10 , 20 , 24 , 30 ]

    img = load_data(img)
    w , h = img.shape[0] , img.shape[1]

    show_hist(img)

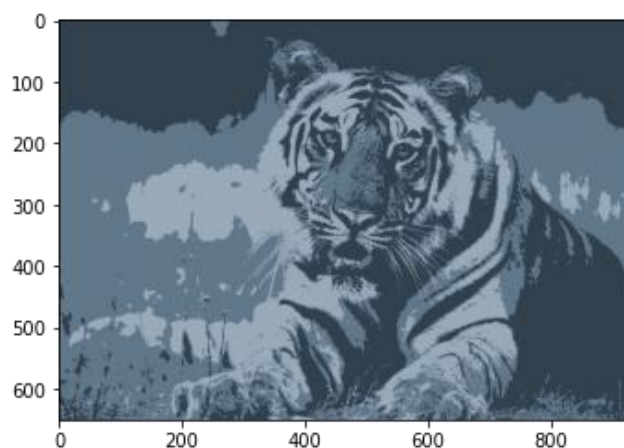
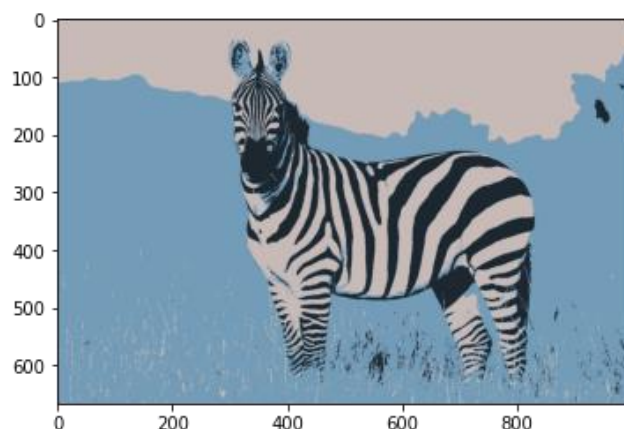
    results = np.asarray(compute_props(img))

    for i , item in enumerate(results):
        scaler = MinMaxScaler().fit_transform(item)
        kmeans = KMeans(n_clusters=k).fit(scaler)
        labels = kmeans.fit_predict(scaler).reshape(w-patch_sizes[i] , h-patch_sizes[i])
        plt.imshow(labels)
        plt.show()
```

۱. در استفاده از تابع k -means بدون در نظر گرفتن ویژگی ها ، تصاویری مانند تصویر گورخر اصلا تصاویر مناسبی ایجاد نمیشود . مراحل اجرا الگوریتم به صورت زیر میباشد :

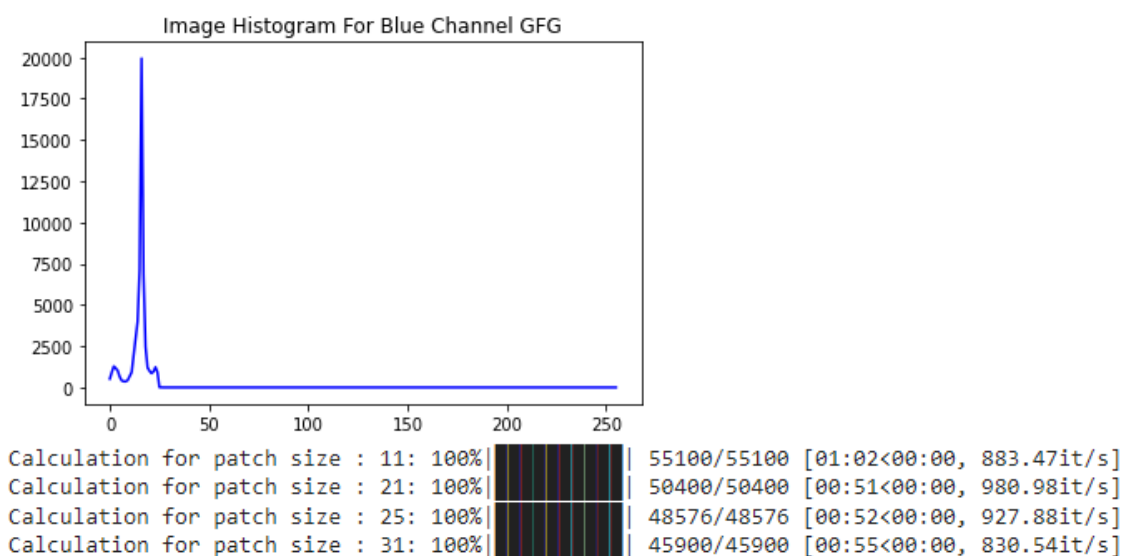
۱. تعداد خوشه های K را انتخاب کنید.
۲. در نقاط K تصادفی، مرکزها را انتخاب کنید.
۳. هر نقطه داده را به نزدیکترین مرکز \rightarrow که خوشه های K را تشکیل می دهد، اختصاص دهید.
۴. مرکز جدید هر خوشه را محاسبه و قرار دهید.
۵. هر نقطه داده را به نزدیکترین مرکز جدید اختصاص دهید. در صورت هرگونه جابجایی انجام شد، به مرحله ۴ بروید، در غیر این صورت، مدل آماده است.

در این حالت نقاط را بر اساس رنگ آنها تقسیم بندی میکند و نقاطی که از نظر رنگی شباهت دارند را به عنوان گروه در نظر میگیرند و به هیچ جزئیات و ویژگی دیگری توجه نمیکند. برای مثال در شکل گورخر تصویر گورخر، آسمان ، چمن ها هر کدام جزئیات خاص خود را دارند که با استفاده از k -means بدون جداسازی این ویژگی ها به هیچ عنوان قابل استفاده نمیشود. تصویر ایجاد شده از این الگوریتم به صورت زیر میباشد. قابل مشاهده است که تصویر گورخر را نتوانسته بخش بندی مناسب کند :

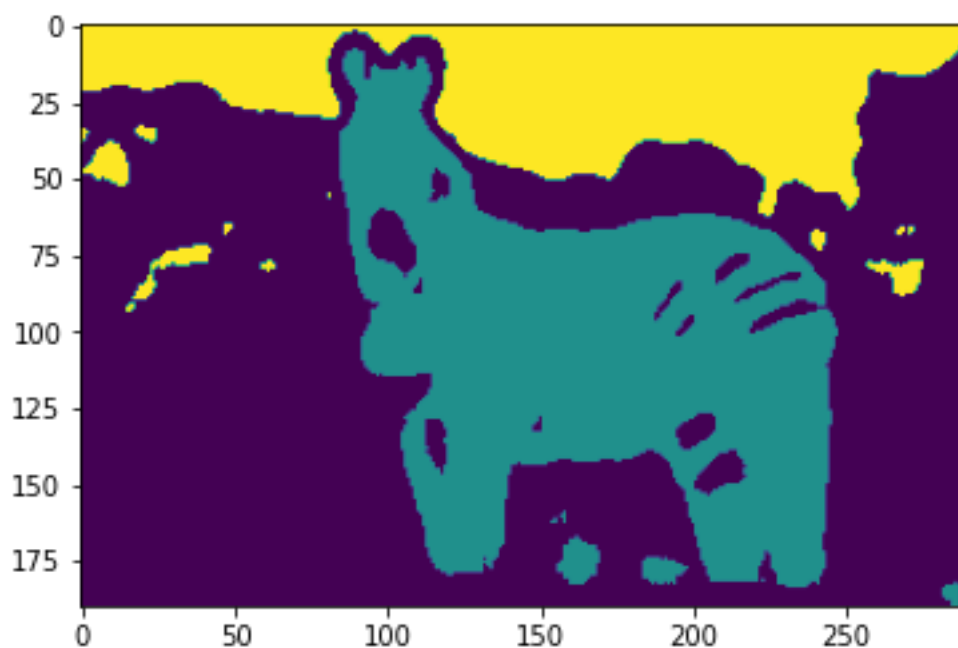


۲.

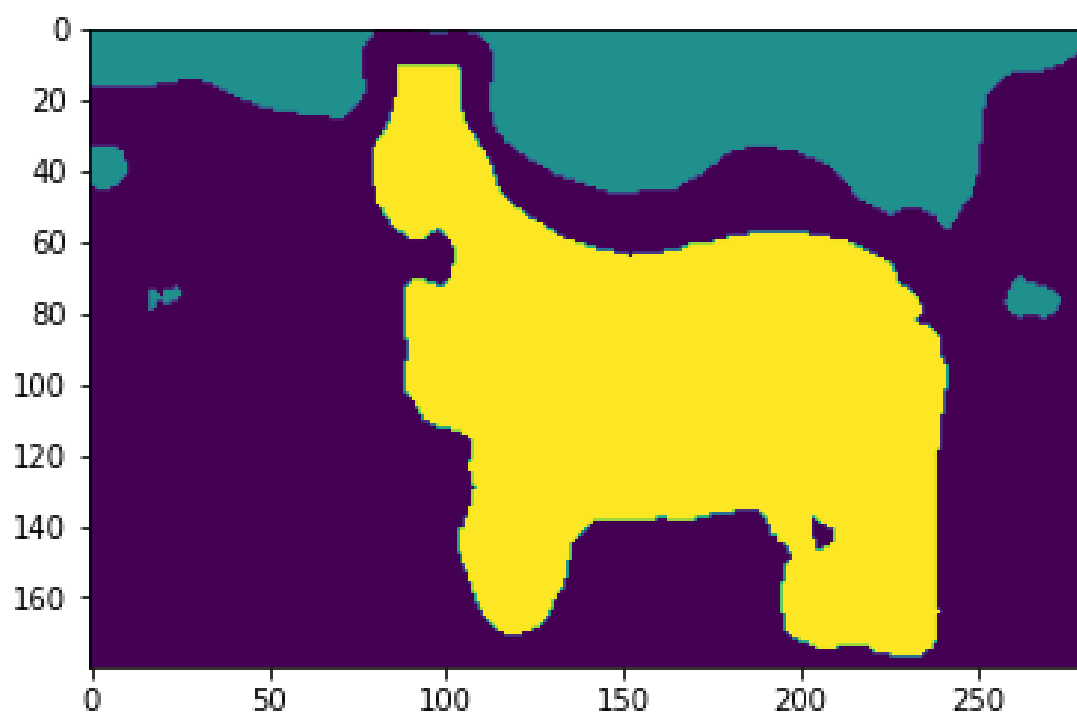
با توجه به کد های توضیح داده شده در قسمت اول ، تصویر گورخر را وارد الگوریتم میکنیم. نتیجه به دست آمده برای حالات مختلف به شرح زیر است :



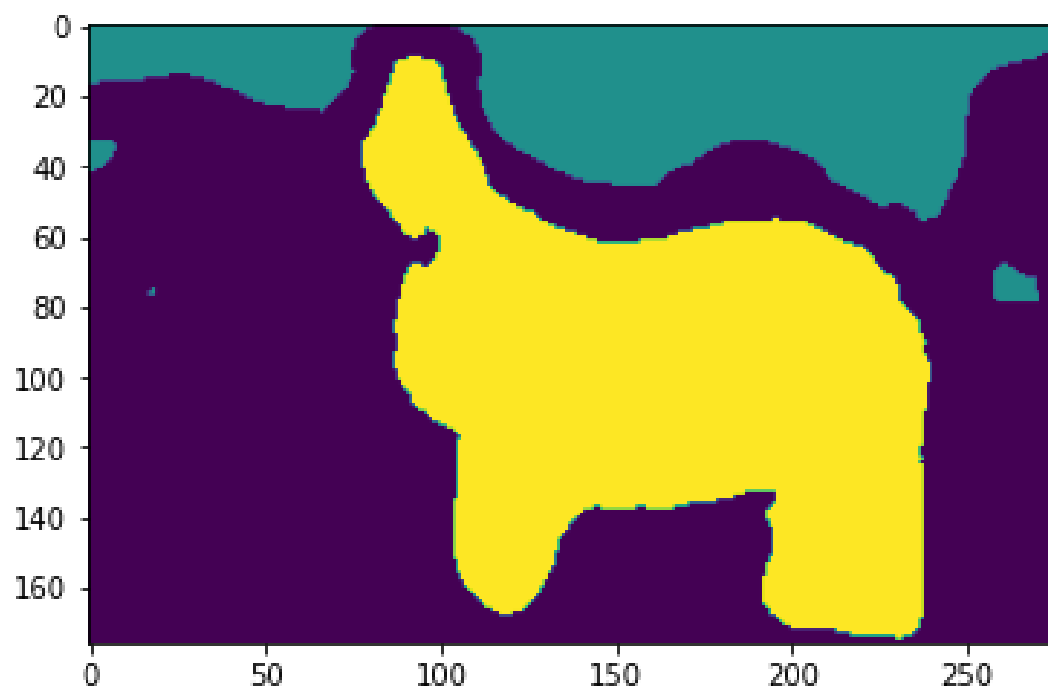
ابتدا برای حالت اندازه پچ ۱۱ :



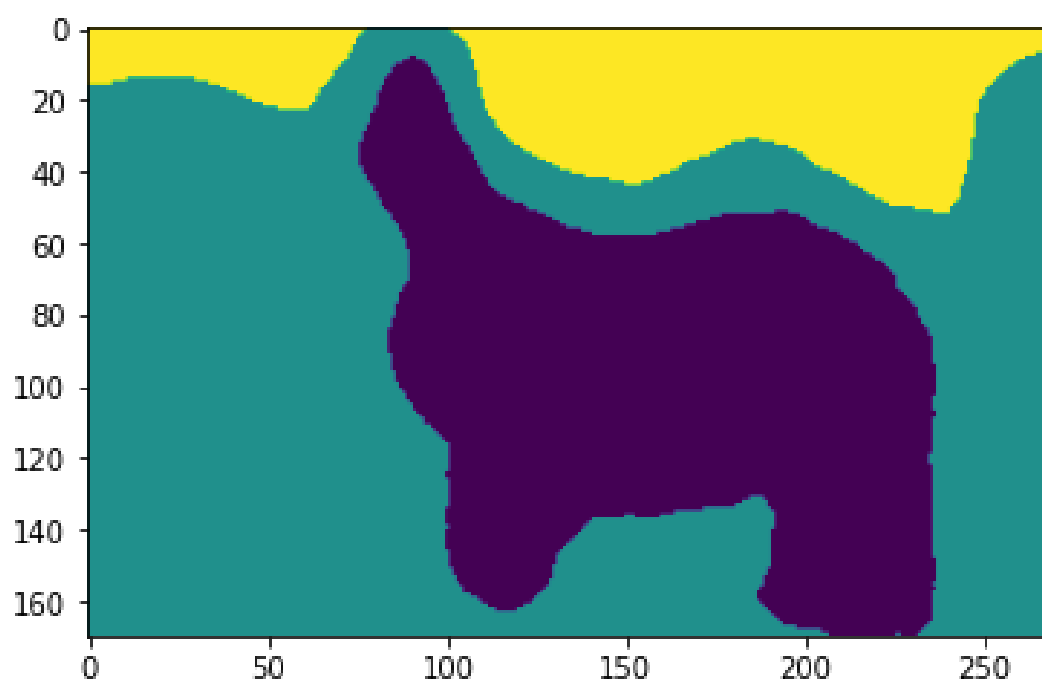
حالت اندازه پیچ ۲۱ :



حالت اندازه پیچ ۲۵ :

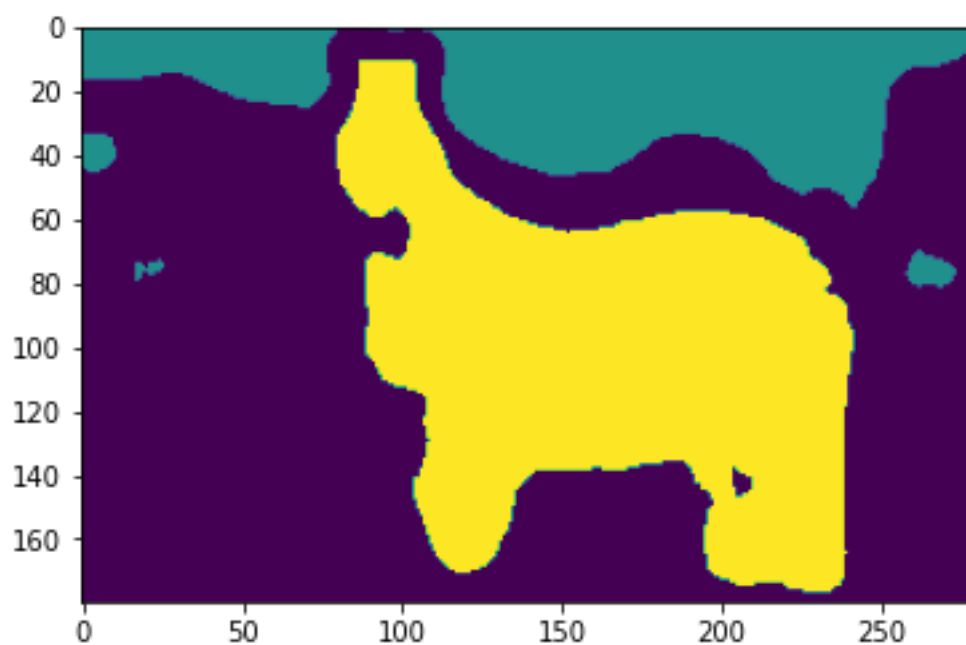


حالت اندازه پچ ۳۱ :

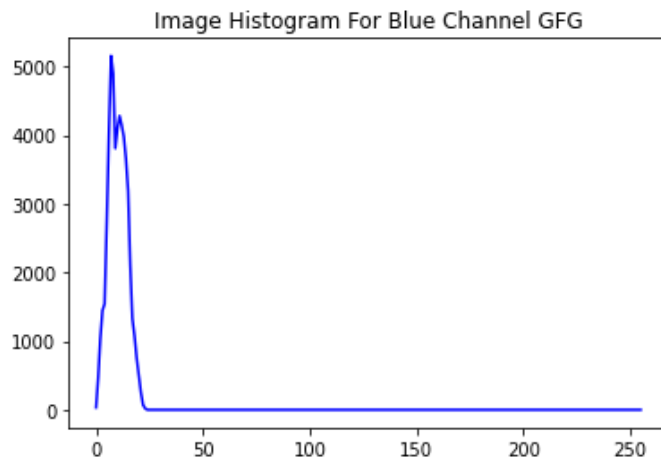


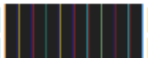
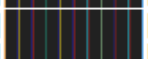


تمامی تصاویر درون فایل پیوست شده است.

در این مدل بهترین تصویر به دست آمده به حالت پچ است که به صورت زیر می باشد :

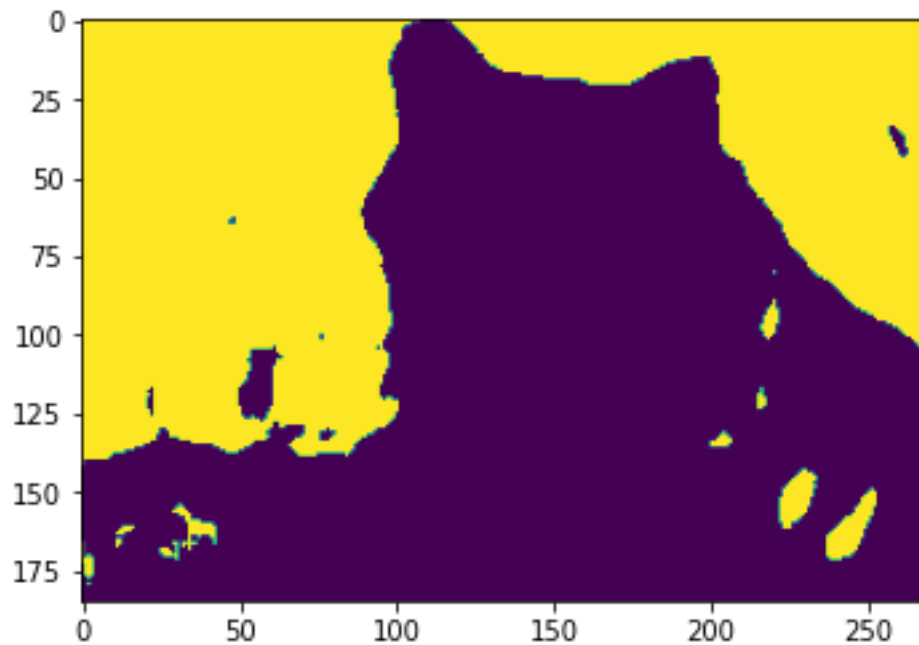


در ادامه تصویر ببر را وارد این الگوریتم میکنیم :

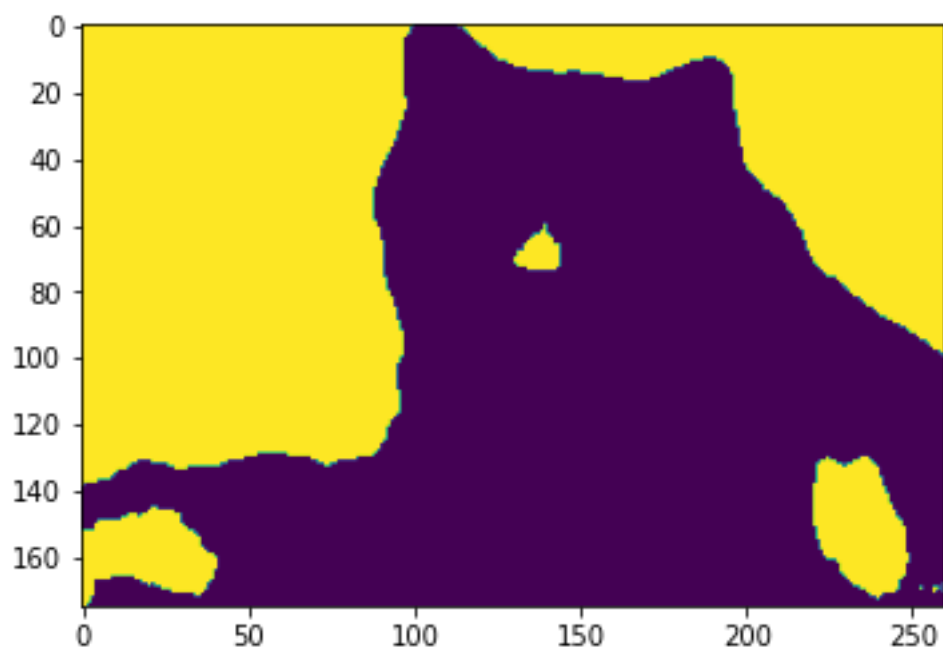


Calculation for patch size : 11: 100%		49950/49950 [00:52<00:00, 948.96it/s]
Calculation for patch size : 21: 100%		45500/45500 [00:49<00:00, 912.15it/s]
Calculation for patch size : 25: 100%		43776/43776 [00:45<00:00, 966.31it/s]
Calculation for patch size : 31: 100%		41250/41250 [00:42<00:00, 965.53it/s]

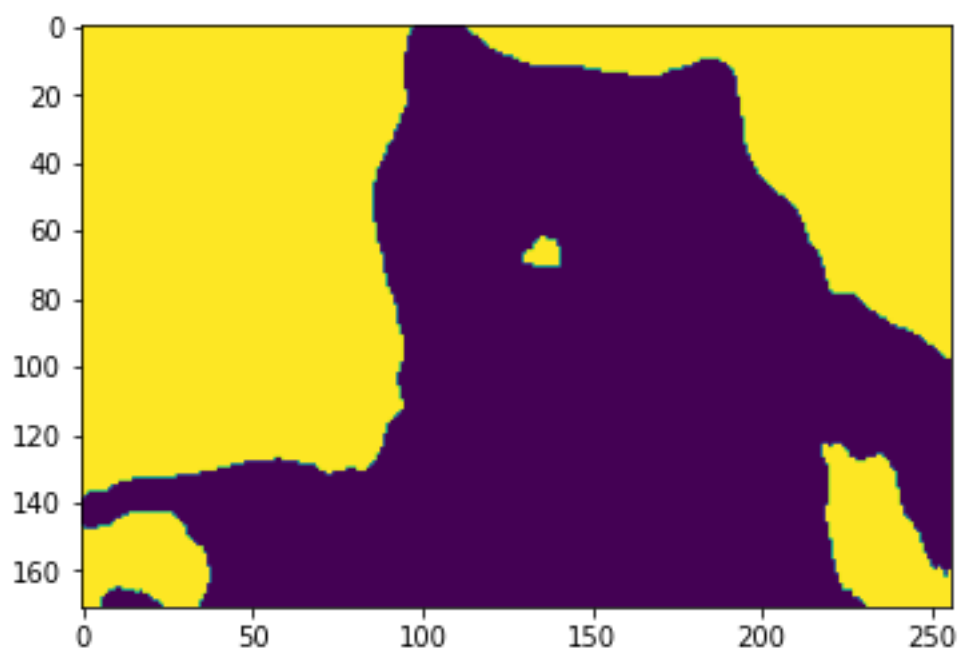
ابتدا برای پچ ۱۱ :



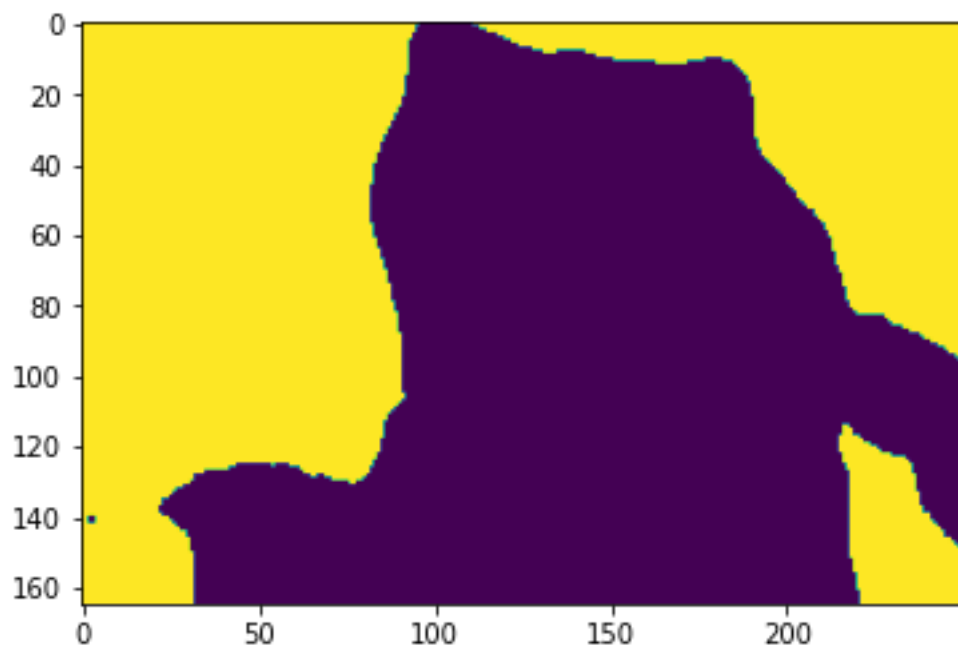
پج ۲۱ :



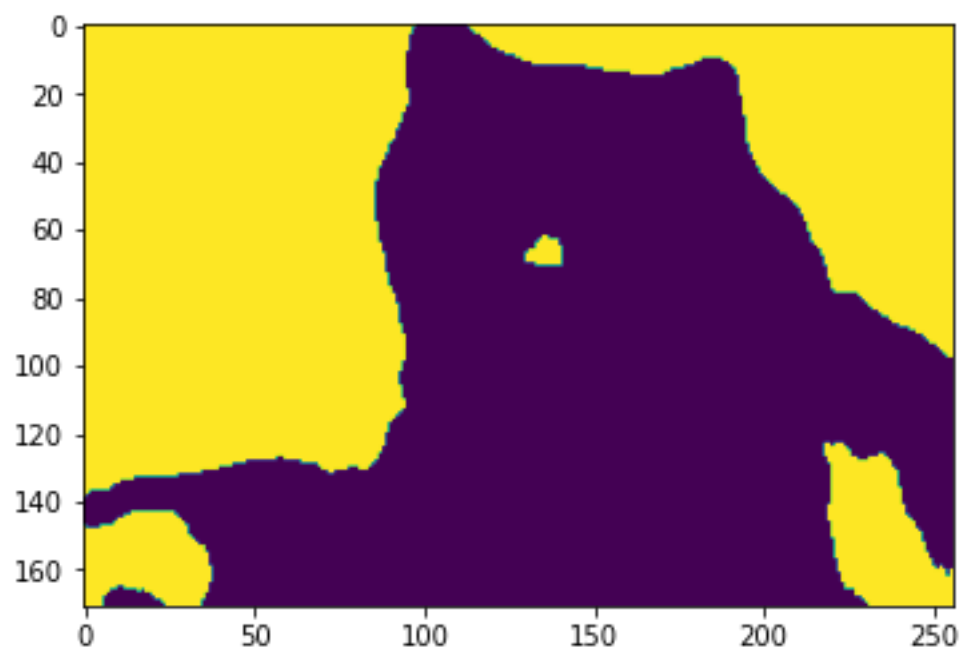
پج ۲۵ :



پج ۳۱ :



بهترین تصویر ایجاد شده در این بخش بندی به صورت زیر است :



برای استفاده از فیلتر گابور نیز ابتدا کتابخانه های مورد نظر را بارگذاری میکنیم :

```
import numpy as np
import cv2 as cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
from sklearn.cluster import KMeans

from skimage.filters import gabor_kernel

import pandas as pd
from sklearn import preprocessing

import warnings
warnings.filterwarnings("ignore")
```

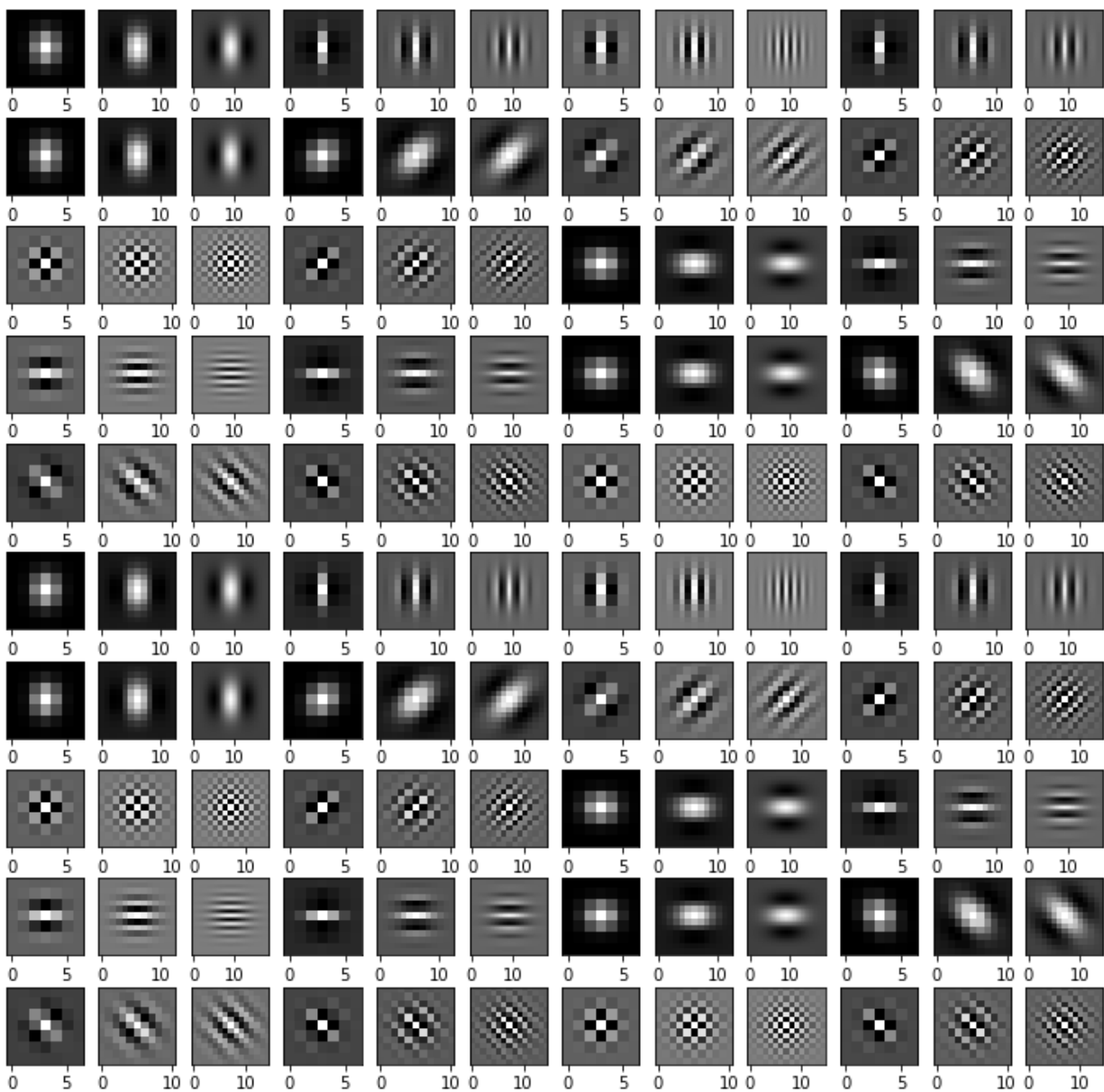
سپس تابع load_data که تصویر را خوانده و به سطح خاکستری میبرد :

```
def load_data(image):
    image = cv2.imread(image,1)
    gray_image = cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
    plt.imshow(gray_image , cmap='gray')
    return gray_image
```

سپس تابع ساخت فیلترهای گابور را ایجاد میکنیم. در این تابع برای فیلتر از تتا های 0 تا 2π با فواصل $\pi/4$ استفاده شده است. همچنین از فرکانس $(0.1, 0.3, 0.5, 0.7, 0.9)$ استفاده شده و از گاما با مقادیر 1 و 2 و 3 استفاده شده است.

```
def build_filters():
    kernels = [] #Create empty list to hold all kernels that we will generate in a loop
    for theta in range(0, 8):
        theta = theta / 4. * np.pi
        for frequency in (0.1, 0.3, 0.5, 0.7, 0.9):
            for sigma in (1, 2, 3):
                kernel = gabor_kernel(frequency, theta=theta , sigma_x=sigma , sigma_y=sigma)
                kernels.append(np.real(kernel))
    return kernels
```

شکل کرنل های ساخته شده به صورت زیر میباشد :



در ادامه لازم است از این کرنل های ساخته شده استفاده شود پس از تابع `compute_kernels` استفاده میشود. درون این تابع ابتدا تصویر به صورت `flatten` در می آید. سپس برای هر پیکسل فیلتر را اعمال میکنیم و نتیجه را درون یک دیتافریم قرار میدهیم.

```
def compute_kernels(img):

    #Save original image pixels into a data frame. This is our Feature #1.
    img2 = img.reshape(-1)
    df = pd.DataFrame()
    df['Original Image'] = img2

    #Generate Gabor features
    num = 1 #To count numbers up in order to give Gabor features a label in the data frame
    filters = build_filters()
    for filter in filters:
        #Now filter the image and add values to a new column
        fimg = cv2.filter2D(img2, cv2.CV_8UC3, filter)
        filtered_img = fimg.reshape(-1)

        df[num] = filtered_img #Labels columns as Gabor1, Gabor2, etc.
        num += 1 #Increment for gabor column label
    return df , num
```

در مرحله بعد نیاز است تا با استفاده از ویژگی های به دست و K-means یک بخش بندی تصویر ایجاد کنیم اما ابتدا لازم است از یک نرمال سازی استفاده کنیم. برای همین از تابع Clustering استفاده میکنیم :

```
def Clustering(df , num , k ,w , h):
    data = df.loc[:, 1:num-1].values

    X_scaled = preprocessing.MinMaxScaler().fit_transform(data)

    kmeans = KMeans(n_clusters=3).fit(X_scaled)
    label = kmeans.fit_predict(X_scaled)
    label = label.reshape(w, h)
    plt.imshow(label)
```

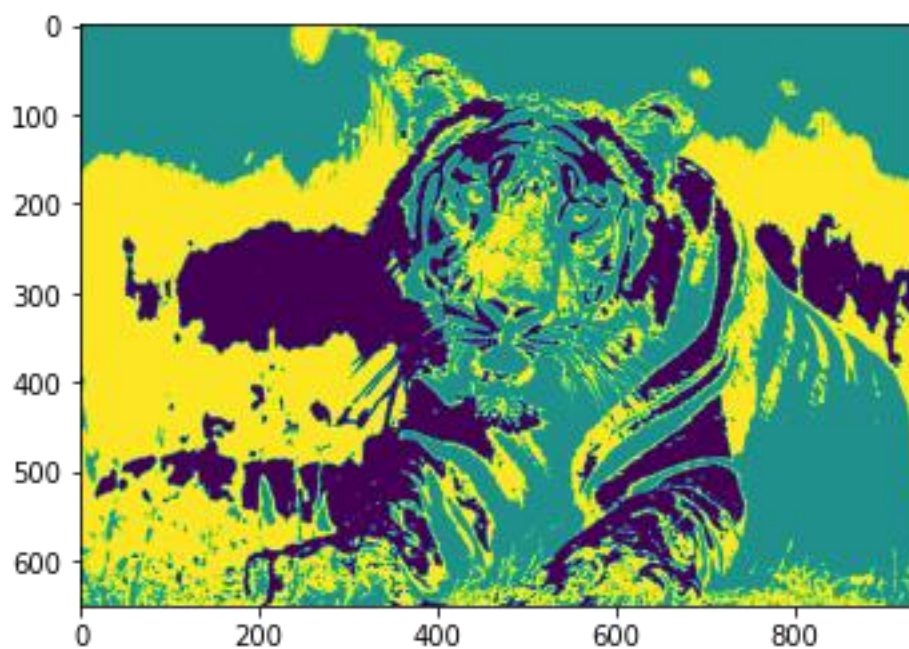
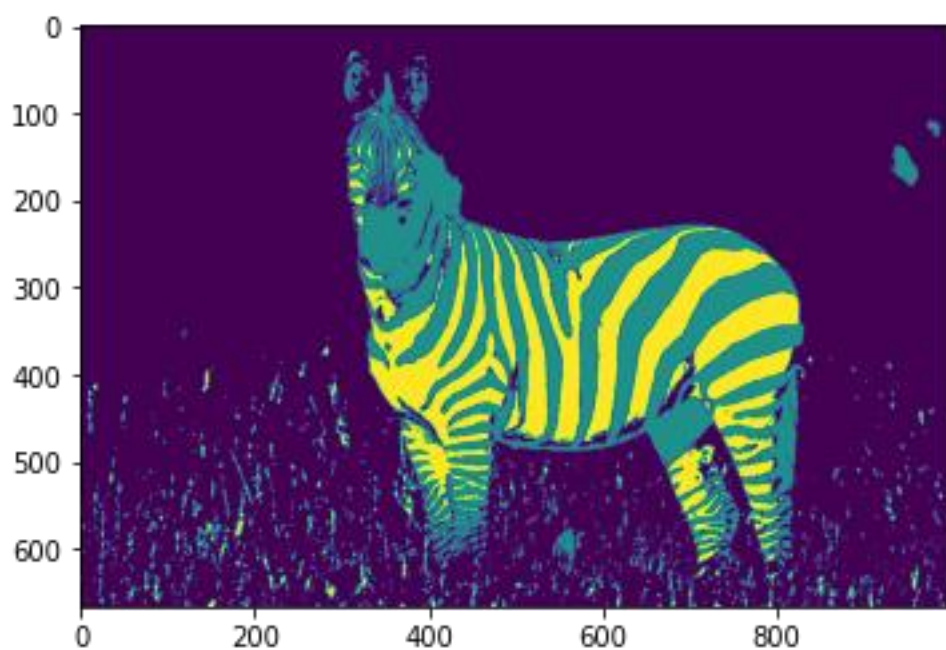
در ادامه لازم است تا مراحل را به ترتیب انجام شود که از تابع use_gabor استفاده میکنیم :

```
def use_gabor(img , k):
    img = load_data(img)
    w , h = img.shape[0] , img.shape[1]

    df , num = compute_kernels(img)

    Clustering(df , num , k , w , h)
```

نتیجه به دست آمده برای ۲ تصویر گورخر و ببر به صورت زیر می باشد :



۴. در الگوریتم GLCM از یک بچ برای هر پیکسل استفاده میکنم که این کار باعث میشود ویژگی های پیکسل نسبت به ویژگی های همسایه آن بررسی شود و ویژگی هایی به دست می آید که حالت مکانی هر پیکسل در نظر گرفته شده است. در این حالت پیکسل ها به حالت مطلوبی میتوانند دسته بندی شوند. اما در مقابل روش فیلتر گابور از ویژگی های محلی و همسایه های هر پیکسل استفاده نمیکند و ماهیت های محلی مانند وابستگی ، انرژی و... را نادیده گرفته و نمیتواند نتیجه مطلوبی ایجاد کند.

در این مسئله روش اول یعنی الگوریتم GLCM نتیجه بسیار مطلوب تری نسبت به روش دوم ایجاد کرده که با توجه به توضیحات بالا علت نتیجه بهتر استفاده از ماهیت های محلی است.

روش اول به علت محاسبه هر ماهیت سرعت نسبتاً کندتری از روش دوم دارد که با استفاده از موازی سازی و پایپلین میتوانیم تا حد مطلوبی این سرعت را بهبود ببخشیم.

با تشکر فراوان

مهدی فیروزبخت