Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Detección de personas usando características HOG

Informe de Tarea 4

Alumno: José Rubio
Profesor: Javier Ruiz del Solar
Auxiliar: Patricio Loncomilla
Ayudantes: Juan Pablo Cáceres B.
Javier Smith D.
Hans Starke D.
José Villagrán E.

Fecha de realización: 23 de noviembre de 2020
Fecha de entrega: 23 de noviembre de 2020
Santiago, Chile

Índice de Contenidos

1. Introducción. **1**

2. Desarrollo. **2**
2.1. Imágenes. . . . 2
2.2. Conversión de imagen a escala de grises de 64x128. . . . 3
2.3. Gradientes de la imagen. . . . 3
2.4. Histogram of Oriented Gradients. . . . 4
2.4.1. Separación en bloques. . . . 4
2.4.2. Cálculo de vectores HOG's. . . . 5
2.5. Block Normalization . . . 8
2.6. Extracción HOG a las imágenes. . . . 9
2.7. SVM con GridSearch. . . . 10
2.8. Random Forest . . . 11
2.9. Resultados y Matrices de confusión. . . . 11
2.10. Análisis de Resultados. . . . 12

3. Conclusiones. **13**

Índice de Figuras

1. Histograma con los valores de los bines para la extracción de los HOG's . . . 5
2. Distribución de una magnitud ubicada según su dirección . . . 7
3. Distribución de una magnitud ubicada según su dirección . . . 8
4. Matrices de confusión para los SVM normal **a)**, y tuneado **b)**. . . . 12
5. Matriz de confusión para el modelo Random Forest. . . . 12

Índice de Tablas

1. Rendimientos de los detectores con los datos de pruebas normalizados. . . . 11

Índice de Códigos

1. Carga de imágenes. . . . 2
2. Transformación de una imagen a escala de grises y redimensionamiento. . . . 3
3. Gradientes de dirección y magnitud de una imagen. . . . 3
4. Función auxiliar para el HOG que separa una imagen en bloques. . . . 4
5. Sistema extractor de características tipo HOG . . . 5
6. Función normalizadora de HOG's por bloques. . . . 8
7. Función extractora de HOG's para imágenes almacenadas en un diccionario. . . . 9
8. Extracción de HOG's de los diccionarios. . . . 9

9. Creación de conjuntos de entrenamiento y validación. 10
10. Llamado y entrenamiento de SVM con GridSearch 10
11. GridSearch para SVM 11
12. Clasificador Random Forest 11

1. Introducción.

La visión computacional posee una gran cantidad de aplicaciones, donde se necesite realizar algún reconocimiento sobre objetos se pueden aplicar los distintos tipos de algoritmos que están presentes dentro de este campo, lo que se traduce en usos en una enorme cantidad de industrias.

Quizás el trabajo por excelencia de estos algoritmos consiste en poder detectar y clasificar diferentes tipos de objetos, esto eleva la importancia de estos sistemas, además de convertirse una herramienta excelente para observar el mundo de una forma mucho más potente. Si hay algo importante de detectar por parte de un sistema que forma parte de una máquina más potente, este correspondería a las personas, ya que la presencia de una de estas podría alterar el funcionamiento de potentes máquinas de todo tipo (brazos robóticos, cortadoras automáticas, vehículos automáticos, etc.). Debido a esto en este trabajo se presentará un algoritmo capaz de detectar la presencia de personas mediante el uso de características **HOG**, que son las siglas de **H**istogram of **O**riented **G**radients (Histograma de orientaciones del gradiente), el cual corresponde a un descriptor de características para la detección de objetos sumamente preciso al momento de realizar trabajos de detección de personas.

En este informe se mostrarán clasificadores que fueron entrenados con características tipo HOG para realizar detecciones de personas. Como al momento de desarrollar estos sistemas, la parte más importante corresponde a la selección de características apropiadas, por lo que el principal enfoque de este informe consiste en explicar el método por el cual se realizó la extracción de características HOG de las imágenes. En este informe se tendrá la siguiente información:

- Se mostrarán las base del extractor de características partiendo de un algoritmo para cargar las imágenes en un diccionario.
- Se mostrará una función que convierte una imagen de 3 canales a escala de grises, ademas de redimensionar el tamaño de esta a una escala de 64x128, independiente del tamaño original.
- Se mostrará un algoritmo que extrae los gradientes de magnitud y dirección de una imagen de 64x128.
- Procesando la respuesta anterior, se apreciará la implementación de una función que separa una imagen en una serie de bloques, y calcula el HOG para cada bloque, donde entrega un arreglo que contiene el histograma de cada bloque.
- Teniendo el arreglo anterior, se mostrará una función que normaliza los histogramas obtenidos tomando bloques de 2x2 para tener un margen superior al momento de realizar la normalización, cabe destacar que este sistema realiza este proceso con bloques de 2x2 usando traslape, donde la salida de este corresponde a un solo vector que contiene los valores normalizados por cada bloque.
- Finalmente, se realizará este proceso para todas las imágenes disponibles, con lo que se desarrollará una base de datos para entrenar un detector de personas mediante el uso de *Support Vector Machine* y *Random Forest*

2. Desarrollo.

2.1. Imágenes.

Para el desarrollo de este trabajo se contó una serie de imágenes para probar el algoritmo. Esta base de datos de **370 imágenes** se separa en las siguientes categorías:

- 185 imágenes con personas.
- 185 imágenes sin personas, (123 figuras de autos, y 62 de sillas).

De todas estas imágenes, el 70 % de ellas serán utilizadas para el conjunto de entrenamiento, es decir 259 imágenes, quedando 111 imágenes para el conjunto de prueba. Estas imágenes fueron guardas en un diccionario siguiendo la siguiente implementación:

```
# Diccionario para inicializar los las variables que contedran la información de las imagenes

images = dict()

q_images = [123, 62, 185]   # Cantidad de imagenes almacenadas en el drive
keys_images, dir_images = ['cars', 'chairs', 'person'], ['/car_side/', '/chair/', '/pedestrian/'] #
    ↪ Llaves de los diccionarios y nombre de carpetas en el drive
dir_drive = '/content/drive/My Drive/T4_Imagenes'    # Directorio del drive

# q_cars, q_chairs, q_persons = 123, 62, 185

for key, dir, q in zip(keys_images, dir_images, q_images):
  directory = dir_drive  + dir
  images[key] = dict()
  if key != 'person':
    for i in range(1, q+1): # Recorriendo en la cantidad de imagenes de las carpetas.
      if i < 10:
        images[key][i] = cv2.imread(directory + 'image_000' + str(i) + '.jpg')
      elif (10 <= i) and (i < 100):
        images[key][i] = cv2.imread(directory + 'image_00' + str(i) + '.jpg')
      elif i >= 100:
        images[key][i] = cv2.imread(directory + 'image_0' + str(i) + '.jpg')
  else:
    for i in range(1, q+1):
      images[key][i] = cv2.imread(directory + str(i) + '.png')
```

Código 1: Carga de imágenes.

Al ejecutar el código anterior, las imágenes se guardan en el diccionario `images`, donde las llaves `'cars','chairs','person'` sirven para acceder a los valores numéricos de las imágenes de los autos, sillas y personas respectivamente. Una vez finalizado el proceso se seleccionará el 70 % de los datos de manera aleatoria para conformar el conjunto de entrenamiento y el resto para el conjunto de prueba.

2.2. Conversión de imagen a escala de grises de 64x128.

El algoritmo para extraer las características HOG de una imagen solo es posible si el ancho y alto de una imagen están en razón de 1:2, por ende se implementó la siguiente función:

```
def gray_image(image):
 return cv2.resize(image[:,:,0],(64,128)).astype(np.float32)
```

Código 2: Transformación de una imagen a escala de grises y redimensionamiento.

Esta función recibe una imagen en 3 canales de cualquier dimensión, donde selecciona el primer canal y luego realiza el redimensionamiento para obtener una matriz de 128 filas y 64 columnas, que corresponden al alto y ancho de una imagen.

2.3. Gradientes de la imagen.

Teniendo las imágenes con las dimensiones apropiadas, lo siguiente es obtener los gradientes de dirección y magnitud, donde se vea representado la intensidad de los pixeles en la imagen junto con la dirección en ángulos a la cual esta magnitud afecta, Para obtener estos valores se pueden seguir las siguientes ecuaciones:

$$\text{mag}(x, y) = \sqrt{g_x^2 + g_y^2} \tag{1}$$

$$\text{dir}(x, y) = \tan^{-1}(\frac{g_y}{g_x}) \tag{2}$$

. Esto se le aplica a cada pixel de la imagen. Para implementar el proceso completo se utilizó el siguiente bloque:

```
def gradients_for_HOG(image):
 gx, gy = cv2.Sobel(image, cv2.CV_32F, 0, 1), cv2.Sobel(image, cv2.CV_32F, 1, 0)
 mag, angle = cv2.cartToPolar(gx, abs(gy), angleInDegrees=True)
 return mag, angle

def gradients_manual(image):
 gx, gy = cv2.Sobel(image, cv2.CV_32F, 0, 1), cv2.Sobel(image, cv2.CV_32F, 1, 0)
 mag = np.sqrt(gx**2 + gy**2)
 ang = np.zeros(image.shape)
 for r in range(image.shape[0]):       # Recorriendo las filas de las imágenes.
  for c in range(image.shape[1]):      # Recorriendo las columnas
   gx_value, gy_value = gx[r,c], abs(gy[r,c])
   if gx_value == 0 and gy_value == 0:
    ang[r,c] = 0
   elif gx_value == 0 and gy_value > 0:
    ang[r,c] = 90
   elif gx_value < 0 and gy_value == 0:
    ang[r,c] = 180
   else:
    degrees = np.degrees(np.arctan(gy_value/gx_value))
    if degrees < 0:
```

```
        degrees = 180 + degrees
        ang[r,c] = degrees
      else:
        ang[r,c] = degrees
  return mag, ang.astype(np.float32)
```

Código 3: Gradientes de dirección y magnitud de una imagen.

Como se aprecia en el bloque anterior, el cálculo de estos gradientes se desarrolló de 2 formas, ambas funciones utilizan `cv2.Sobel` para extraer los gradientes en el eje x e y, pero la función `gradients_for_HOG` utiliza la función `cartToPolar` de `cv2` para obtener las magnitudes y las direcciones directamente en grados. Cabe destacar que las direcciones obtenidas están todas entre 0 y 180 grados, ya que se impuso $g_y > 0$, por otro lado la función `gradients_manual` realiza el mismo proceso que el anterior, obteniendo al valor de las magnitudes siguiendo la ecuación (1). Sin embargo para obtener el valor de las direcciones positivas, la función se coloca en los casos límites para poder imponer los valores correctos como 0°, 90°, 180°. La dirección se calcula mediante la ecuación (2), donde al final de este cálculo el valor se pasa a grados (al aplicar la función `arctan` entrega los grados en radianes), si el resultado es negativo, entonces se suman 180°, ya que lo que nos interesa es la dirección no el sentido.

2.4. Histogram of Oriented Gradients.

Tras la obtención de los gradientes, entonces se debe realizar la extracción de los **HOG's**, sin embargo antes de realizar la extracción se debe separar las imágenes de los gradientes en bloques, ya que la salida de este sistema corresponde a un arreglo de dimensiones (8,16,9).

2.4.1. Separación en bloques.

Para obtener los bloques, se utilizó la siguiente función

```
def block_separation(image):
  im_gray = image.copy()
  image_cells = dict()
  k = 0
  rows_division, cols_division = np.arange(0, 128, 8).astype(np.int32), np.arange(0, 64, 8)
  for r_d in rows_division:
    image_cells[k] = dict()
    i = 0
    for c_d in cols_division:
      if r_d == 0:
        if c_d == 0:
          image_cells[k][i] = im_gray[:r_d+8, :c_d+8]
          i += 1
        elif c_d == 56:
          image_cells[k][i] = im_gray[:r_d+8, c_d:]
          i += 1
        else:
          image_cells[k][i] = im_gray[:r_d+8,c_d:c_d+8]
          i += 1
      elif r_d == 120:
```

```
      if c_d == 0:
        image_cells[k][i] = im_gray[r_d:, :c_d+8]
        i += 1
      elif c_d == 56:
        image_cells[k][i] = im_gray[r_d:, c_d:]
        i += 1
      else:
        image_cells[k][i] = im_gray[r_d:,c_d:c_d+8]
        i += 1
    else:
      if c_d == 0:
        image_cells[k][i] = im_gray[r_d:r_d+8, :c_d+8]
        i += 1
      elif c_d == 56:
        image_cells[k][i] = im_gray[r_d:r_d+8, c_d:]
        i += 1
      else:
        image_cells[k][i] = im_gray[r_d:r_d+8,c_d:c_d+8]
        i += 1
  k += 1
return image_cells
```

Código 4: Función auxiliar para el HOG que separa una imagen en bloques.

El código anterior guarda en un diccionario partes de las imágenes, para este trabajo se utilizaron separaciones de 8 pixeles, por ende cada diccionario contenía información de 8x8 pixeles. Usando esta función auxiliar entonces se realizó la extracción de los HOG's.

2.4.2. Cálculo de vectores HOG's.

La obtención de los valores de los bines de los histogramas no fue trivial, las magnitudes obtenidas fueron colocadas en bines del siguiente histograma:

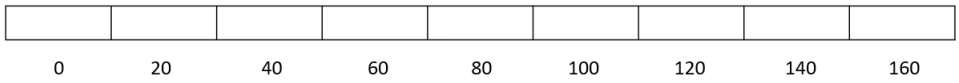


Figura 1: Histograma con los valores de los bines para la extracción de los HOG's

Con esto, los valores colocados en los bines de este histograma fueron calculados con el siguiente bloque:

```
def HOG_features(grad_mag, grad_ang, contribution):
  mag_cells, ang_cells = block_separation(grad_mag), block_separation(grad_ang) # Separación
     ↪ de las imágenes en las celdas.
  HOG_array = np.zeros([16, 8, 9], dtype=np.int64)

  # Configurando la contribución de las direcciones, contribución total

  if contribution == 'full':
    for k in range(16):
```

```
    for i in range(8):
      mag_block, dir_block = mag_cells[k][i], ang_cells[k][i]
      im_size = mag_block.shape
      hog_vector = np.zeros(9)
      for r in range(im_size[0]):
        for c in range(im_size[1]):
          val_mag, val_dir = mag_block[r,c], dir_block[r,c]
          if val_dir < 20 or val_dir == 180:
            hog_vector[0] += val_mag
          elif val_dir >= 20 and val_dir < 40:
            hog_vector[1] += val_mag
          elif val_dir >= 40 and val_dir < 60:
            hog_vector[2] += val_mag
          elif val_dir >= 60 and val_dir < 80:
            hog_vector[3] += val_mag
          elif val_dir >= 80 and val_dir < 100:
            hog_vector[4] += val_mag
          elif val_dir >= 100 and val_dir < 120:
            hog_vector[5] += val_mag
          elif val_dir >= 120 and val_dir < 140:
            hog_vector[6] += val_mag
          elif val_dir >= 140 and val_dir < 160:
            hog_vector[7] += val_mag
          elif val_dir >= 160 and val_dir < 180:
            hog_vector[8] += val_mag
      HOG_array[k,i,:] = hog_vector
      # print(HOG_array[k,i,:])
  # print(HOG_array)
  return HOG_array

# Contribuciones parciales.

elif contribution == 'partial':
  for k in range(16):
    for i in range(8):
      mag_block, dir_block = mag_cells[k][i], ang_cells[k][i]
      im_size = mag_block.shape
      hog_vector = np.zeros(9)
      for r in range(im_size[0]):
        for c in range(im_size[1]):
          val_mag, val_dir = mag_block[r,c], dir_block[r,c]
          if val_dir < 20:
            prop_contribution_1, prop_contribution_2 = (val_dir - 0)/20, (20 - val_dir)/20
            hog_vector[0], hog_vector[1] =  hog_vector[0] + prop_contribution_2*val_mag,
    ↪ hog_vector[1] + prop_contribution_1*val_mag
          elif val_dir >= 20 and val_dir < 40:
            prop_contribution_1, prop_contribution_2 = (val_dir - 20)/20, (40 - val_dir)/20
            hog_vector[1], hog_vector[2] =  hog_vector[1] + prop_contribution_2*val_mag,
    ↪ hog_vector[2] + prop_contribution_1*val_mag
          elif val_dir >= 40 and val_dir < 60:
            prop_contribution_1, prop_contribution_2 = (val_dir - 40)/20, (60 - val_dir)/20
```

```
58            hog_vector[2], hog_vector[3] =  hog_vector[2] + prop_contribution_2*val_mag,
    ↪ hog_vector[3] + prop_contribution_1*val_mag
59          elif val_dir >= 60 and val_dir < 80:
60            prop_contribution_1, prop_contribution_2 = (val_dir - 60)/20, (80 - val_dir)/20
61            hog_vector[3], hog_vector[4] =  hog_vector[3] + prop_contribution_2*val_mag,
    ↪ hog_vector[4] + prop_contribution_1*val_mag
62          elif val_dir >= 80 and val_dir < 100:
63            prop_contribution_1, prop_contribution_2 = (val_dir - 80)/20, (100 - val_dir)/20
64            hog_vector[4], hog_vector[5] =  hog_vector[4] + prop_contribution_2*val_mag,
    ↪ hog_vector[5] + prop_contribution_1*val_mag
65          elif val_dir >= 100 and val_dir < 120:
66            prop_contribution_1, prop_contribution_2 = (val_dir - 100)/20, (120 - val_dir)/20
67            hog_vector[5], hog_vector[6] =  hog_vector[5] + prop_contribution_2*val_mag,
    ↪ hog_vector[6] + prop_contribution_1*val_mag
68          elif val_dir >= 120 and val_dir < 140:
69            prop_contribution_1, prop_contribution_2 = (val_dir - 120)/20, (140 - val_dir)/20
70            hog_vector[6], hog_vector[7] =  hog_vector[6] + prop_contribution_2*val_mag,
    ↪ hog_vector[7] + prop_contribution_1*val_mag
71          elif val_dir >= 140 and val_dir < 160:
72            prop_contribution_1, prop_contribution_2 = (val_dir - 140)/20, (160 - val_dir)/20
73            hog_vector[7], hog_vector[8] =  hog_vector[7] + prop_contribution_2*val_mag,
    ↪ hog_vector[8] + prop_contribution_1*val_mag
74          elif val_dir >= 160 and val_dir <= 180:
75            prop_contribution_1, prop_contribution_2 = (val_dir - 160)/20, (180 - val_dir)/20
76            hog_vector[8], hog_vector[0] =  hog_vector[8] + prop_contribution_2*val_mag,
    ↪ hog_vector[0] + prop_contribution_1*val_mag
77        HOG_array[k,i,:] = hog_vector
78    return HOG_array
```

Código 5: Sistema extractor de características tipo HOG

La función anterior posee 2 tipos de salidas, esta depende del tipo de contribución que se seleccione en la entrada de la función.

- **Contribución Total:** Este se selecciona al colocar el *string* 'full' en el argumento `contribution`. Al seleccionar este método entonces los valores de la magnitud es distribuido en los bines cuyo valor de dirección es inferior a la dirección asociada. Esto se puede apreciar de mejor manera en la siguiente figura.

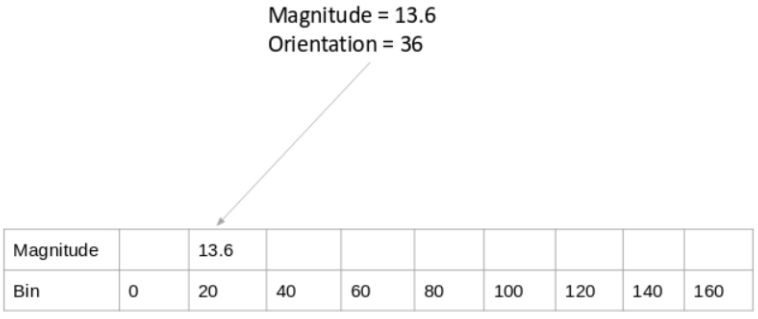


Figura 2: Distribución de una magnitud ubicada según su dirección

Aplicando esto para todos los pixeles y sumando los valores de las magnitudes para cada bin se forma un histograma, por lo que el proceso se repite para cada bloque de la imagen y finalmente se guardan estos valores en el arreglo `HOG_array` entregándolo como salida de esta función.

- **Contribución parcial:** Al seleccionar el *string* `'partial'` se escoge este método para construir los histogramas. Este sistema es muy similar al anterior, sin embargo las magnitudes son ponderadas dependiendo del valor de la dirección y son repartidas entre los bines que se encuentra. Esta ponderación sigue las siguientes relaciones:

$$B_i = B_i + \frac{D_{k,j} - D_i}{20} \tag{3}$$

$$B_{i+1} = B_{i+1} + \frac{D_{i+1} - D_{k,j}}{20} \tag{4}$$

Donde B_i representa el valor de la magnitud actual ubicado en la posición i del histograma, D_i corresponde al límite inferior del bin donde se ubica $D_{k,j}$, el cual contiene la dirección del pixel k, j. De la misma forma B_{i+1} y D_{i+1} representan la magnitud en la posición i+1 y el limite superior del bin respectivamente. Este proceso se aprecia mejor en la siguiente figura:

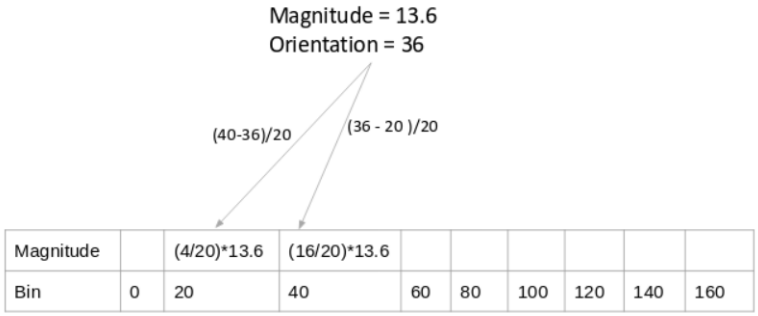


Figura 3: Distribución de una magnitud ubicada según su dirección

Igualmente que en el método anterior, esta operación se aplica a todos los pixeles, y se van sumando los valores ponderados para cada bin del histograma, donde se guardan estos valores en el arreglo `HOG_array` para que sea la salida mediante esta condición.

2.5. Block Normalization

La salida de la función anterior corresponde a un arreglo de dimensiones (8,16,9) que contiene los HOG's separados por bloques, entonces antes de obtener el arreglo final se debe realizar una normalización por bloques con traslape. Para ello se implementó la siguiente función:

```
def block_normalization(hog_f):
  h, w, d = hog_f.shape
  hog_vector_norm = np.array([])
  for r in range(h-1):
    for c in range(w-1):
      vector_norm = np.array([])
      v_1, v_2, v_3, v_4 = hog_f[r,c,:], hog_f[r,c+1,:], hog_f[r+1,c,:], hog_f[r+1,c+1,:]
```

```
    vector_block = np.array([v_1,v_2,v_3,v_4])
    vector_norm = np.array([])
    for v in vector_block:
      vector_norm = np.append(vector_norm, v)
    block_max, block_min = np.max(vector_norm), np.min(vector_norm)
    if block_max == 0:
      hog_vector_norm = np.append(hog_vector_norm, vector_norm)
    else:
      max_array, min_array = np.ones(len(vector_norm))*block_max, np.ones(len(vector_norm))*
    ↪ block_min
      vector_norm = (vector_norm - min_array)/(max_array - min_array)
      hog_vector_norm = np.append(hog_vector_norm, vector_norm)
 return hog_vector_norm.reshape(1,len(hog_vector_norm))
```

Código 6: Función normalizadora de HOG's por bloques.

Esta función realiza un recorrido el arreglo original de (8,16,9) donde concatena 4 histogramas, si el bloque esta en la posición (i, j), concatena el histograma de esa posición junto con su vecino derecho $(i+1, j)$, inferior $(i, k+1)$ e inferior derecho $(i+1, j+1)$, donde una vez teniendo este arreglo entonces se normalizaba y se añadia a la variable **hog_norm** que iba conteniendo todos los vectores HOG'S normalizados. Este proceso se hizo para todos los bloques realizando un traslape, por lo que el vector resultante consistió en un arreglo de dimensiones (1,3780).

2.6. Extracción HOG a las imágenes.

Una vez diseñado y probado el sistema con casos puntuales, se debió procesar todas las imágenes con este sistema, como la información de las imágenes estaban guardadas en diccionarios, entonces se definió la siguiente función para realizar la extracción de los HOG's:

```
def hog_dic(Dict, method):
 q_images = len(Dict)
 hog_features = np.zeros([1,3780])
 for k in range(1, q_images+1):
   gray_im = gray_image(Dict[k])
   im_gradients_mag, im_gradients_ang = gradients_manual(gray_im)
   hog_image = HOG_features(im_gradients_mag, im_gradients_ang, method)
   hog_block_normalization = block_normalization(hog_image)
   hog_features = np.vstack([hog_features, hog_block_normalization])
 hog_features = np.delete(hog_features, 0, 0)
 return hog_features
```

Código 7: Función extractora de HOG's para imágenes almacenadas en un diccionario.

La salida de esta función corresponde a una matriz de (tamaño del diccionario, 3780), donde cada fila corresponde al HOG de una imagen del diccionario. Aplicando esta función a los 3 diccionarios se obtienen las características HOG de toda la base de datos disponible.

```
hog_cars = hog_dic(images['cars'],'partial')
hog_chairs = hog_dic(images['chairs'],'partial')
```

```
hog_person = hog_dic(images['person'],'partial')
```

Código 8: Extracción de HOG's de los diccionarios.

Donde se utilizó la función `gradients_manual` y el método `'partial'` para la extracción, este cómputo tardó aproximadamente 2 minutos, y una vez calculado se ejecutó el siguiente bloque:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

no_person_data = np.vstack([hog_cars, hog_chairs])
label_no_person, label_person = np.zeros(no_person_data.shape[0]), np.ones(hog_person.shape[0])
data_np = np.c_[no_person_data, label_no_person]
data_p = np.c_[hog_person, label_person]     se concatenan las personas con la clase creada.
data_total = np.vstack([data_np, data_p])

X_data, y_data = data_total[:,:-1], data_total[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, train_size=0.70)

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Código 9: Creación de conjuntos de entrenamiento y validación.

Donde este bloque concatena los HOG's de las imagenes de las sillas y autos para crear un dataset que contenga todas los HOG's de las imagenes que no tengan personas, tras esto se crean 2 arreglos que contienen ceros y unos solamente, donde el tamaño de estos arreglos son los largos del del `dataset` y `hog_person` respectivamente. El valor de estos arreglos corresponde a las clases de las imágenes donde se asigna el valor 0 a las imagenes con solo objetos y 1 a las imágenes que contienen personas.

Desarrollando lo anterior, entonces toda esta información se concatena formando la matriz `data_total` la cual posee todos los HOG's y las clases asociadas a estos, por lo que mediante el uso de la función `train_test_split` se separa en conjunto de entrenamiento y prueba, tomando el 70 % de los datos para el conjunto de entrenamiento, donde además esta función desordena los datos para que el conjunto de entrenamiento contenga datos seleccionados al azar, de igual manera para el conjunto de prueba, donde una vez obtenidos estos conjuntos se normalizan mediante la función .

2.7. SVM con GridSearch.

Uno de los detectores solicitados correspondió al *Support Vector Machine*, para ello en primer lugar se importaron las funciones de `sklearn`, donde luego inicializando los objetos se entrenan con los datos normalizados por el `StandarScaler`, para esto se utilizó el siguiente bloque:

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

clf = SVC()
```

```
clf.fit(X_train_scaled, y_train)
```

Código 10: Llamado y entrenamiento de SVM con GridSearch

El resultado obtenido de ejecutar este bloque consiste en un objeto entrenado con los datos escalados mediante el `StandarScaker`. Tras esto se mejoró el clasificador SVM mediante una busqueda de grilla utilizando la función `GridSearch`. Esta implementación se realizó mediante el siguiente código:

```
svm_clf = SVC()
grid_kernels = {'kernel':('linear', 'poly', 'rbf', 'sigmoid')}
svm_grid = GridSearchCV(svm_clf, grid_kernels, scoring='accuracy', verbose=1)
svm_grid.fit(X_train_scaled, y_train)
```

Código 11: GridSearch para SVM

Este bloque busca el mejor núcleo para entrenar el modelo realizando varios entrenamientos, donde al final selecciona el núcleo que obtuvo mejor rendimiento.

2.8. Random Forest

El segundo clasificador fue un sistema Random Forest. Para entrenar este modelo se usaron los mismos datos que el SVM, el siguiente bloque fue el que se utilizó para entrenar este clasificador:

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
rf.fit(X_train_scaled, y_train)
```

Código 12: Clasificador Random Forest

Para este clasificador se usaron los hiperparámetros predeterminados, por lo que el entrenamiento de este sistema se hizo rápidamente.

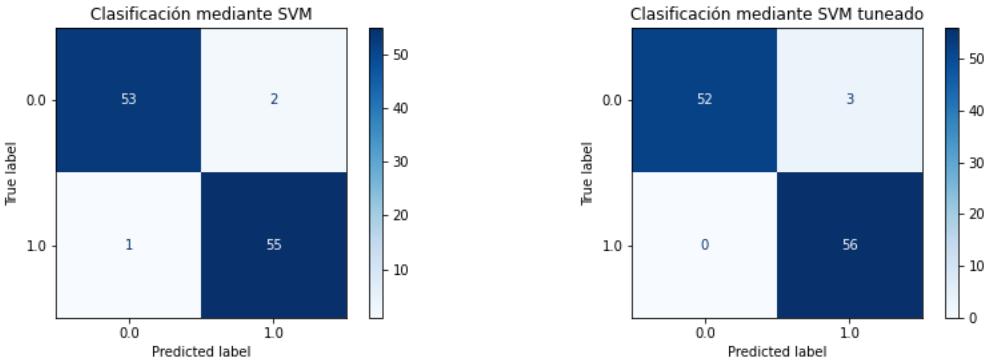
2.9. Resultados y Matrices de confusión.

Para analizar el rendimiento de los clasificadores obtenidos se utilizó la métrica *Accuracy* en el conjunto de prueba, tras aplicarlo se calcularon las matrices de confusión. Los valores obtenidos se encuentran en la siguiente tabla:

Modelo	Accuracy
SVM	0.9727
SVM Tuneado	0.9727
Random Forest	0.9454

Tabla 1: Rendimientos de los detectores con los datos de pruebas normalizados.

Un rendimiento más específico se pueden ver en las siguientes matrices:



(a) (b)

Figura 4: Matrices de confusión para los SVM normal **a)**, y tuneado **b)**.

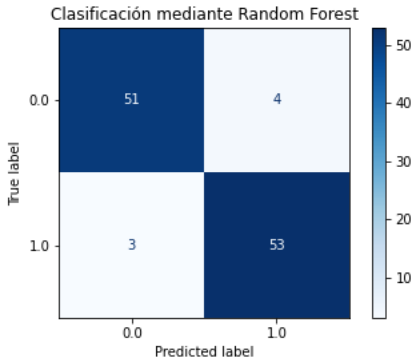


Figura 5: Matriz de confusión para el modelo Random Forest.

2.10. Análisis de Resultados.

Los accuracys obtenidos indican un buen desempeño de todos los sistemas, cabe destacar que el tuneo no afectó en gran medida el rendimiento, ya que se aumenta la detección de personas pero aumentan los falsos negativos. Se tiene que el sistema con menor desempeño corresponde al Random Forest, bajando su accuracy a 0.94. Esto es en parte a que los Random Forest tienen preferencias para problemas multiclase por lo que necesitan una mayor cantidad de datos para funcionar mejor en estas dimensiones. Siguiendo esta línea, los SVM tienen afinidad para los problemas de detección ya que su mecanismo consta en encontrar el hiperplano óptimo de separación de clases, por lo que funciona de mejor forma que otros sistemas.

3. Conclusiones.

Los rendimientos obtenidos indican que este método de extracción de características es confiable para la detección de personas. Donde si bien el Random Forest tuvo un menor acierto posee una gran cantidad de aciertos al momento de realizar esta tarea. Estos comportamientos obedecen a la teoría, ya que estos método de extracción de características fue muy popular antes de la llegada de los modelos entrenados mediante *deep learning*.

La extracción de los HOG's fue la actividad más complicada al desarrollar esta experiencia, se programaron 2 formas para poder colocar la distribución de las magnitudes en los bines donde el rendimiento del sistema mejoró al momento de extraer las características ponderando las magnitudes obtenidas dependiendo del valor de su dirección, sin embargo este sistema esta incompleto debido a que falta añadir las contribuciones espaciales de los pixeles. Sin embargo debido a los altos rendimientos obtenidos con estos sistemas, se esperaba un pequeño aumento en el *accuracy*. Pero dada esta naturaleza se tiene que este proceso permitiría mejorar los resultados obtenidos.

Uno de los puntos destacados al realizar esta tarea corresponde al uso de varias funciones que sintetizan lo desarrollado en tareas pasadas (funciones como `cv2.Sobel o cv2.cartToPolar`), donde se aprecia que los procesos realizados en experiencias anteriores poseen funciones predeterminadas en la librería `cv2`, debido a esto se tiene que para futuras experiencias, donde se realicen trabajos similares se pueden usar estos sistemas para optimizar el rendimiento de los sistemas desarrollados.