Trabajo práctico 4 - Balance

Alumnos:

- Carol lugones Ignacio (100073)
- Torresetti Lisandro (99846)

Objetivo

Trabajando sobre la imagen sombreado:

1. Encontrar el umbral con búsqueda binaria según lo indicado en el Jupyter Notebook y los videos. Comparar el resultado con la binarización por el método de Otzu.
2. Programar el método de binarización local por Bernsen. Ajustarlo para obtener buenos resultados de binarización sobre esta imagen.

In [1]:

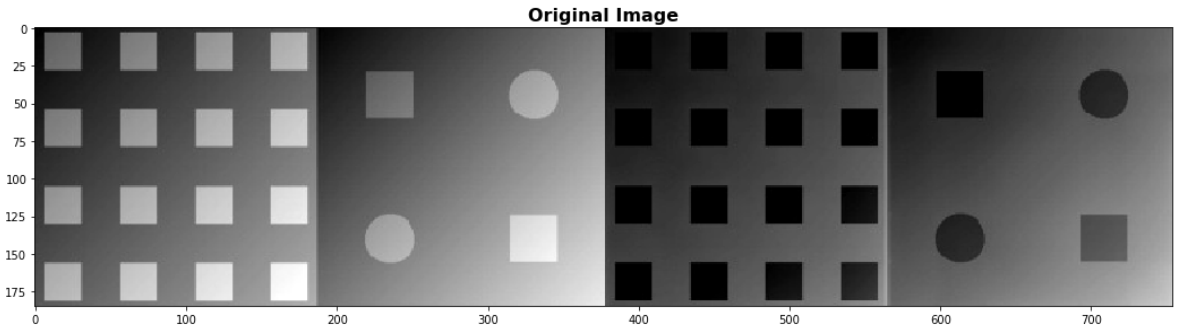
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from glob import glob
%matplotlib inline
```

In [2]:

```
def plotter(image, title = '', imgSize = (18,9), grayScale = False): #Funcion auxil
    plt.figure(figsize=imgSize)
    plt.title(title, fontsize = 16, fontweight = "bold")
    plt.imshow(image) if not grayScale else plt.imshow(image, cmap='gray', vmin=0,
    plt.show()
```

In [3]:

```
img = cv.imread('Sombreado.png', cv.IMREAD_GRAYSCALE)
plotter(img, 'Original Image', grayScale=True)
```



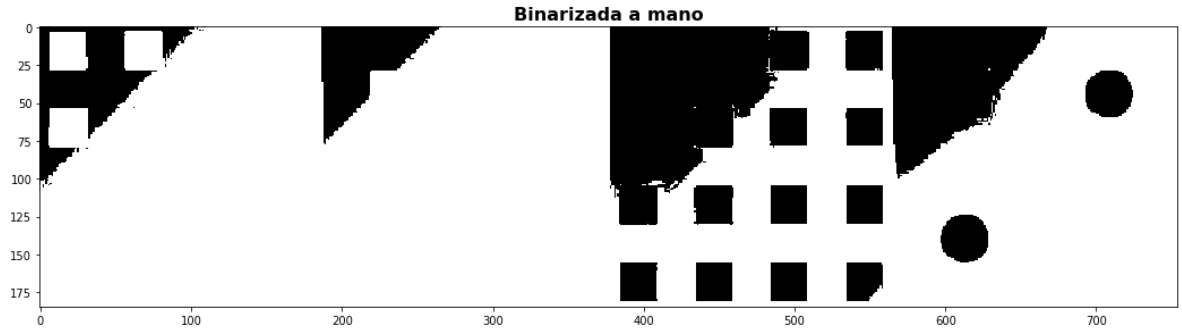
1) Búsqueda del umbral

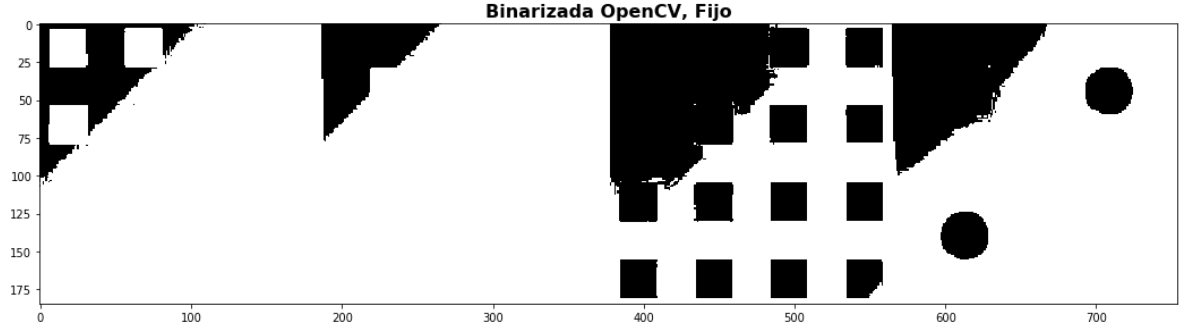
In [4]:

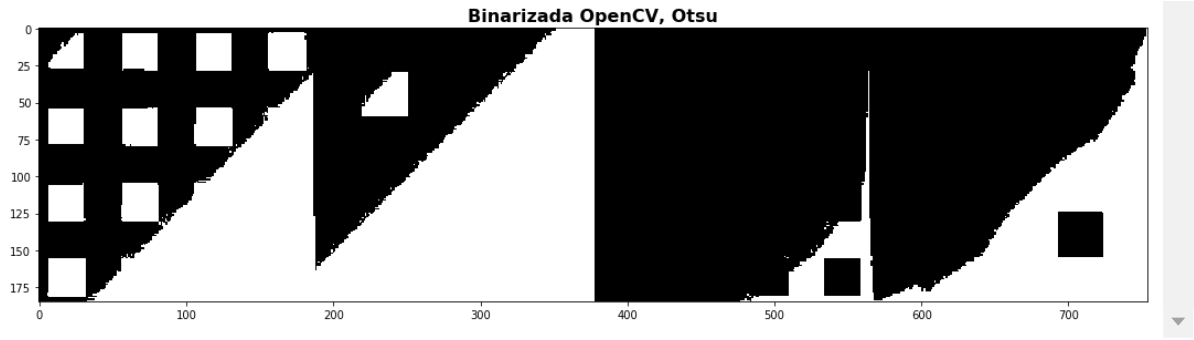
```
#Ejercicio: Completar lo que falte y comparar contra el método de Otsu (por ejemplo

#Paso1: Definir umbral inicial (en general la media de la imagen)

def buscar_umbral(img, umbral=128, delta_T=1.0):

    #Paso2: Dividir la imagen en dos partes
    # Usar np.where para encontrar los índices
    xp1,yp1 = np.where(img < umbral)
    xp2,yp2 = np.where(img >= umbral)


    #Paso3: Encontrar la media de cada parte
    media_p1 = np.mean(img[xp1][yp1])
    media_p2 = np.mean(img[xp2][yp2])

    #Paso4: Calcular el nuevo umbral (promedio entre media anterior y actual)
    nuevo_umbral = (media_p1 + media_p2) / 2

    #Paso5: Criterio de detención (o recalculo)
    if abs(nuevo_umbral - umbral) < delta_T:
        return nuevo_umbral
    else:
        return buscar_umbral(img, umbral=nuevo_umbral)
```

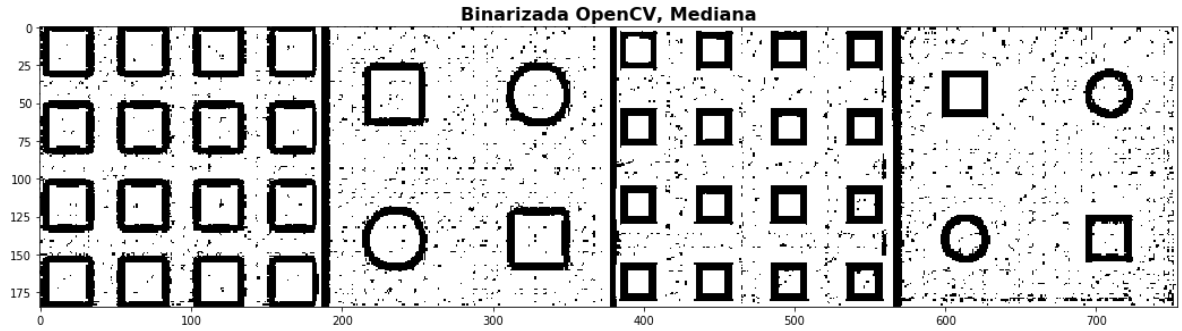
In [5]:

```
# Funcion de binarización (a mano)
def global_threshold(image, thres_value, val_high, val_low):
    img = image.copy()
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            if image[i,j] > thres_value:
                img[i,j] = val_high
            else:
                img[i,j] = val_low
    return img

umbral = np.round(buscar_umbral(img, round(np.mean(img)))) #Obtenemos el umbral
# Realizamos los graficos de binarizacion con distintos metodos y pasandole el umbr

# Imagen binarizada (a mano - Fijo)
img_bin = global_threshold(img, umbral, 255, 0);
plotter(img_bin, 'Binarizada a mano', grayScale=True)

# Imagen binarizada (embebida - Fijo) - Parámetros: Imag_original,umbral,valor_máxi
ret, thresh = cv.threshold(img,umbral,255,cv.THRESH_BINARY)
plotter(thresh,'Binarizada OpenCV, Fijo', grayScale=True)

# Imagen binarizada (embebida - Otsu) - Parámetros: Imag_original,umbral,valor_máxi
ret, thresh = cv.threshold(img,umbral,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
plotter(thresh,'Binarizada OpenCV, Otsu', grayScale=True)

# Imagen binarizada (embebida - Mediana) - Parámetros: Imag_original,valor_máximo,m
thresh = cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_MEAN_C,cv.THRESH_BINARY,11
plotter(thresh,'Binarizada OpenCV, Mediana', grayScale=True)
```









2) Método de binarización local por Bernsen

In [6]:

```
def getSubmatrix(matrix, row, col, size):
    submatrix = []
    for i in range(row, (row + size) if row + size < len(matrix) else len(matrix)):
        for j in range(col, (col + size) if col + size < len(matrix[0]) else len(ma
            submatrix.append(matrix[i][j])
    return submatrix
```

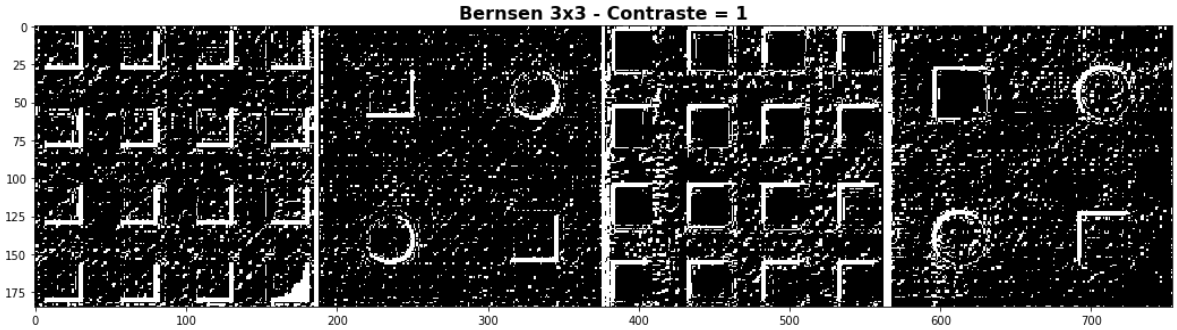
In [7]:

```
def bernsen(image, windowSize, contrast):
    img = image.copy()
    aux = []
    for row in range(0, len(image), 1):
        for col in range(0, len(image[0]), 1):
            submatrix = getSubmatrix(image, row, col, windowSize) #Generamos la ven
            midGray = np.mean(submatrix)
            maxIntensity, minIntensity = max(submatrix), min(submatrix)
            localContrast = maxIntensity - minIntensity
            aux.append(localContrast)

            for i in range(len(submatrix)):
                if (localContrast < contrast):
                    submatrix[i] = 255 if midGray >= 128 else 0
                else:
                    submatrix[i] = 255 if submatrix[i] >= midGray else 0

            #Ahora actualizo los valores de la imagen original
            auxCont = 0
            for i in range(row, row + windowSize):
                if (i >= len(img)):
                    break
                for j in range(col,  col + windowSize):
                    if (j >= len(img[0])):
                        break
                    img[i][j] = submatrix[auxCont]
                    auxCont += 1

    print("Min local contrast = {} // Max local contrast = {}".format(min(aux), max
    return img
```

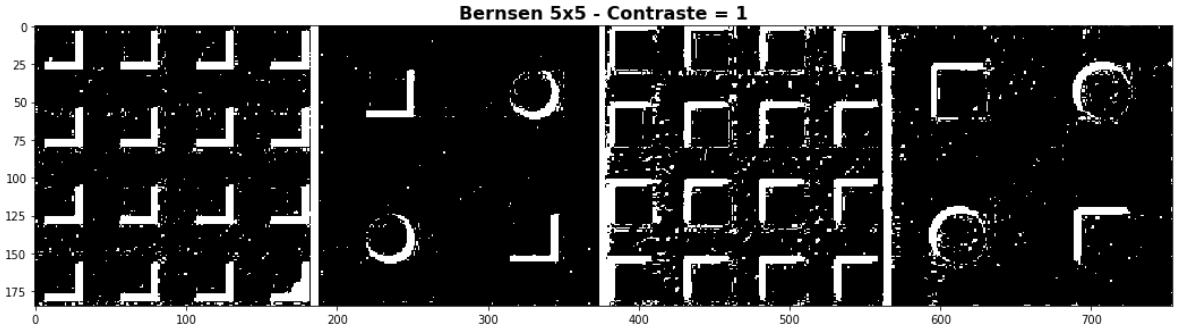
In [8]:

```
#contrasts = range(1,11) #Si tarda mucho cambiar el rango
contrasts = [1,5,7,10]  #Para ver la diferenia entre contraste y contraste mas rapi
for contrast in contrasts:
    for windowSize in [3,5,7]:
        imgCopy = img.copy()
        imgCopy = bernsen(imgCopy, windowSize, contrast)
        title = 'Bernsen {}x{} - Contraste = {}'.format(windowSize, windowSize, con
        plotter(imgCopy, title, grayScale=True)
```

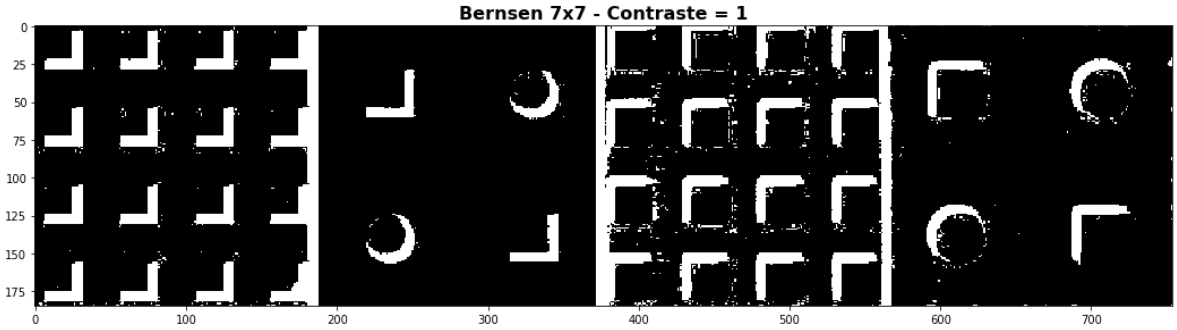
Min local contrast = 0 // Max local contrast = 153



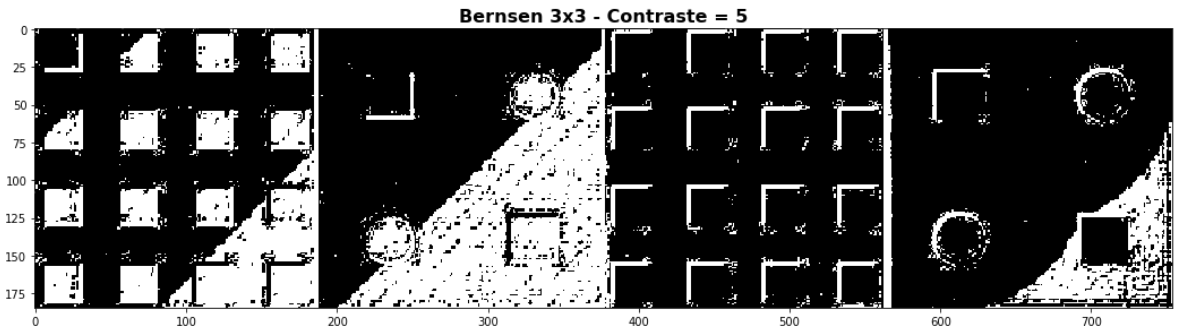
Min local contrast = 0 // Max local contrast = 165



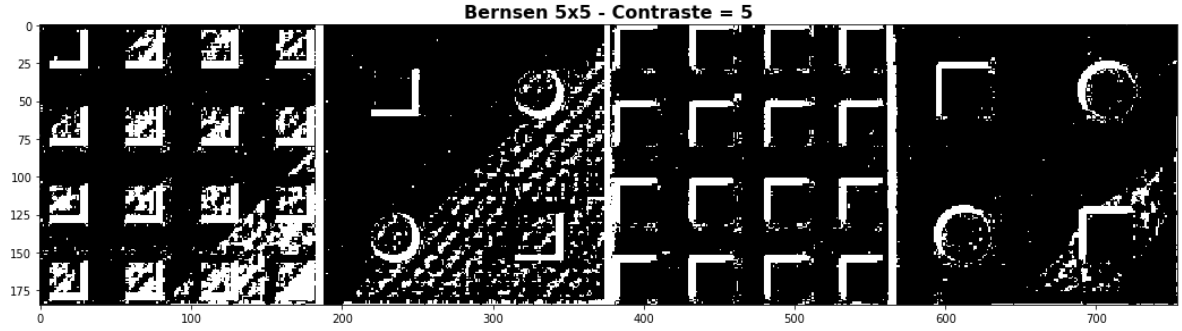
Min local contrast = 0 // Max local contrast = 168



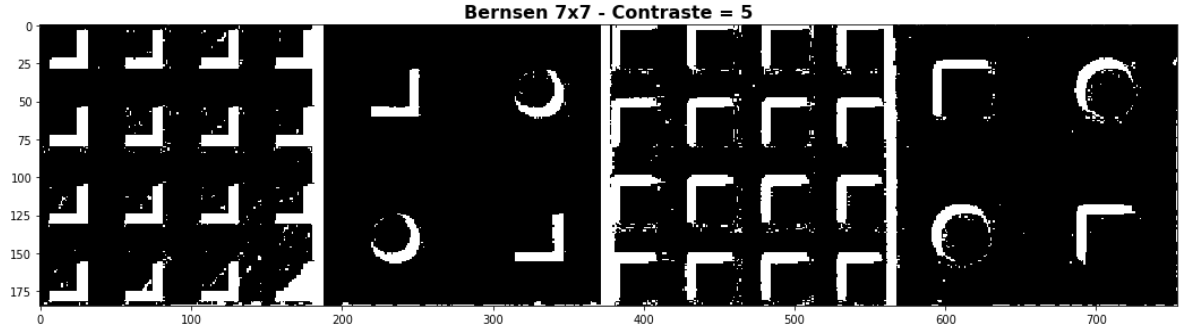
Min local contrast = 0 // Max local contrast = 153



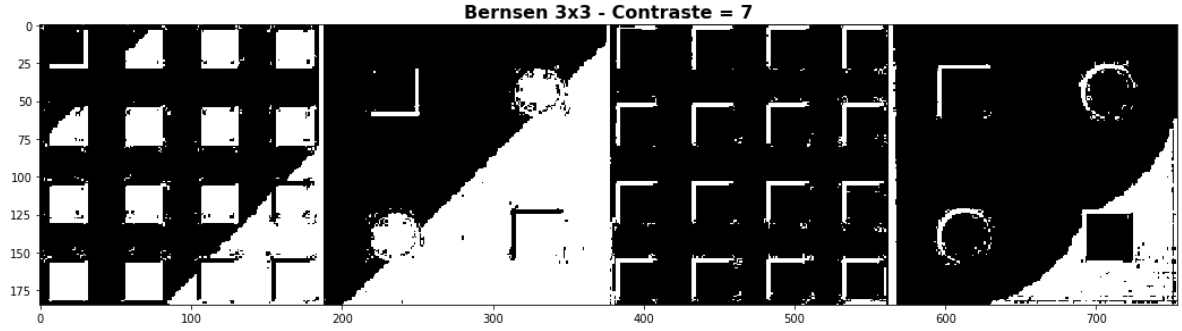
Min local contrast = 0 // Max local contrast = 165



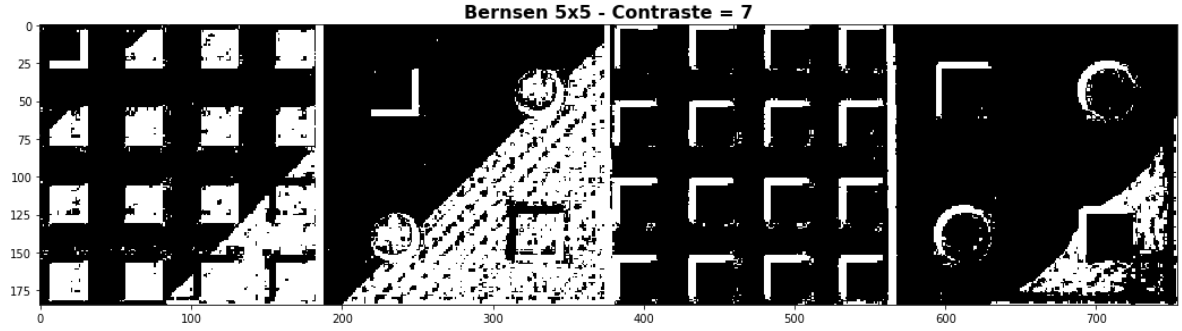
Min local contrast = 0 // Max local contrast = 168



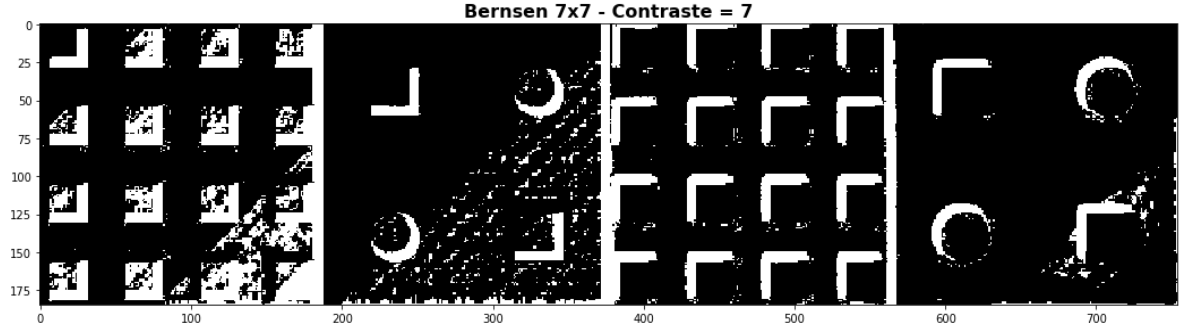
Min local contrast = 0 // Max local contrast = 153



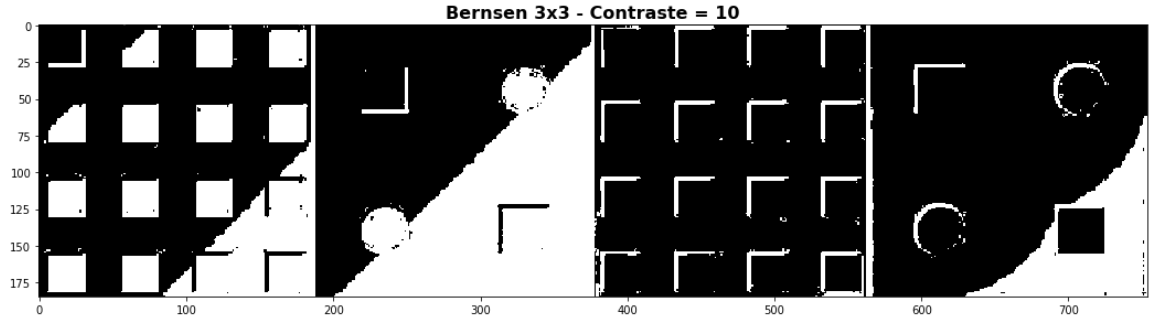
Min local contrast = 0 // Max local contrast = 165



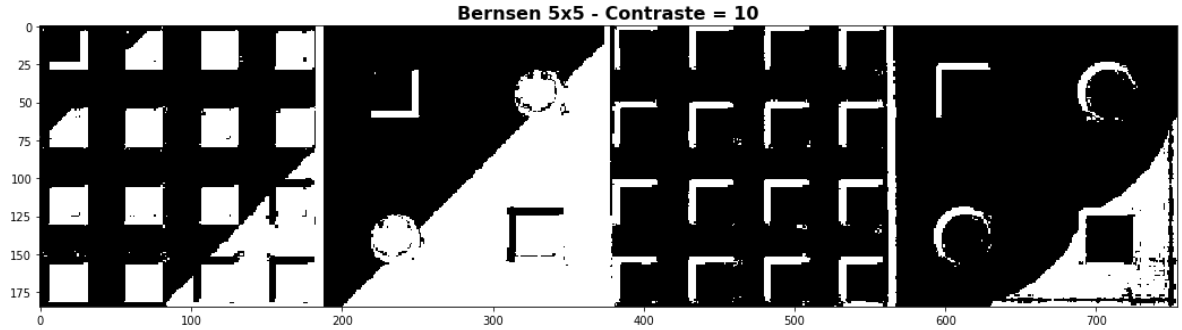
Min local contrast = 0 // Max local contrast = 168



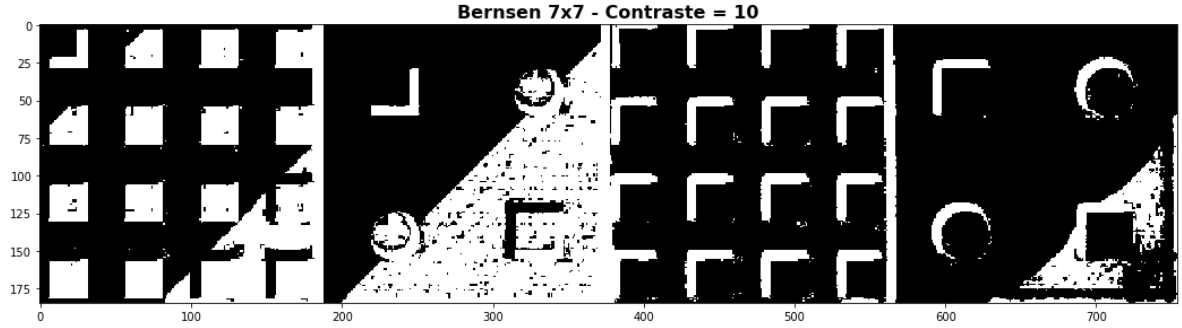
Min local contrast = 0 // Max local contrast = 153




```
Min local contrast = 0 // Max local contrast = 165
```




```
Min local contrast = 0 // Max local contrast = 168
```



Conclusiones

Punto 1:

En este punto se pudo notar que la performance para obtener la binarización fue mucho más rápida que en el punto 2, pero se puede ver que la imagen obtenida por cada método no se asemeja tanto a la original como sucede en el punto 2. También se puede observar las diferencias entre cada método de binarización; en el caso de la binarización con el algoritmo otorgado por la cátedra y la binarización por openCV no se observan diferencias entre las imagenes obtenidas, sin embargo si las comparamos con la binarización de Otsu si. Por último se ve que la binarización de openCV por la mediana sirve para obtener los bordes de las figuras geométricas.

Punto 2:

Como mencionamos anteriormente para obtener cada una de las imagenes generadas el tiempo requerido fue mayor, en parte esto se debe a que el algoritmo no fue implementado de forma óptima. Del resultaso se puede notar que a medida que aumentamos el contraste menos diferencias se ven entre un tamaño de ventana y otro, y además la imagen binarizada se parece más a la original ya que se distinguen mejor las figuras geométricas como también el fondo, y no se ven partes en un solo color como en el punto 1 que solo se diferencian unas figuras y luedo es todo de un solo color. Podemos concluir que si bien este método es más lento permite obtener mejores resultados que los métodos de binariación utilizados en el punto anterior.