

Amirkabir University of Technology
(Tehran Polytechnic)

تمرین شماره ۵

اعضای گروه :

عرفان قصری ۹۹۲۳۰۶۱

سید علی عبداللهیان ۹۹۲۳۱۱۷

استاد درس:

دکتر شریفیان

بهار ۱۴۰۳

سرفصل مطلب

2.....	۱. سوال
9.....	۲. سوال
17.....	۳. سوال

در این تمرین از کد هایی که در اختیار ما قرار گرفته برای نمایش تصویر استفاده کرده ایم:

```
# Our Setup, Import Libraries, Create our Imshow Function and
Download our Images
import cv2
import numpy as np
from matplotlib import pyplot as plt

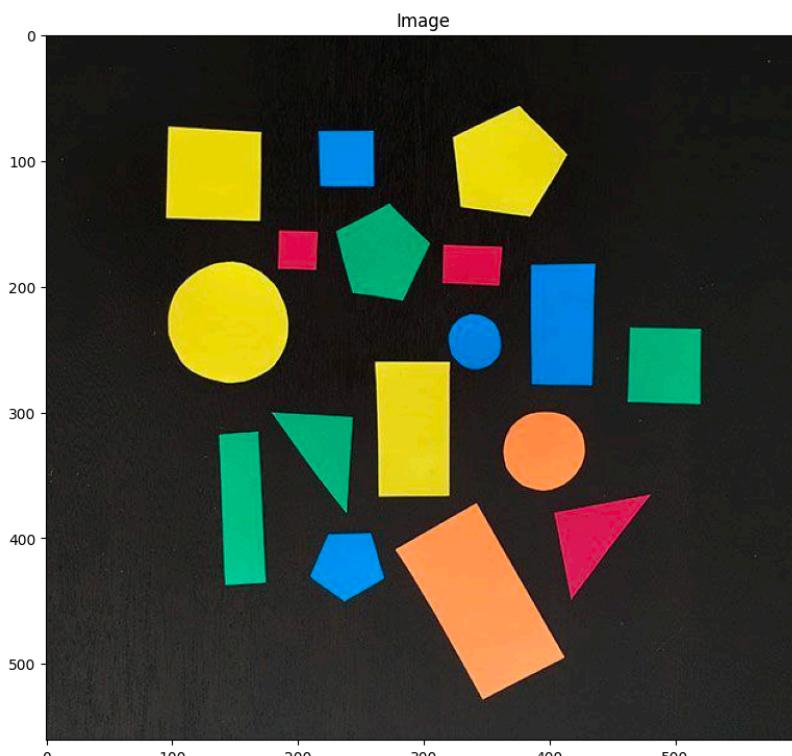
# Define our imshow function
def imshow(title = "Image", image = None, size = 10):
    w, h = image.shape[0], image.shape[1]
    aspect_ratio = w/h
    plt.figure(figsize=(size * aspect_ratio, size))
    plt.imshow(cv2.cvtColor(image,
cv2.COLOR_BGR2RGB).astype('float32') / 255)
    plt.title(title)
    plt.show()
```

حال تصویر را نمایش می دهیم:

```
# reading Shapes.jpg
image = cv2.imread("Shapes.jpg")

# Display the image
imshow(image = image)
```

۲

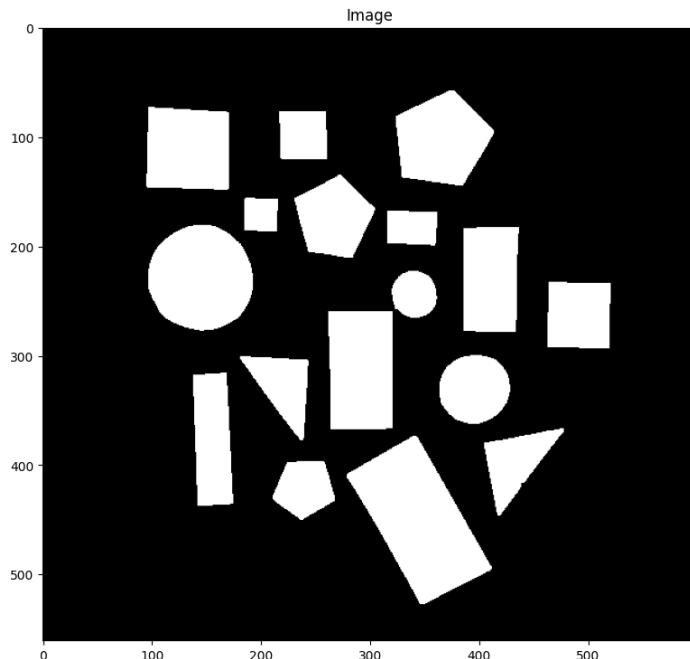


در ابتدا تصویر را سیاه و سفید می کنیم و روی آن ترشهد میزدیم، سپس با مرفو لوژی اوپن سعی در حذف نویز و با کلوز سعی در اتصال بخش های حذف شده از تصویر می کنیم:

```
# threshold image
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
_, thresh = cv2.threshold(gray, 80, 255, cv2.THRESH_BINARY)

# opening morphology
kernel = np.ones((3,3), np.uint8)
thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)
#closing morphology
thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel,
iterations=2)

# Display the image
imshow(image = thresh)
```

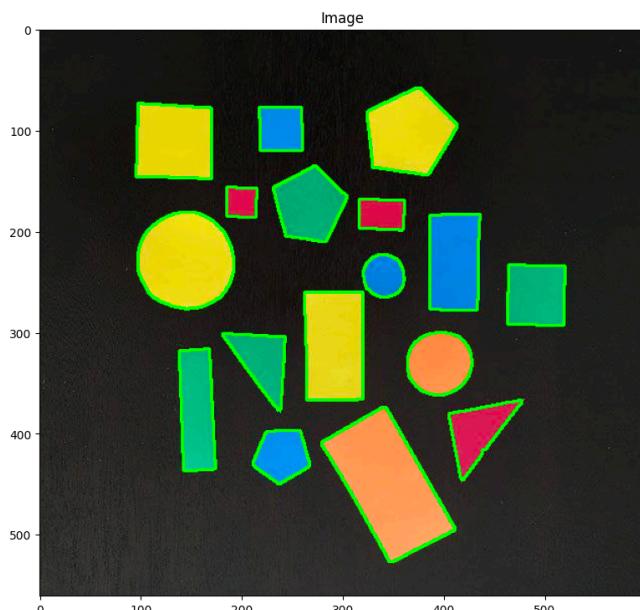


سپس کانتور ها را پیدا کرده و رسم می کنیم:

```
# Find the contours in the image
contours, _ = cv2.findContours(thresh, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)

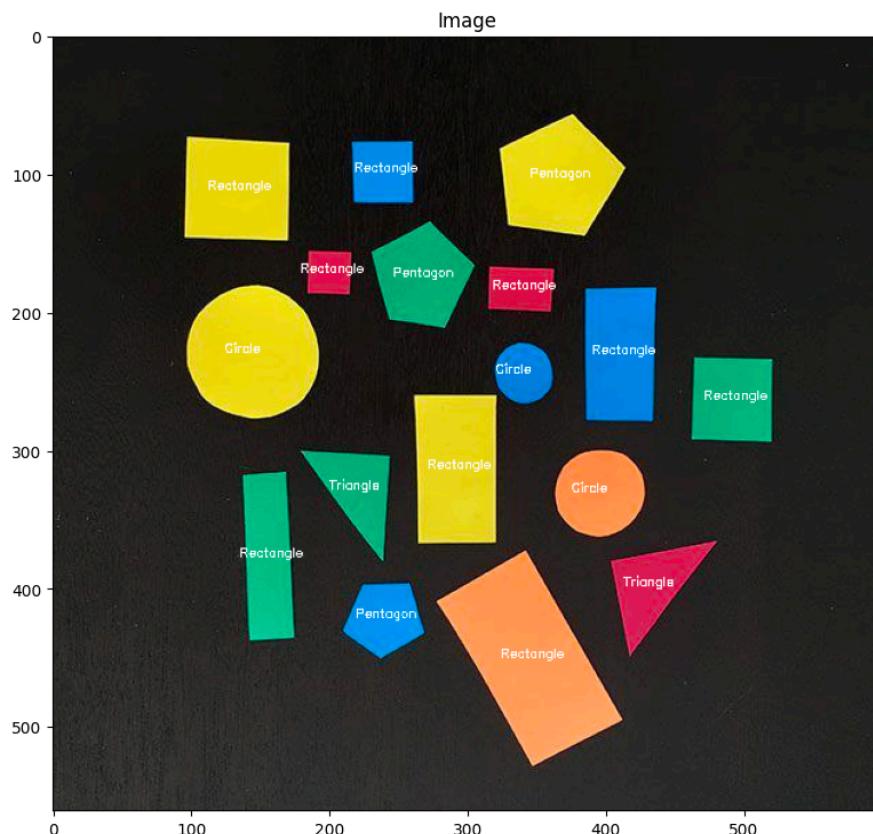
# Draw the contours on the image
cntimage = image.copy()
cv2.drawContours(cntimage, contours, -1, (0, 255, 0), 2)

# Display the image
imshow(image = cntimage)
```



پس از این، با الگوریتم داگلاس پیوکر تعداد گوشه های تصویر را پیدا می کنیم و سپس نام شکل را در مرکز آن می نویسیم:

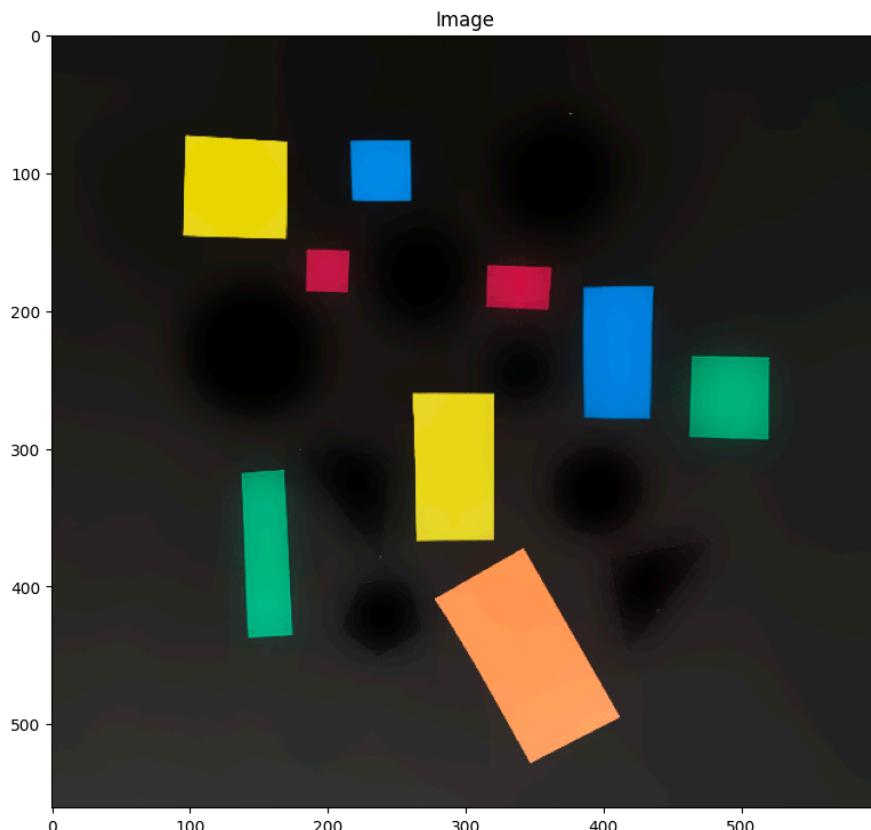
```
findimage = image.copy()
for cnt in contours:
    approx = cv2.approxPolyDP(cnt, 0.02*cv2.arcLength(cnt,
True), True)
    # print (len(approx))
    if len(approx) == 3:
        shape = "Triangle"
    elif len(approx) == 4:
        shape = "Rectangle"
    elif len(approx) == 5:
        shape = "Pentagon"
    else:
        shape = "Circle"
    # Draw the shape name center of the image
    M = cv2.moments(cnt)
    cX = int(M["m10"] / M["m00"])
    cY = int(M["m01"] / M["m00"])
    cv2.putText(findimage, shape, (cX-20, cY),
cv2.FONT_HERSHEY_SIMPLEX, 0.3, (255, 255, 255), 1)
```



برای بخش دیگر سوال، ابتدا بر روی تصاویر غیر چهار ضلعی یک حجم سیاه رسم می کنیم:

```
nonRect = []
for cnt in contours:
    approx = cv2.approxPolyDP(cnt, 0.02*cv2.arcLength(cnt,
True), True)
    # print (len(approx))
    if len(approx) != 4:
        nonRect.append(cnt)
# draw black filled counter in image to remove non rectangle
shapes a little bigger than counter
rectimage = image.copy()

cv2.drawContours(rectimage, nonRect, -1, (0, 0, 0), -1)
#bilateralFilter to remove noise
rectimage = cv2.bilateralFilter(rectimage, 50, 130, 130)
imshow(image = rectimage)
```

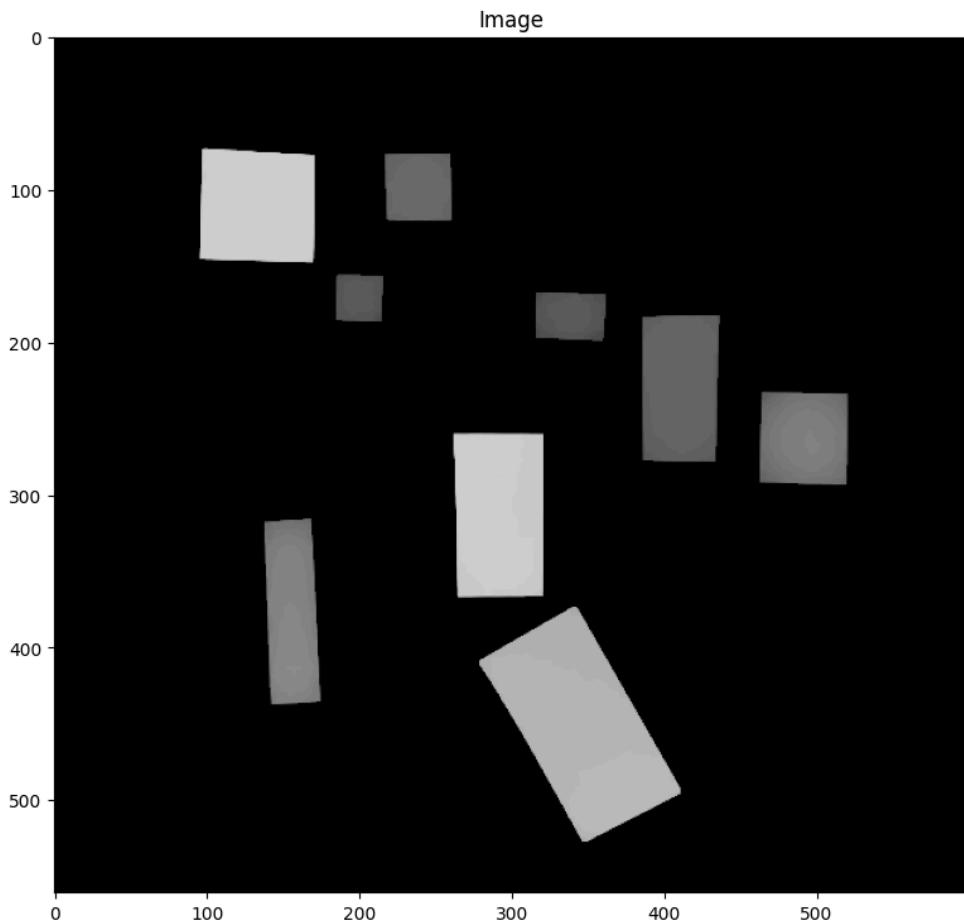


برای بهبود تصویر آن را به مقیاس YCrCb می بریم و بخش Y آن را جدا می کنیم و روی آن بخش جدا شده ، اوپنینگ و کلوزینگ اعمال می کنیم تا نویز آن گرفته شود.

```
# decompose image to YCrCb
YCrCb = cv2.cvtColor(rectimage, cv2.COLOR_BGR2YCrCb)
Y, Cr, Cb = cv2.split(YCrCb)

# threshold Y channel(Grayscale)
Ythresh = np.where(Y > 60, Y, 0)

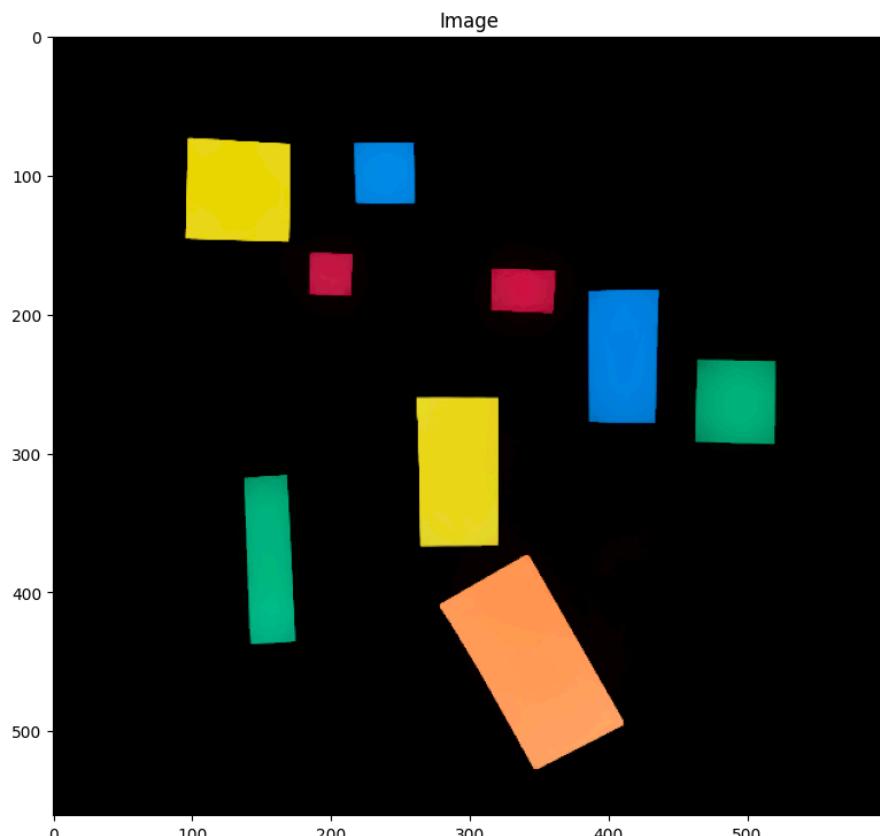
# opening morphology
kernel = np.ones((3,3), np.uint8)
Ythresh = cv2.morphologyEx(Ythresh, cv2.MORPH_OPEN, kernel)
#closing morphology
Ythresh = cv2.morphologyEx(Ythresh, cv2.MORPH_CLOSE, kernel,
iterations=2)
imshow(image = Ythresh)
```



سپس بخش Y را با دیگر بخش های دیگر تصویر در کنار هم قرار می دهیم:

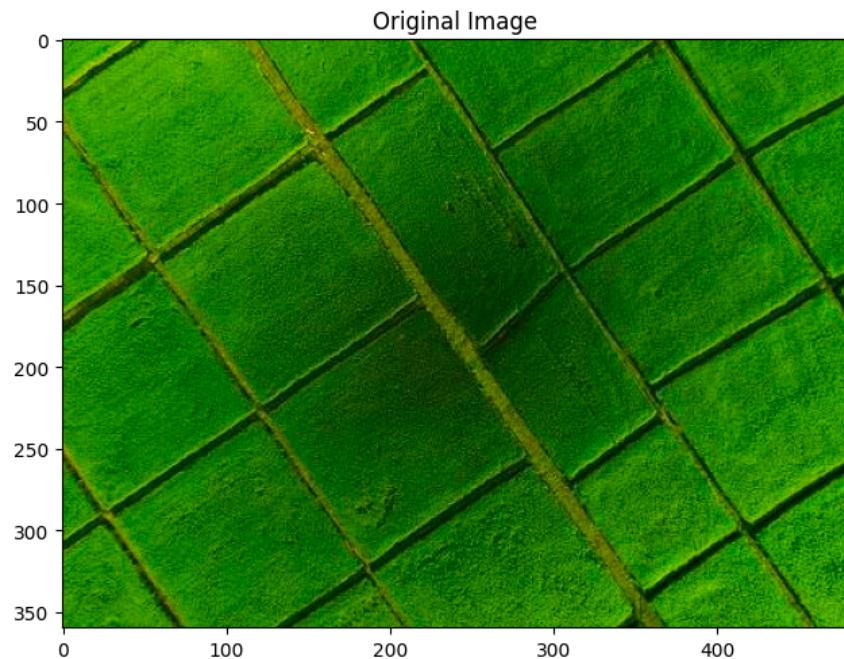
```
# rebuild image
YCrCb = cv2.merge((Ythresh, Cr, Cb))
image = cv2.cvtColor(YCrCb, cv2.COLOR_YCrCb2BGR)

imshow(image = image)
```



ابتدا تصویر را نمایش می دهیم:

```
# Read the image
img = cv2.imread("FARMS_2.jpeg")
image = img.copy()
imshow("Original Image", img)
```



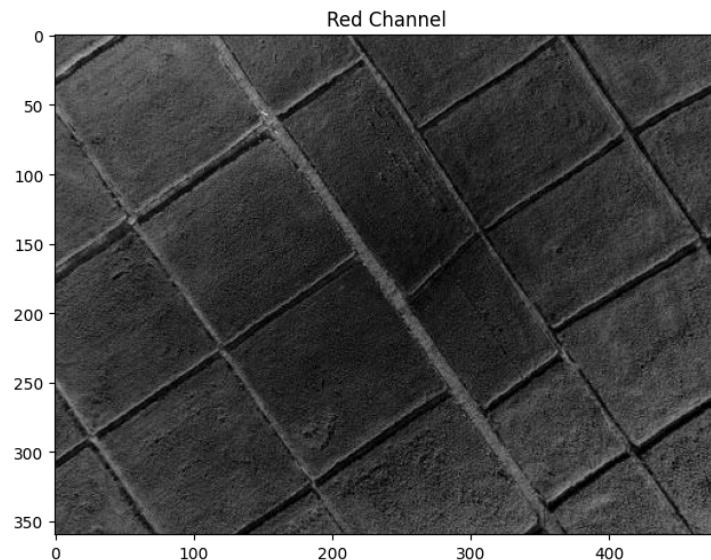
سپس تابع آتو کنی را تعریف می کنیم:

```
def autoCanny(image):
    # Finds optimal thresholds based on median image pixel
    intensity
    med_val = np.median(image)
    lower = int(max(0, 0.6 * med_val))
    upper = int(min(255, 1.3 * med_val))
    edges = cv2.Canny(image=image, threshold1=lower,
                      threshold2=upper)
    return edges
```

ابتدا کانال قرمز را جدا می کنیم و تنها با آن کار می کنیم:

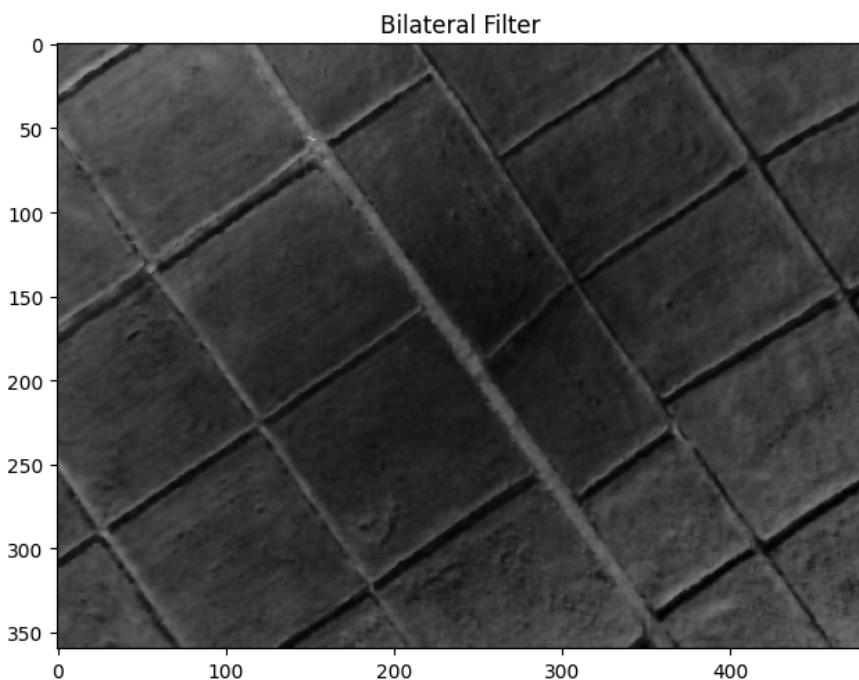
```
# Pick the red channel of the image instead of grayscale
imageB, imageG, imageR = cv2.split(img)
gray = imageR

# Display the image
imshow("Red Channel", gray)
```



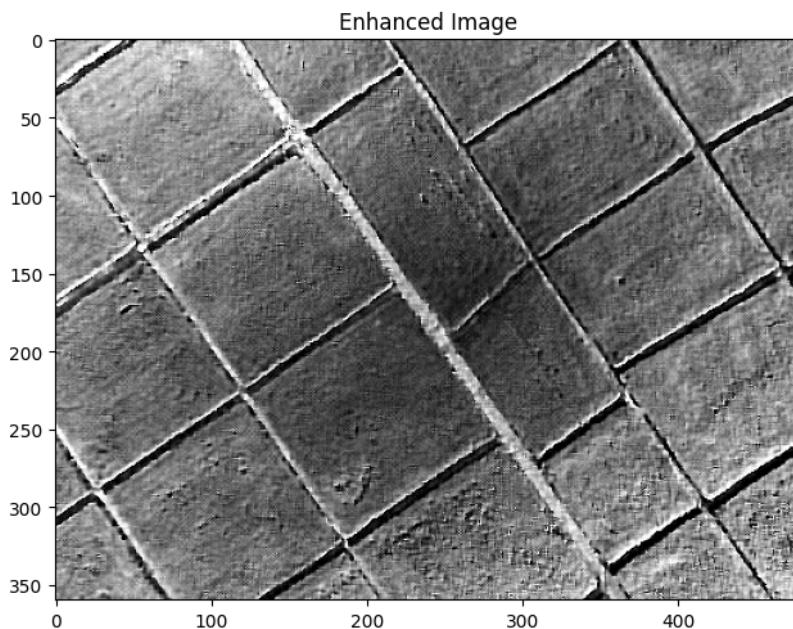
سپس روی آن فیلتر بای لترال اعمال می کنیم:

```
# bilateralFilter to remove noise
blt = cv2.bilateralFilter(gray, 5, 45, 45)
imshow("Bilateral Filter", blt)
```



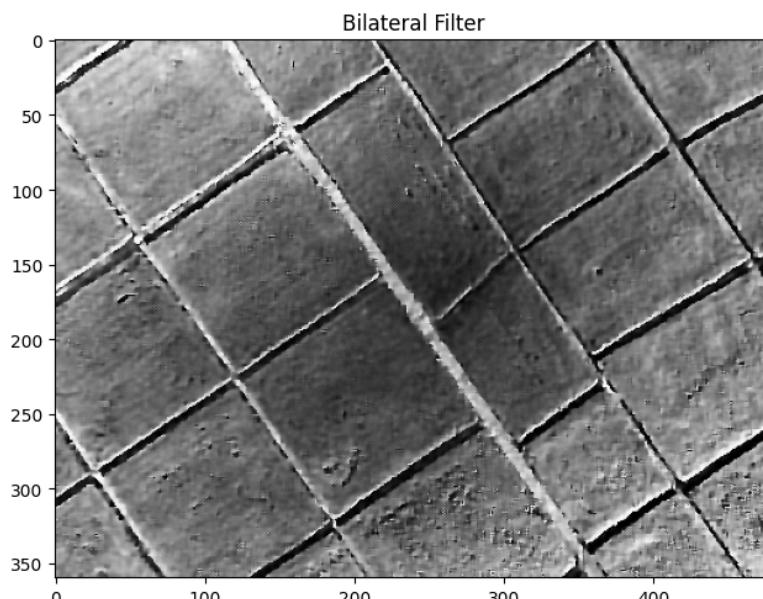
سپس تصویر را شارپ می کنیم:

```
# Increasing Contrast  
sharpening_kernel = np.array([[-1, -1, -1], [-1, 10, -1], [-1, -1, -1]])  
# Apply the sharpening filter  
Enhanced = cv2.filter2D(blt, -1, sharpening_kernel)  
  
# Display the image  
imshow("Enhanced Image", Enhanced)
```



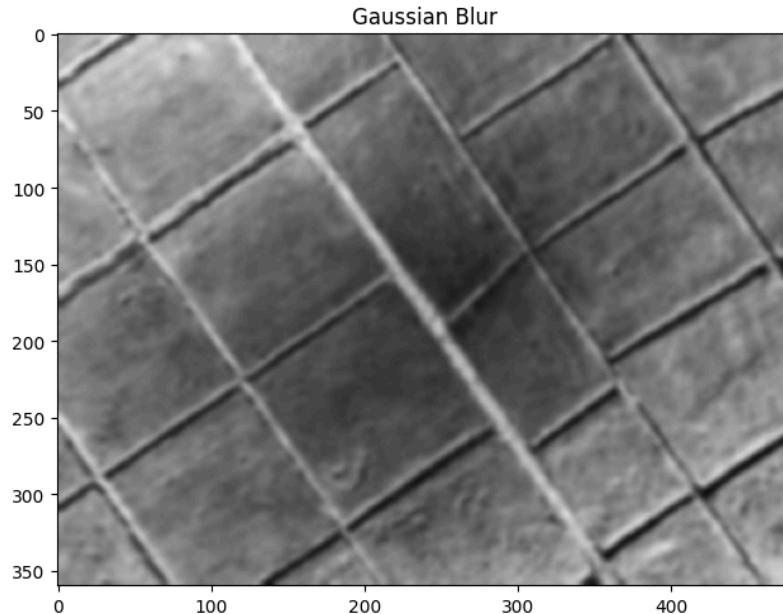
بار دیگر فیلتر بای لترال اعمال می کنیم:

```
# bilateralFilter to remove noise  
blt = cv2.bilateralFilter(Enhanced, 5, 45, 45)  
imshow("Bilateral Filter", blt)
```



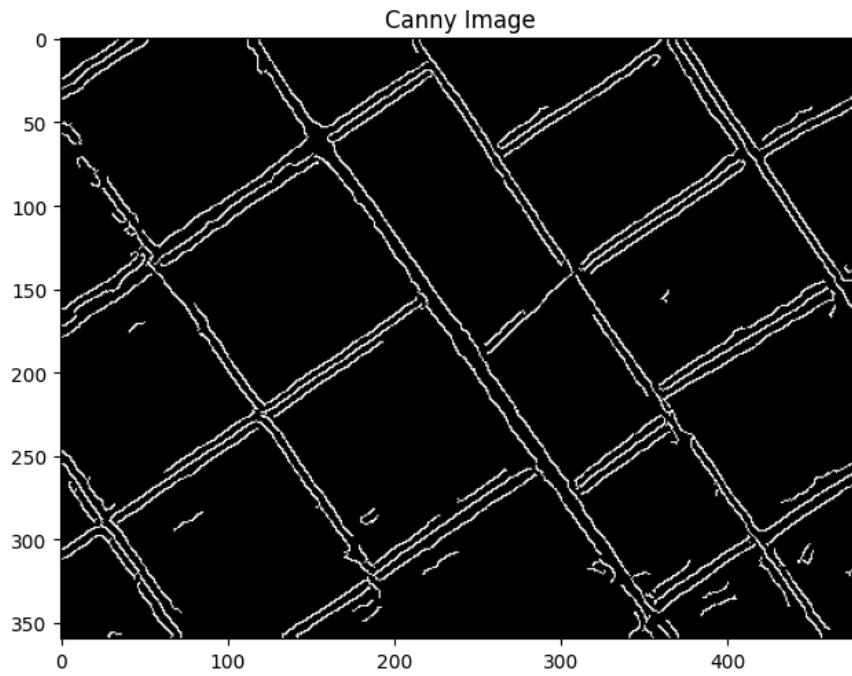
سپس گوسین بلور اعمال می کنیم:

```
# Blur the image
blur = cv2.GaussianBlur(blt, (11, 11), 0)
# Display the image
imshow("Gaussian Blur", blur)
```



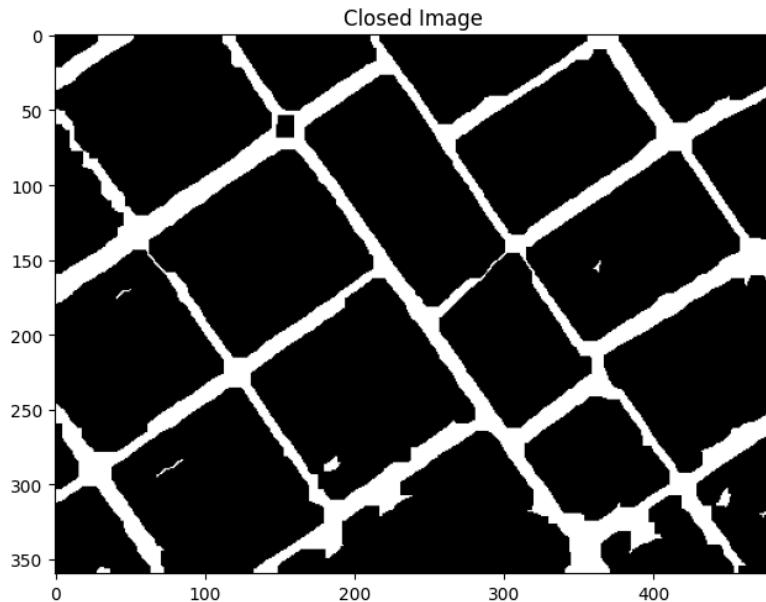
سپس آتو کنی را اعمال می کنیم:

```
# Canny image
canny_image = autoCanny(blur)
# Display the image
imshow("Canny Image", canny_image)
```



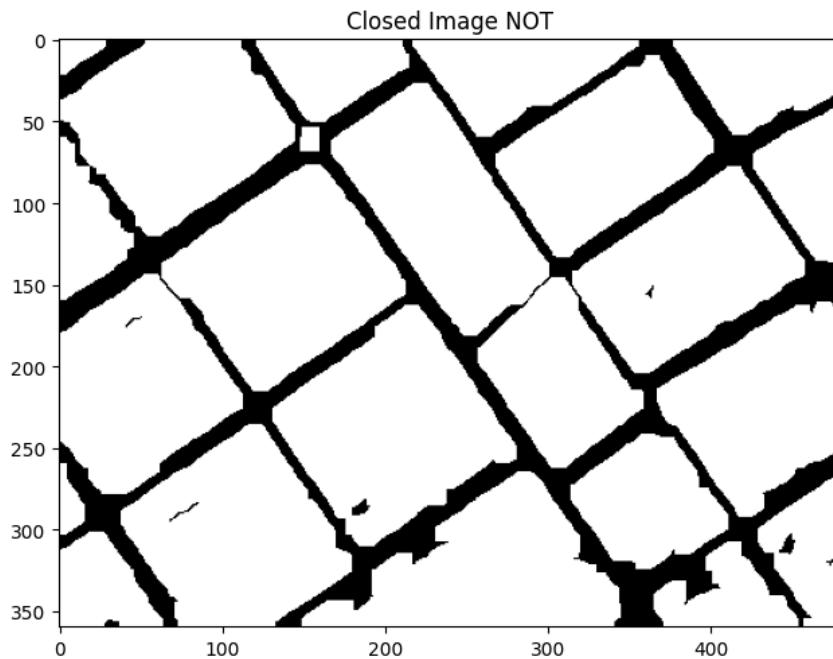
سپس کلوزینگ اعمال می کنیم:

```
Closed_image = cv2.morphologyEx(canny_image, cv2.MORPH_CLOSE,  
np.ones((9, 9), np.uint8), iterations=1)  
# Display the image  
imshow("Closed Image", Closed_image)
```



تصویر را بیت وایز نات می کنیم تا مزرعه ها سفید باشند نه مرز ها:

```
# Bitwise NOT -> the background should be black  
Closed_image_Not = cv2.bitwise_not(Closed_image)  
  
# Display the image  
imshow("Closed Image NOT", Closed_image_Not)
```



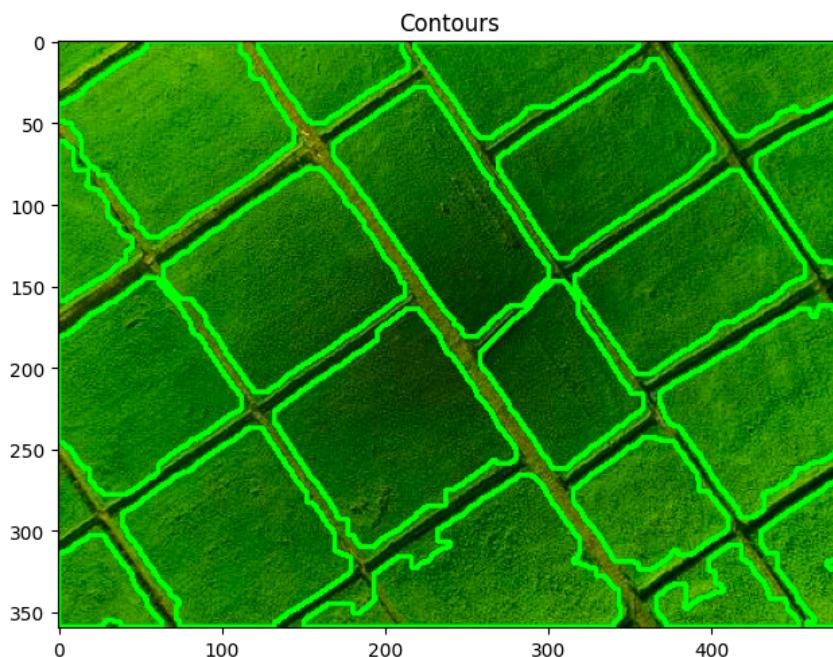
پس از یک بار ارود کردن تصویر ، کانتور ها را رسم می کنیم و از کانتور ها با ابعاد کوچک صرف نظر می کنیم:

```
kernel = np.ones((2,2),np.uint8)
Closed_image_Not = cv2.erode(Closed_image_Not,kernel,iterations
= 1)

# extracting contours
contours, hierarchy2 = cv2.findContours(Closed_image_Not,
cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)

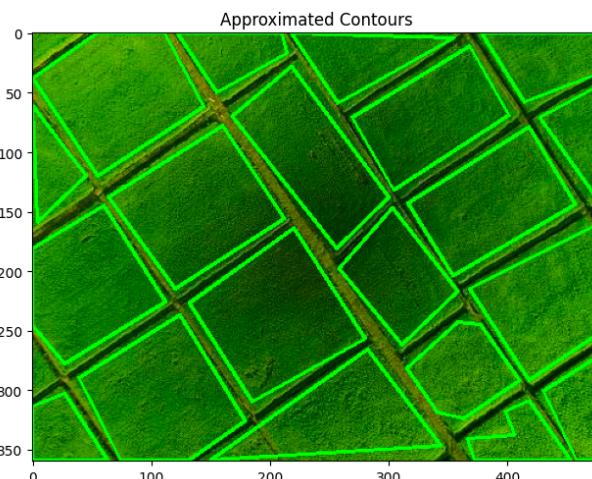
# sorting contours by their area
contours = sorted(contours, key=cv2.contourArea, reverse=True)
result_image = img.copy()
# writing the rank of farms according to their areas
target_conturs = []
for i, cntr in enumerate(contours):
    if cv2.contourArea(cntr) < 1000:
        continue
    # Find the centeroid of contours
    M = cv2.moments(cntr)
    cx = int(M['m10'] / M['m00'])
    cy = int(M['m01'] / M['m00'])
    cv2.drawContours(result_image, [cntr], -1, (0, 255, 0), 2)
    # cv2.putText(result_image, str(i+1), (cx, cy),
cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0,0,100), 2)
    # print("Farm " + str(i+1) + " Area: " +
str(cv2.contourArea(cntr)) + " pixels")
    target_conturs.append(cntr)

# showing result
imshow('Contours', result_image)
```



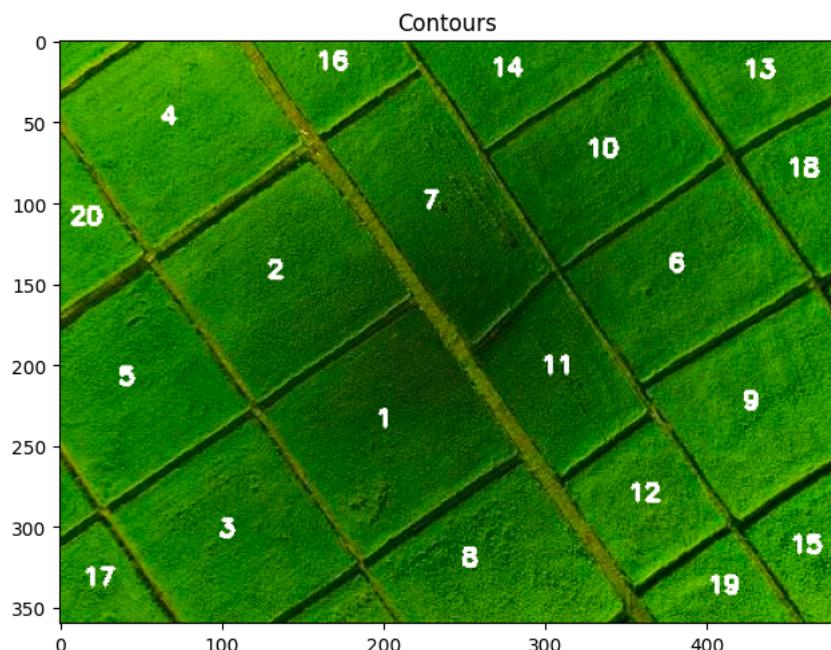
سپس با داگلاس پیوکر بر روی مزرعه ها به نحوی کانتور جدید با شکل واضح تر رسم می کنیم که مساحت شکل رسم شده حداقل به اندازه اپسیلون کمتر از کانتور اصلی باشد(دو اپسیلون برای کانتور های متفاوت با مساحت بیشتر و کمتر از ۸۰۰۰ واحد در نظر گرفته شده است):

```
# creating a Douglas-Peucker algorithm to get the approximated contours
def approx_contours(contours, epsilon):
    approximated_contours = []
    for contour in contours:
        approximated_contour = cv2.approxPolyDP(contour, epsilon, True)
        approximated_contours.append(approximated_contour)
    return approximated_contours
# splitting the contours into 2 parts the one with the area bigger than 8000 and the other with the area less than 8000
big_contours = []
small_contours = []
for contour in target_contours:
    if cv2.contourArea(contour) > 8000:
        big_contours.append(contour)
    else:
        small_contours.append(contour)
# Approximate the contours
epsilon = 0.030* cv2.arcLength(target_contours[0], True)
approximated_big_contours = approx_contours(big_contours, epsilon)
approximated_small_contours = approx_contours(small_contours, 0.65 * epsilon)
# merge the approximated contours
approximated_contours = approximated_big_contours + approximated_small_contours
# Draw the approximated contours
approximated_image = img.copy()
cv2.drawContours(approximated_image, approximated_contours, -1, (0, 255, 0), 2)
# Display the image
imshow("Approximated Contours", approximated_image)
```



سپس بر اساس مساحت ، زمین ها مرتب کرده و در مرکز آن می نویسیم:

```
# sort and rank the contours by their surface area
approximated_contours = sorted(approximated_contours,
key=cv2.contourArea, reverse=True)
result_image = img.copy()
for i, cntr in enumerate(approximated_contours):
    if cv2.contourArea(cntr) < 1000:
        continue
    # Find the centroid of contours
    M = cv2.moments(cntr)
    cx = int(M['m10'] / M['m00'])
    cy = int(M['m01'] / M['m00'])
    # cv2.drawContours(result_image, [cntr], -1, (0, 255, 0), 2)
    cv2.putText(result_image, str(i+1), (cx-10, cy),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,255,255), 2)
    print("Farm " + str(i+1) + " Area: " +
str(cv2.contourArea(cntr)) + " pixels")
# showing result
imshow('Contours', result_image)
```



برای این سوال ابتدا مدل resNet50 که بر روی دیتاست ImageNet ترین شده را فراخوانی می کنیم. خروجی این مدل یک عدد است که از فایل جیسون مربوطه می نوان به نام شی شناسایی شده دست پیدا کرد.

```
import torch
from torchvision import models, transforms
from PIL import Image

def getName(path : str):
    # Load the pre-trained model
    model = models.resnet50(pretrained=True)
    model.eval()

    # Define the image transformation
    transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]),
    ])

    # Load and transform the image
    image_path = path # replace with your image path
    image = Image.open(image_path)
    image = transform(image).unsqueeze(0)

    # Perform inference
    with torch.no_grad():
        outputs = model(image)

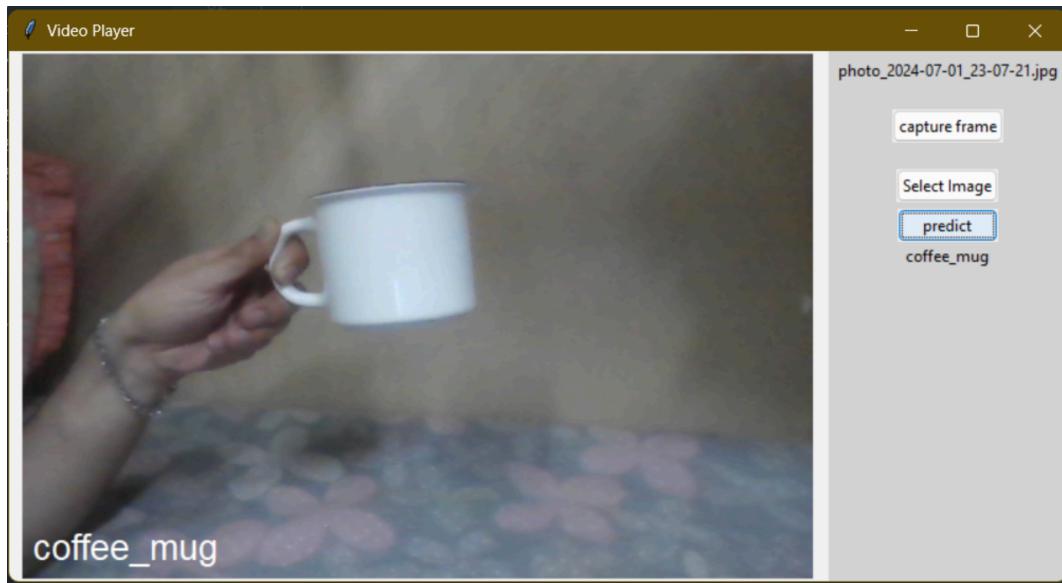
    # Get the predicted class
    _, predicted = torch.max(outputs, 1)
    print('Predicted class:', predicted.item())

    import json
    with open('imagenet.json') as json_file:
        labels_dict = json.load(json_file)

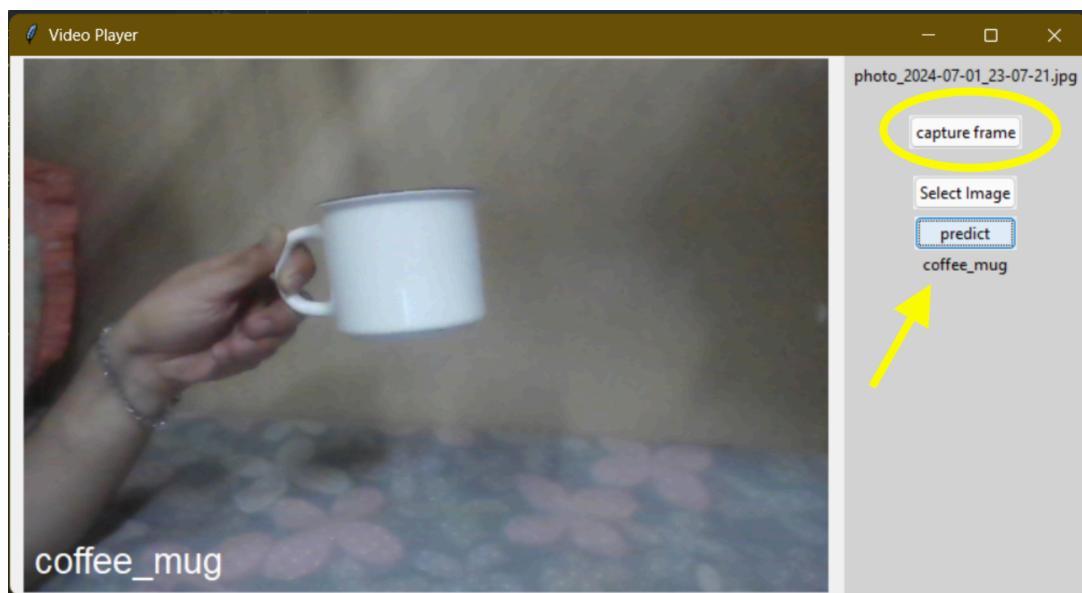
    lable_name = labels_dict[str(predicted.item())]
    print(f'the class name is {lable_name}')
```

حال GUI مربوط به تمرین اول که توانایی کار با وبکم و گرفتن عکس را دارا بود استفاده کردیم در کنار این موضوع، به یکی از دکمه ها حالت انتخاب فایل دادیم و تا با زدن آن پنجره انتخاب فایل باز شودو تصویر مورد نظر را انتخاب کنیم.

نتیجه GUI بدین صورت است:



در صورتی که دکمه capture انتخاب شود یک عکس از وبکم گرفته شده و به مدل داده می شود و با زدن دکمه پریدیکت نام شی پیش بینی شده را در زیر دکمه نمایش می دهد، این رویکرد همچنین در صورت انتخاب عکس از روی کامپیوتر نیز انجام می شود و پس از نمایش داده شدن اسم فایل در بالای برنامه ، با زدن دکمه پریدیکت شی پیش بینی شده آن در پایین دکمه نمایش داده می شود:



همچنین در هر لحظه تصویر و بكم به مدل داده شده و پیش بینی آن در پایین تصویر(بدون رایت شدن روی تصویر اصلی) نوشته می شود:

