Nombres:
Marco Alan Moreno León A01747315
Ernesto Ibhar Guevara Gómez A01746121

Proyecto Final: Detector de Sudoku en OpenCV

Fase 1: Pre-processing

En esta parte se procesó la input image para hacerla binaria y poder deshacernos del ruido y/o de partes que no nos importaran. En esta parte se aplico una erosión a la imagen y posterior a esto se insertó un rectángulo de color blanco desde la coordenada (0,0) hasta la ultima coordenada de la imagen para aplicar un floodFill de color negro a todo lo que estuviera tocando este rectangulo y finalmente se aplico una dilatación para recuperar algunos pixeles de algunas imagenes. Finalmente se obtuvieron los contornos de la imagen.

```
   # Convert BGR to grayscale:
   grayscaleImage = cv2.cvtColor(inputImage, cv2.COLOR_BGR2GRAY)
   #showImage("grayscaleImage: "+fileName, grayscaleImage)

   # Convert grayScale to binary:
   binaryImage =
cv2.adaptiveThreshold(grayscaleImage,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
            cv2.THRESH_BINARY+cv2.THRESH_BINARY_INV,155,25)
   #showImage("Adaptative: "+fileName, binaryImage)

   #Structuring element size
   kernelSize = 3
   opIerations = 1
   # Get the Structuring element:
   morphKernel = cv2.getStructuringElement(cv2.MORPH_RECT, (kernelSize,
kernelSize))

   # Perform Erosion
   erodeImage = cv2.morphologyEx(binaryImage,cv2.MORPH_ERODE,morphKernel,
iterations=opIerations)
   #showImage("Erosion"+fileName,erodeImage)

   #Obtaining the Height and the Width of the image
   originalHeight, originalWidth = inputImage.shape[:2]
   # Draw a rectangle with wite line borders of thickness of 350 px
   image = cv2.rectangle(erodeImage, (0,0),(originalWidth,originalHeight),
255, 350)
   #showImage("Rect"+fileName,image)


   #Aplying floodFill
   fillColor = (0, 0, 0)
   leftCorner = (0, 0)
   cv2.floodFill(erodeImage, None, leftCorner, fillColor)
   #showImage("Flood-Filled Image" +fileName, erodeImage)

   # Perform Dilatation
   opIerations = 2
   dilatedImage = cv2.morphologyEx(erodeImage,cv2.MORPH_DILATE,morphKernel,
iterations=opIerations)
```

```
   #showImage("Dilated"+fileName,dilatedImage)

   contours,hierarchy = cv2.findContours(dilatedImage, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
   # Contour conunter:
   contourCounter = 1
   #Store the code bounding rects here:
   codeRectangles=[]
   #Store the code bounding rects of the numbers here:
   codeRectangles2=[]
   #Store hipot
   hipotList = {}
   #Loop through the countours list
```

Fase 1: Identificación de las esquinas del tablero de números

En esta parte se obtuvieron los contornos de la imagen y mediante el área y la herencia determinamos que contorno era el que queríamos y de ese fue del que obtuvimos las las esquinas del tablero. Seteando cada vértice en un diccionario con keys de la distancia entre las coordenadas del vértice y el punto (0,0), esto para determinar que todos los vértices fueran iguales en todas las imágenes.

```
#Fase 1. Identificación de las esquinas del tablero de números

   for i in range(len(contours)):
       c=contours[i]
       h = hierarchy[0][i]
       #print(h)
       #Draw contours:
       color=(0,0,255)

       cv2.drawContours(dilatedImage, [c], 0, color,3)
       # Get the bounding rect:
       boundRect = cv2.boundingRect(c)

       # Get the bounding Rect data:
       rectX = int(boundRect[0])
       rectY = int(boundRect[1])
       rectWidth = int(boundRect[2])
       rectHeight = int(boundRect[3])

       # Draw Label:
       font = cv2.FONT_HERSHEY_SIMPLEX
       color = (0, 255, 0)
       fontScale = 1
       fontThickness = 2

       # Compute bounding rect area:
       rectArea = rectWidth * rectHeight

       # Compute the aspect ratio:
       rectAspectRatio = rectHeight / rectWidth

       # increase counter:
       contourCounter += 1
       # Default color:
       color = (0, 0, 255)
       #minArea of the contour
       minArea=1200000
```

```
       minAspectRatio = 0.8

       if rectArea > minArea and rectAspectRatio > minAspectRatio and
h[3]==-1:

            # Draw Rectangle:
            color = (0, 255, 0) # Green
            #print(h)
            #Store the rectangle:
            codeRectangles.append((rectX,rectY,rectWidth,rectHeight))
             #Approximate the contour to a polygon:
            perimeter = cv2.arcLength(c,True)
            #Approximate accuracy:
            approxAccuracy = 0.05 * perimeter

            #Get vertices
            #Last Flags indicates a closed curve:
            vertices = cv2.approxPolyDP(c,approxAccuracy,True)

            #Print the polygons vertices
            verticesFound = len(vertices)
            #print(verticesFound)
            #Prepare the input points array
            inPoints = np.zeros((4,2), dtype="float32")
            #if verticesFound == 4:
             #   print(vertices)

            #Format points:
            for p in range(verticesFound):
                #Get corner points
                currenPoints = vertices[p][0]

                #Get x, y
                x = int(currenPoints[0])
                y = int(currenPoints[1])
                #Distance to determinate points
                hipot= x+y
                #Adding elements of the distance and (x,y) of the vertices in
a dictionary
                hipotList[hipot]=(x,y)
                #Sorting the keys (distance) of dictionary
                sorted_keys= sorted(hipotList.keys())
                #print(inPoints[p][0])
                #print(sorted_keys)

                if len(sorted_keys)==4:
                    #Changing the values to set the cornes only in one order
for corner 2 and 3
                    if hipotList[sorted_keys[1]][0]<1800:
                        sorted_keys[1],sorted_keys[2]=
sorted_keys[2],sorted_keys[1]
                    #print(sorted_keys)

                    #Showing the corners in same position for every image
                    for i in range(len(sorted_keys)):
                        #print(hipotList[sorted_keys[i]][1])
                        #Store in points array:
                        inPoints[i][0] = hipotList[sorted_keys[i]][0]
                        inPoints[i][1] = hipotList[sorted_keys[i]][1]
                        #Draw the corner points
```

```
cv2.circle(inputImage,hipotList[sorted_keys[i]],5,(255,0,0),5)
                        # Draw corner number:
                        cv2.putText(inputImage, str(i+1),
hipotList[sorted_keys[i]], cv2.FONT_HERSHEY_SIMPLEX, fontScale=2, color=(0,
0, 255),
                        thickness=2)

                        # Show them corners:
                        #showImage("Corners"+fileName, inputImage)
```

Fase 2. Extracción de los números del tablero

En esta parte una vez tomada la perspectiva de todos los vértices se voltearon las imágenes que estaban desde otra perspectiva para poderlas ver las imágenes planas. Posteriormente hicimos igual que la limpieza anterior, pusimos un rectángulo de (0,0) al último píxel de la imagen para aplicar un floodfill y eliminar todo lo que estuviera tocando ese rectángulo al igual que se aplicó una erosión para quitar algunos píxeles que se habían quedado ahí y por último una dilatación para que los números recuperarán su grosor. Finalmente igual sacamos sus contornos.

```
            #Aplying FloodFill
            originalHeight2, originalWidth2 = rectifiedImage.shape[:2]
            image2 = cv2.rectangle(rectifiedImage,
(0,0),(originalWidth2,originalHeight2), 255, 30)
            #showImage("Rect2"+fileName,image2)
            fillColor = (0, 0, 0)
            leftCorner = (0, 0)
            cv2.floodFill(rectifiedImage, None, leftCorner, fillColor)
            #showImage("Flood-Filled Image" +fileName, rectifiedImage)

            #Perform Erosion
            opIerations = 1
            erodeImage2 =
cv2.morphologyEx(rectifiedImage,cv2.MORPH_ERODE,morphKernel,
iterations=opIerations)
            #showImage("Erosion2"+fileName,erodeImage2)
            #Perform Dilation
            opIerations = 1
            dilatedImage2 =
cv2.morphologyEx(erodeImage2,cv2.MORPH_DILATE,morphKernel,
iterations=opIerations)
            #showImage("Dilated2"+fileName,dilatedImage2)
            # Contours:
            contours2, hierarchy2 =
cv2.findContours(dilatedImage2,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

            colorImage = cv2.cvtColor(dilatedImage2,cv2.COLOR_GRAY2BGR)
            showImage("colored"+fileName,colorImage)
```

Fase 3: Números

En esta parte se extrajeron los números mediante el contorno y el centro del número de todas las imágenes.

```
            for c in contours2:

                # Approximate the contour to a circle:
                 # Compute blob area:
                blobArea = cv2.contourArea(c)
                # calculate moments for each contour

                M = cv2.moments(c)
                # calculate x,y coordinate of center

                cX = int(M["m10"] / M["m00"])

                cY = int(M["m01"] / M["m00"])

                cv2.circle(colorImage, (cX, cY), 3, (0, 0, 255), -1)
                #Obtaining boundingRect
                boundRect2 = cv2.boundingRect(c)
                # Get the bounding Rect data:
                x, y, w, h = cv2.boundingRect(c)

                #print("x,y,w,h:",x,y,w,h)

                boundingRectangle = cv2.rectangle(colorImage,(x,y), (w+x,h+y),
(20, 255, 57), 2)
                #showImage("Bounding rectangle2"+ fileName, boundingRectangle)
```

Fase 3. Procesamiento de los números en muestras utilizables por el clasificador

En esta parte de cada número que se extraía le hicimos un resize y pegamos en el centro para que todas las imágenes fueran de 70x70.

```
numbersImage = dilatedImage2[y: y + h, x: x + w]
               (height,width)= numbersImage.shape[:2]
               aspectRatio = width/height
               newHeight= int(height*2)
               neWidth = int(newHeight*aspectRatio)
               newSize = (neWidth, newHeight)
               numbersImage = cv2.resize(numbersImage,newSize,cv2.INTER_AREA)

               sampleHeight = 70
               sampleWidth = sampleHeight
               canvas = np.zeros((sampleHeight, sampleWidth), np.uint8)
               canvasCopy = canvas.copy()
               #print("Canvas W: " + str(sampleWidth) + ", H: " +
str(sampleHeight))
               # Get the bounding rect data:
               rectX = int(boundRect2[0])
               rectY = int(boundRect2[1])
               rectWidth = int(boundRect2[2])
               rectHeight = int(boundRect2[3])

               ox = int(sampleWidth/2)-(neWidth//2)
               oy = int(sampleHeight/2)-(newHeight//2)
```

```
            canvas[oy:oy + newHeight, ox:ox + neWidth] = numbersImage
            #showImage("canvas",canvas)
```

Fase 4. Classifier

En esta parte entrenamos a nuestro classifier al inicio del código con todas las imágenes de entrenamiento que se extraen de las carpetas de train. Para los atributos de cada imagen utilizamos cada pixel como atributo para lograr esto hicimos un reshape de las imágenes y seteamos toda la información en nuestro outputArray en la función computeAttributes.

```
#Compute shape attributes:
def computeAttributes(numberOfSamples,features):
    #Prepare the out array:
   outArrayTrain = np.zeros((numberOfSamples, features+1), dtype=np.float32)
   trainFiles = os.listdir(pathTrain)
   imageWidth = 70
   imageHeight = imageWidth
   counter = 0
   for k in trainFiles:
       imagePaths = os.listdir(pathTrain + k)
       #print(imagePaths)
       for imagePath in imagePaths:
           # Read the image:
           outImage = cv2.imread(pathTrain+k+"/"+imagePath)
           #showImage("Train Image: ", outImage)
           # Convert BGR to grayscale:
           outImage = cv2.cvtColor(outImage,cv2.COLOR_BGR2GRAY)
           # Convert grayscale to binary:
           _,outImage = cv2.threshold(outImage,0, 255,cv2.THRESH_OTSU)
           #showImage("Train Image: ", outImage)
           #Store the features in the out array:
           outArrayTrain[counter][0] = int(k)
           # Numpy reshape: numpy.reshape(a, newshape) -> a: array_like,
newshape: int or tuple of ints
           testReshape =  outImage.reshape(-1,
imageWidth*imageHeight).astype(np.float32)
           outArrayTrain[counter][1:] = testReshape
           counter += 1
   return outArrayTrain


#Cerate the train matrix:
trainMatrix = computeAttributes(lenTrainImages,features=4900)
#Get train data and labels:
(trSamples, attributes) = trainMatrix.shape
#Get train labels (first column):
trainLabels = trainMatrix[0:trSamples, 0:1].astype(np.int32)
# Get train data (second column to last):
trainData = trainMatrix[0:trSamples, 1:attributes].astype(np.float32)

# Create the SVM:
SVM = cv2.ml.SVM_create()
# Set hyperparameters:
SVM.setKernel(cv2.ml.SVM_LINEAR)  # Sets the SVM kernel, this is a linear
kernel
```

```
SVM.setType(cv2.ml.SVM_NU_SVC)  # Sets the SVM type, this is a "Smooth"
Classifier
SVM.setNu(0.1)  # Sets the "smoothness" of the decision boundary, values:
[0.0 - 1.0]

SVM.setTermCriteria((cv2.TERM_CRITERIA_COUNT, 25, 1.e-01))
print("SVM parameters set...")
SVM.train(trainData, cv2.ml.ROW_SAMPLE, trainLabels)
# Check if SVM is ready to predict:
svmTrained = SVM.isTrained()
if svmTrained:
   print("SVM has been trained, ready to test. ")
else:
   print("Warning: SVM IS NOT TRAINED!")
```

Fase 4. Class Out

En esta parte regresando a donde dejamos el código con la imagen canvas que son las imágenes de salida, ya obtuvimos la predicción de cada imagen del sudoku de número que le enviamos al igual haciéndole un reshape a esta imagen y realizamos la matriz de salida con los valores que predice nuestra SVM.

```
                # Numpy reshape: numpy.reshape(a, newshape) -> a: array_like,
newshape: int or tuple of ints
                testShaped =
canvas.reshape(-1,sampleHeight*sampleWidth).astype(np.float32)

                svmResult = SVM.predict(testShaped)
                #print(svmResult)
                #print(svmResult[1])
                #print(int(svmResult[1][0]))
                #print ("Coordenadas en x: ", x, ", Coordenadas en y: ", y)
                #print (boundingRectangle.shape)
                if (y > 0 and  y < 49 ):
                    if ( x > 0 and  x < 49 ):
                        outMatrix[0,0] = int(svmResult[1][0])
                    elif ( x > 50 and  x < 98 ):
                        outMatrix[0,1] = int(svmResult[1][0])
                    elif ( x > 99 and  x < 146 ):
                        outMatrix[0,2] = int(svmResult[1][0])
                    elif ( x > 147 and  x < 195 ):
                        outMatrix[0,3] = int(svmResult[1][0])
                    elif ( x > 196 and  x < 244 ):
                        outMatrix[0,4] = int(svmResult[1][0])
                    elif ( x > 245 and  x < 293 ):
                        outMatrix[0,5] = int(svmResult[1][0])
                    elif ( x > 294 and  x < 342 ):
                        outMatrix[0,6] = int(svmResult[1][0])
                    elif ( x > 343 and  x < 391 ):
                        outMatrix[0,7] = int(svmResult[1][0])
                    elif ( x > 392 and  x < 440 ):
                        outMatrix[0,8] = int(svmResult[1][0])

                elif (y > 50 and  y < 98 ):
                    if ( x > 0 and  x < 49 ):
                        outMatrix[1,0] = int(svmResult[1][0])
                    elif ( x > 50 and  x < 98 ):
```

```
            outMatrix[1,1] = int(svmResult[1][0])
        elif ( x > 99 and  x < 146 ):
            outMatrix[1,2] = int(svmResult[1][0])
        elif ( x > 147 and  x < 195 ):
            outMatrix[1,3] = int(svmResult[1][0])
        elif ( x > 196 and  x < 244 ):
            outMatrix[1,4] = int(svmResult[1][0])
        elif ( x > 245 and  x < 293 ):
            outMatrix[1,5] = int(svmResult[1][0])
        elif ( x > 294 and  x < 342 ):
            outMatrix[1,6] = int(svmResult[1][0])
        elif ( x > 343 and  x < 391 ):
            outMatrix[1,7] = int(svmResult[1][0])
        elif ( x > 392 and  x < 440 ):
            outMatrix[1,8] = int(svmResult[1][0])

    elif (y > 99 and  y < 146 ):
        if ( x > 0 and  x < 49 ):
            outMatrix[2,0] = int(svmResult[1][0])
        elif ( x > 50 and  x < 98 ):
            outMatrix[2,1] = int(svmResult[1][0])
        elif ( x > 99 and  x < 146 ):
            outMatrix[2,2] = int(svmResult[1][0])
        elif ( x > 147 and  x < 195 ):
            outMatrix[2,3] = int(svmResult[1][0])
        elif ( x > 196 and  x < 244 ):
            outMatrix[2,4] = int(svmResult[1][0])
        elif ( x > 245 and  x < 293 ):
            outMatrix[2,5] = int(svmResult[1][0])
        elif ( x > 294 and  x < 342 ):
            outMatrix[2,6] = int(svmResult[1][0])
        elif ( x > 343 and  x < 391 ):
            outMatrix[2,7] = int(svmResult[1][0])
        elif ( x > 392 and  x < 440 ):
            outMatrix[2,8] = int(svmResult[1][0])

    elif (y > 147 and  y < 195 ):
        if ( x > 0 and  x < 49 ):
            outMatrix[3,0] = int(svmResult[1][0])
        elif ( x > 50 and  x < 98 ):
            outMatrix[3,1] = int(svmResult[1][0])
        elif ( x > 99 and  x < 146 ):
            outMatrix[3,2] = int(svmResult[1][0])
        elif ( x > 147 and  x < 195 ):
            outMatrix[3,3] = int(svmResult[1][0])
        elif ( x > 196 and  x < 244 ):
            outMatrix[3,4] = int(svmResult[1][0])
        elif ( x > 245 and  x < 293 ):
            outMatrix[3,5] = int(svmResult[1][0])
        elif ( x > 294 and  x < 342 ):
            outMatrix[3,6] = int(svmResult[1][0])
        elif ( x > 343 and  x < 391 ):
            outMatrix[3,7] = int(svmResult[1][0])
        elif ( x > 392 and  x < 440 ):
            outMatrix[3,8] = int(svmResult[1][0])

    elif (y > 196 and  y < 244 ):
        if ( x > 0 and  x < 49 ):
            outMatrix[4,0] = int(svmResult[1][0])
        elif ( x > 50 and  x < 98 ):
            outMatrix[4,1] = int(svmResult[1][0])
```

```
    elif ( x > 99 and  x < 146 ):
        outMatrix[4,2] = int(svmResult[1][0])
    elif ( x > 147 and  x < 195 ):
        outMatrix[4,3] = int(svmResult[1][0])
    elif ( x > 196 and  x < 244 ):
        outMatrix[4,4] = int(svmResult[1][0])
    elif ( x > 245 and  x < 293 ):
        outMatrix[4,5] = int(svmResult[1][0])
    elif ( x > 294 and  x < 342 ):
        outMatrix[4,6] = int(svmResult[1][0])
    elif ( x > 343 and  x < 391 ):
        outMatrix[4,7] = int(svmResult[1][0])
    elif ( x > 392 and  x < 440 ):
        outMatrix[4,8] = int(svmResult[1][0])

elif (y > 245 and  y < 293 ):
    if ( x > 0 and  x < 49 ):
        outMatrix[5,0] = int(svmResult[1][0])
    elif ( x > 50 and  x < 98 ):
        outMatrix[5,1] = int(svmResult[1][0])
    elif ( x > 99 and  x < 146 ):
        outMatrix[5,2] = int(svmResult[1][0])
    elif ( x > 147 and  x < 195 ):
        outMatrix[5,3] = int(svmResult[1][0])
    elif ( x > 196 and  x < 244 ):
        outMatrix[5,4] = int(svmResult[1][0])
    elif ( x > 245 and  x < 293 ):
        outMatrix[5,5] = int(svmResult[1][0])
    elif ( x > 294 and  x < 342 ):
        outMatrix[5,6] = int(svmResult[1][0])
    elif ( x > 343 and  x < 391 ):
        outMatrix[5,7] = int(svmResult[1][0])
    elif ( x > 392 and  x < 440 ):
        outMatrix[5,8] = int(svmResult[1][0])

elif (y > 294 and  y < 342 ):
    if ( x > 0 and  x < 49 ):
        outMatrix[6,0] = int(svmResult[1][0])
    elif ( x > 50 and  x < 98 ):
        outMatrix[6,1] = int(svmResult[1][0])
    elif ( x > 99 and  x < 146 ):
        outMatrix[6,2] = int(svmResult[1][0])
    elif ( x > 147 and  x < 195 ):
        outMatrix[6,3] = int(svmResult[1][0])
    elif ( x > 196 and  x < 244 ):
        outMatrix[6,4] = int(svmResult[1][0])
    elif ( x > 245 and  x < 293 ):
        outMatrix[6,5] = int(svmResult[1][0])
    elif ( x > 294 and  x < 342 ):
        outMatrix[6,6] = int(svmResult[1][0])
    elif ( x > 343 and  x < 391 ):
        outMatrix[6,7] = int(svmResult[1][0])
    elif ( x > 392 and  x < 440 ):
        outMatrix[6,8] = int(svmResult[1][0])

elif (y > 343 and  y < 391 ):
    if ( x > 0 and  x < 49 ):
        outMatrix[7,0] = int(svmResult[1][0])
    elif ( x > 50 and  x < 98 ):
        outMatrix[7,1] = int(svmResult[1][0])
    elif ( x > 99 and  x < 146 ):
```

```
            outMatrix[7,2] = int(svmResult[1][0])
        elif ( x > 147 and  x < 195 ):
            outMatrix[7,3] = int(svmResult[1][0])
        elif ( x > 196 and  x < 244 ):
            outMatrix[7,4] = int(svmResult[1][0])
        elif ( x > 245 and  x < 293 ):
            outMatrix[7,5] = int(svmResult[1][0])
        elif ( x > 294 and  x < 342 ):
            outMatrix[7,6] = int(svmResult[1][0])
        elif ( x > 343 and  x < 391 ):
            outMatrix[7,7] = int(svmResult[1][0])
        elif ( x > 392 and  x < 440 ):
            outMatrix[7,8] = int(svmResult[1][0])

    elif (y > 392 and  y < 440 ):
        if ( x > 0 and  x < 49 ):
            outMatrix[8,0] = int(svmResult[1][0])
        elif ( x > 50 and  x < 98 ):
            outMatrix[8,1] = int(svmResult[1][0])
        elif ( x > 99 and  x < 146 ):
            outMatrix[8,2] = int(svmResult[1][0])
        elif ( x > 147 and  x < 195 ):
            outMatrix[8,3] = int(svmResult[1][0])
        elif ( x > 196 and  x < 244 ):
            outMatrix[8,4] = int(svmResult[1][0])
        elif ( x > 245 and  x < 293 ):
            outMatrix[8,5] = int(svmResult[1][0])
        elif ( x > 294 and  x < 342 ):
            outMatrix[8,6] = int(svmResult[1][0])
        elif ( x > 343 and  x < 391 ):
            outMatrix[8,7] = int(svmResult[1][0])
        elif ( x > 392 and  x < 440 ):
            outMatrix[8,8] = int(svmResult[1][0])
#Printing the format of the matrix
print ("Matriz de Input Image" + fileName)
for x in outMatrix:
    for y in x:
        print ("["+ str(y)+"] " , end="")
    print()
#print (outMatrix)
outMatrix = np.zeros((9, 9), type(np.uint8))
```