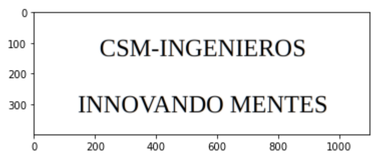
Deteccion de Caracteres OCR

```
In [9]: import cv2 as cv
        import numpy as np
        import matplotlib.pyplot as plt

        img = cv.imread("img.png")
        plt.imshow(img)
        img2 = img[250:650,100:1200]
        plt.imshow(img2)
        cv.imwrite("img_csm.png",img2)
```

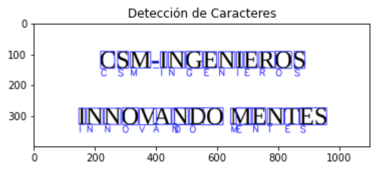
```
Out[9]: True
```



Caracteres de una imagen

```
In [10]: import cv2 as cv
         import pytesseract as ocr
         import numpy as np
         import matplotlib.pyplot as plt

         img = cv.imread('img_csm.png')
         img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
         hImg, wImg,_ = img.shape
         boxes = ocr.image_to_boxes(img)
         for b in boxes.splitlines():
             #print(b)
             b = b.split(' ')
             #print(b)
             x, y, w, h = int(b[1]), int(b[2]), int(b[3]), int(b[4])
             cv.rectangle(img, (x,hImg- y), (w,hImg- h), (50, 50, 255), 2)
             cv.putText(img,b[0],(x,hImg- y+25),cv.FONT_HERSHEY_SIMPLEX,1,(50,50,255),2)
         plt.imshow(img), plt.title("Detección de Caracteres")
         #cv.imshow('Detección de Caracteres', img)
         #cv.waitKey(0)
         #cv.destroyAllWindows()
```

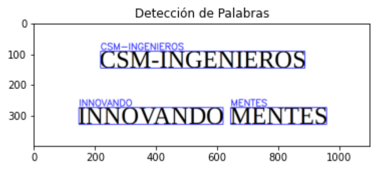
```
Out[10]: (<matplotlib.image.AxesImage at 0x7f5c59488320>,
          Text(0.5, 1.0, 'Detección de Caracteres'))
```



Palabras en una Imagen

```
In [11]: # #[   0          1           2           3           4          5          6       7       8         9        10       11 ]
         # #['level', 'page_num', 'block_num', 'par_num', 'line_num', 'word_num', 'left', 'top', 'width', 'height', 'conf', 'text']

         import cv2 as cv
         import pytesseract as ocr
         import numpy as np
         import matplotlib.pyplot as plt

         img = cv.imread('img_csm.png')
         img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

         boxes = ocr.image_to_data(img)
         for a,b in enumerate(boxes.splitlines()):
                 #print(b)
                 if a!=0:
                     b = b.split()
                     if len(b)==12:
                         x,y,w,h = int(b[6]),int(b[7]),int(b[8]),int(b[9])
                         cv.putText(img,b[11],(x,y-5),cv.FONT_HERSHEY_SIMPLEX,1,(50,50,255),2)
                         cv.rectangle(img, (x,y), (x+w, y+h), (50, 50, 255), 2)

         plt.imshow(img), plt.title("Detección de Palabras")
         #cv.imshow('Detección de Palabras', img)
         #cv.waitKey(0)
         #cv.destroyAllWindows()
```

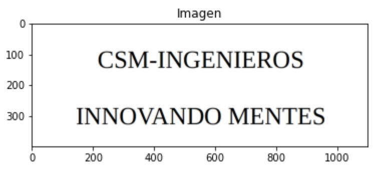
```
Out[11]: (<matplotlib.image.AxesImage at 0x7f5c594d61d0>,
          Text(0.5, 1.0, 'Detección de Palabras'))
```



Extrayendo texto

In [12]:
```
import cv2 as cv
import pytesseract as ocr
import numpy as np
import matplotlib.pyplot as plt

img = cv.imread('img_csm.png')
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

text = ocr.image_to_string(img,lang="eng")
plt.imshow(img),plt.title("Imagen")
print("Texto extraido de la imagen: \n\n",text)
```

```
Texto extraido de la imagen:

 CSM-INGENIEROS

INNOVANDO MENTES
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

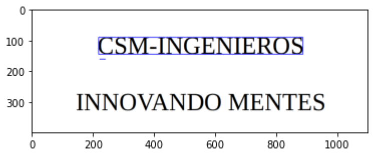
In [ ]:

In [ ]:

Detectar solo Digitos

In [18]:
```
import cv2 as cv
import pytesseract as ocr
import numpy as np
import matplotlib.pyplot as plt

img = cv.imread('img_csm.png')
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
hImg, wImg,_ = img.shape
conf = r'--oem 3 --psm 6 outputbase digits'
boxes = ocr.image_to_boxes(img,config=conf)
for b in boxes.splitlines():
    print(b)
    b = b.split(' ')
    print(b)
    x, y, w, h = int(b[1]), int(b[2]), int(b[3]), int(b[4])
    cv.rectangle(img, (x,hImg- y), (w,hImg- h), (50, 50, 255), 2)
    cv.putText(img,b[0],(x,hImg- y+25),cv.FONT_HERSHEY_SIMPLEX,1,(50,50,255),2)

plt.imshow(img)
cv.imshow('img', img)
cv.waitKey(0)
cv.destroyAllWindows()
```

```
- 219 255 888 310 0
['-', '219', '255', '888', '310', '0']
```



Detectando caracteres desde la webcam

In [ ]:
```
import cv2 as cv
import pytesseract as ocr
import numpy as np
from PIL import ImageGrab
import time

cap = cv.VideoCapture(0)
cap.set(3,640)
cap.set(4,480)
def captureScreen(bbox=(300,300,1500,1000)):
    capScr = np.array(ImageGrab.grab(bbox))
    capScr = cv.cvtColor(capScr, cv.COLOR_RGB2BGR)
    return capScr
while True:
    timer = cv2.getTickCount()
    _,img = cap.read()
    #img = captureScreen()
    #DETECTING CHARACTERES
    hImg, wImg,_ = img.shape
    boxes = ocr.image_to_boxes(img)
    for b in boxes.splitlines():
        #print(b)
        b = b.split(' ')
        #print(b)
        x, y, w, h = int(b[1]), int(b[2]), int(b[3]), int(b[4])
        cv.rectangle(img, (x,hImg- y), (w,hImg- h), (50, 50, 255), 2)
        cv.putText(img,b[0],(x,hImg- y+25), cv.FONT_HERSHEY_SIMPLEX,1,(50,50,255),2)
    fps = cv.getTickFrequency() / (cv.getTickCount() - timer);
    #cv2.putText(img, str(int(fps)), (75, 40), cv.FONT_HERSHEY_SIMPLEX, 0.7, (20,230,20), 2);
    cv.imshow("Result",img)
    cv.waitKey(1)
    cv.destroyAllWindows
```

Lectura de codigos QR

Lectura desde Imagen

In [8]:
```
import cv2 as cv
import numpy as np
from pyzbar.pyzbar import decode

img = cv.imread("qr1.jpg")
#code = decode(img)
#print(code)
for barcode in decode(img):
    print(barcode.data)
    myData = barcode.data.decode("utf-8")
    print(myData)
```

```
b'www.indicioseditores.com'
www.indicioseditores.com
```

In [2]:
```
type(code)
```

Out[2]:
```
list
```

Lectura desde Camara WEB

In [2]:
```
import cv2 as cv
import numpy as np
from pyzbar.pyzbar import decode

cap = cv.VideoCapture(0)
cap.set(3,640)
cap.set(4,480)

while(cap.isOpened()):

    ret, frame = cap.read()

    for barcode in decode(frame):
        print(barcode.data)
        myData = barcode.data.decode("utf-8")
        print(myData)
    if cv.waitKey(1) & 0xFF == ord('q'):
        break

    cv.imshow("Result QR",frame)
    cv.waitKey(1)
cap.release()
cv.destroyAllWindows()
```

Cuadro delimitador del QR

In [9]:
```
import cv2 as cv
import numpy as np
from pyzbar.pyzbar import decode

cap = cv.VideoCapture(1)
cap.set(3,640)
cap.set(4,480)

while(cap.isOpened()):

    ret, frame = cap.read()
    if ret:

        for barcode in decode(frame):
            print(barcode.data)
            myData = barcode.data.decode("utf-8")
            print(myData)
            pst = np.array([barcode.polygon], np.int32)
            pst = pst.reshape((-1,1,2))
            cv.polylines(frame,[pst],True, (255,0,255))

        if cv.waitKey(1) & 0xFF == ord('q'):
            break

        cv.imshow("Result QR",frame)
        cv.waitKey(1)
    else:
        break
cap.release()
cv.destroyAllWindows()
```

In [ ]:

In [ ]:

In [ ]:

Deteccion de Peatones con HOG y SVM

In [5]:
```
from __future__ import print_function
from imutils.object_detection import non_max_suppression
import numpy as np
import cv2
import imutils

image = cv2.imread("p3.jpg")
#resize image -scale
scale = 1.0
w = int(image.shape[1] / scale)
image = imutils.resize(image, width=w)
#image = imutils.resize(image, width=min(400, image.shape[1]))
# initialize the HOG descriptor/person detector
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

orig = image.copy()

# detect people in the image
(rects, weights) = hog.detectMultiScale(image, winStride=(4, 4),
                                        padding=(8, 8), scale=1.05)

 # draw the original bounding boxes
for (x, y, w, h) in rects:
           cv2.rectangle(orig, (x, y), (x + w, y + h), (0, 0, 255), 2)

 # apply non-maxima suppression to the bounding boxes using a
 # fairly large overlap threshold to try to maintain overlapping
 # boxes that are still people
rects = np.array([[x, y, x + w, y + h] for (x, y, w, h) in rects])
pick = non_max_suppression(rects, probs=None, overlapThresh=0.5)

 # draw the final bounding boxes
for (xA, yA, xB, yB) in pick:
             cv2.rectangle(image, (xA, yA), (xB, yB), (0, 255, 0), 2)

cv2.imshow("Output", orig)
cv2.imshow("OutputNMS", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In [7]:
```
from __future__ import print_function
from imutils.object_detection import non_max_suppression
import numpy as np
import cv2
import imutils

image = cv2.imread("p2.jpg")
#resize image -scale
scale = 1.0
w = int(image.shape[1] / scale)
image = imutils.resize(image, width=w)
#image = imutils.resize(image, width=min(400, image.shape[1]))
# initialize the HOG descriptor/person detector
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

orig = image.copy()

# detect people in the image
(rects, weights) = hog.detectMultiScale(image, winStride=(4, 4),
                                        padding=(8, 8), scale=1.05)

 # draw the original bounding boxes
for (x, y, w, h) in rects:
           cv2.rectangle(orig, (x, y), (x + w, y + h), (0, 0, 255), 2)

 # apply non-maxima suppression to the bounding boxes using a
 # fairly large overlap threshold to try to maintain overlapping
 # boxes that are still people
rects = np.array([[x, y, x + w, y + h] for (x, y, w, h) in rects])
pick = non_max_suppression(rects, probs=None, overlapThresh=0.65)

 # draw the final bounding boxes
for (xA, yA, xB, yB) in pick:
             cv2.rectangle(image, (xA, yA), (xB, yB), (0, 255, 0), 2)

cv2.imshow("Output", orig)
cv2.imshow("OutputNMS", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In [ ]: