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

Tarea 3

Detección de Personas usando características tipo HOG

Integrantes: Joaquín Zepeda
Profesor: Javier Ruiz del Solar
Auxiliar: Patricio Loncomilla
Santiago de Chile

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Índice de Códigos

1. Introducción

El procesamiento digital de imágenes tiene un papel importante en la sociedad, siendo estas una base para el reconocimiento de objetos/personas, diagnosticar condiciones médicas, astronomía, etc. este corresponde a un conjunto de técnicas que permiten cambiar la información que contiene la imagen con el fin de tener una mejor representación de esta y/o resaltar o suprimir alguna característica. Esta disciplina dio sus comienzos cuando comenzaron a digitalizarse las imágenes, es decir, se representaron las imágenes mediante matrices las cuales se guardan en memoria. Dentro de computadoras se pueden operar técnicas matemáticas y cálculos los cuales en la actualidad se puede realizar medianamente rápido, convirtiéndose en una disciplina muy útil en múltiples áreas. En el desarrollo de esta disciplina, aparecieron los puntos de interés en la imagen que tienen como objetivo identificar zonas claves.
El objetivo de esta tarea es diseñar y construir un sistema de detección de personas, que utilice características tipo HOG (Histograms of Oriented Gradients) y clasificadores SVM y Random Forest, esto con el fin de poner en practica los conceptos vistos en clases, poner en practica las habilidades de programación para resolver un problema real y poder apreciar en diferentes situaciones los resultados de estos métodos y de los distintos clasificadores. Para esto se utilizarán 3 tipos de imágenes: imágenes con personas, imágenes con autos e imágenes con sillas. Se analizará tanto el clasificador multiclase de los 3 tipos de objetos como el clasificador binario de personas y no personas.

Esta tarea se desarrolla utilizando el lenguaje de programación Python y además se completan funciones en Cython, lo cual nos permite escribir código en C/C++ desde Python permitiendo tener un programa más rápido, a continuación se muestran y analizan los resultados en la sección de Desarrollo para luego finalizar con las conclusiones.

2. Desarrollo

En esta sección se encuentran los items 1,2,3,4 del enunciado. Se comentaran las funciones más importantes y su funcionamiento:

- Transformación de la imagen: La función transform(img) transforma la imagen a escala de grises, luego la redimensiona al tamaño 64x128 y luego la transforma al tipo np.float32.

```
def transform(img):
    #transforma la imagen a escala de grises, al tipo np.float32
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #redimensiona la imagen a tamaño 64x128
    gray32 = np.float32(gray)
    out = cv2.resize(gray32, (64,128), interpolation = cv2.INTER_AREA)
    return out

```

- HOG(dx,dy)

  Esta función recibe como parametros los gradientes en x e y de la imagen transformada (como se muestra en el diagrama de la figura 2. Estos gradientes se calculan utilizando las funciones de la tarea anterior correspondientes a $grad\_x()$ y $grad\_y()$ que calculan el gradiente en x e y respectivamente. Para esto se utiliza una aproximación de la derivada discreta, la cual se calcula a partir de la **convolución** de la imagen con el kernel que representa la derivada centrada discreta aproximada. Estos kernels corresponden a:

$$Kx = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix} \tag{1}$$

$$Ky = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \tag{2}$$

  Luego de tener los gradientes, se determinan 2 matrices: la matriz de magnitudes y la matriz de direcciones correspondientes a la magnitud y ángulo de los gradientes respectivamente. Esto se calcula a partir de:

$$mag = \sqrt{dx^2 + dy^2} \tag{3}$$

$$dir = \arctan dx/dy \tag{4}$$

  En el código se realizó una función auxiliar que realiza este proceso a partir de los gradientes, en esta para los ángulos mayores o iguales a 180º se le resta 180º pues buscamos que estos queden en el intervalo [0,180)

```
%%cython
import cython
```

```
3   import numpy as np
4   cimport numpy as np
5
6   cpdef MagDir(np.ndarray[np.float32_t, ndim=2] dx,np.ndarray[np.float32_t, ndim=2] dy):
7       cdef int rows,cols, i,j
8       cdef np.ndarray[np.float32_t, ndim=2] magnitud = np.zeros([dx.shape[0], dx.shape[1]],
        ↪ dtype = np.float32)
9       cdef np.ndarray[np.float32_t, ndim=2] angulo = np.zeros([dx.shape[0], dx.shape[1]],
        ↪ dtype = np.float32)
10      rows = dx.shape[0]
11      cols = dx.shape[1]
12      for i in range(rows):
13          for j in range(cols):
14              magnitud[i][j] = np.sqrt(dx[i][j]**2+dy[i][j]**2)
15              # Angulo en grados para simplificar los futuros histogramas
16              angulo[i][j] = np.degrees(np.arctan2(dx[i][j],dy[i][j]))
17
18              #los angulos deben quedar en el rango de 0° a 180°
19              if angulo[i][j]>=180:
20                  angulo[i][j] = angulo[i][j] - 180
21      return magnitud,angulo
22
```

Este proceso se puede observar de forma gráfica en las figuras 1, en estas se muestra la imagen original (a) , luego la imagen transformada (b) y finalmente la imagen de las matrices de modulo (c) y angulo (d).

(a) (b) (c) (d)

Figura 1: Ejemplo preprocesamiento de una imagen.

Se dividen las matrices en 8x16 celdas. Luego en cada celda se calcula un histograma de orientación del gradiente con 9 componentes, es decir, se realiza un histograma el cual separa los ángulos en 9 intervalos cubriendo cada uno 20º, luego por cada pixel de la celda se agregan los valores de la magnitud proporcionales a la distancia del ángulo a los intervalos más cercanos, de esta manera los votos que se agregan son proporcionales a las distancias. Estas interpolaciones se realizaron de 2 formas, con votos solo en la misma celda y con repartición de los votos en las 4 celdas más cercanas. Cada histograma tiene 9 elementos, por lo que el vector que queda luego de realizar los votos es de dimensión 8x16x9. Finalmente se realiza el block normalization el cual toma los histogramas cada 4 celdas con traslape, los concatena y luego los normaliza, quedando 7x15 vectores de 1x36 los cuales se unen al final para formar el vector de características HOG de tamaño 1x3780.

Para realizar la repartición de votos solamente en la celda a la cual pertenece el pixel, se determina el bin al cual pertenece, para luego realizar una votación proporcional a la distancia del ángulo actual con sus respectivos bin's más cercanos. Por otro lado la repartición de votos para las celdas más cercanas, el voto es proporcional a la distancia de los centros de las 3 celdas más cercanas (4 celdas si se cuenta la celda en la que está el pixel), luego a partir de estás distancias se realiza un voto inversamente proporcional a la distancia a los respectivos centros, estos votos se realizan como el primer método donde la votación es proporcional a la distancia del ángulo del pixel que vota con sus respectivos bin's más cercanos. Se realizó una función auxiliar llamada block normalization la cual realiza el proceso que tiene el mismo nombre mencionado anteriormente. En la figura 2 se muestra de forma general el flujo de la función HOG la cual permite extraer las características utilizando ese método.

```
1
2  def HOG(dx,dy,interpolacion=0):
3      """3. Implementar en python una función que, a partir de los gradientes, calcule las
4      características HOG usando 8x16 celdas (la salida debe ser un arreglo de numpy de
5      dimensión 8x16x9).
6      Para esto:
7      1. se determinan 2 matrices, una de la magnitud de las derivadas y otra
8      de las direcciones de estas.
9      2. Se dividen las matrices en 8x16 celdas.
10     3. En cada celda se calcula un histograma de orientación del gradiente
11     con 9 componentes, es decir, se realiza un histograma el cual separa los angulos
12     en 9 intervalos cubriendo cada uno 20°, luego por cada pixel de la celda se agregan
13     los valores de la magnitud proporcionales a la distancia del angulo a los intervalos
14     más cercanos, de esta manera los votos que se agregan son proporcionales a las distancias.
15     4. Luego se reunen los histogramas de orientación del gradiente de cada celda
16     los cuales corresponden a los features que retorna este método. Esto se reune
17     en un arrelgo de numpy de dimensión 8x16x9.
18
19     Para utilizar la interpolacion normal: Interpolacion = 0
20     Para utilizar la interpolacion bilinear: Interpolacion = 1
21     """
22     histograms = np.zeros([16,8,9],np.float32)
23
24     #utilizamos una función auxiliar para determinar la magnitud y el angulo
25     magnitud, angulo = MagDir(dx,dy)
26     rows = dx.shape[0]
```

```
    cols = dx.shape[1]
    ...
    features = block_normalization(histograms)

    return features

```

Código 1: Extracto función HOG

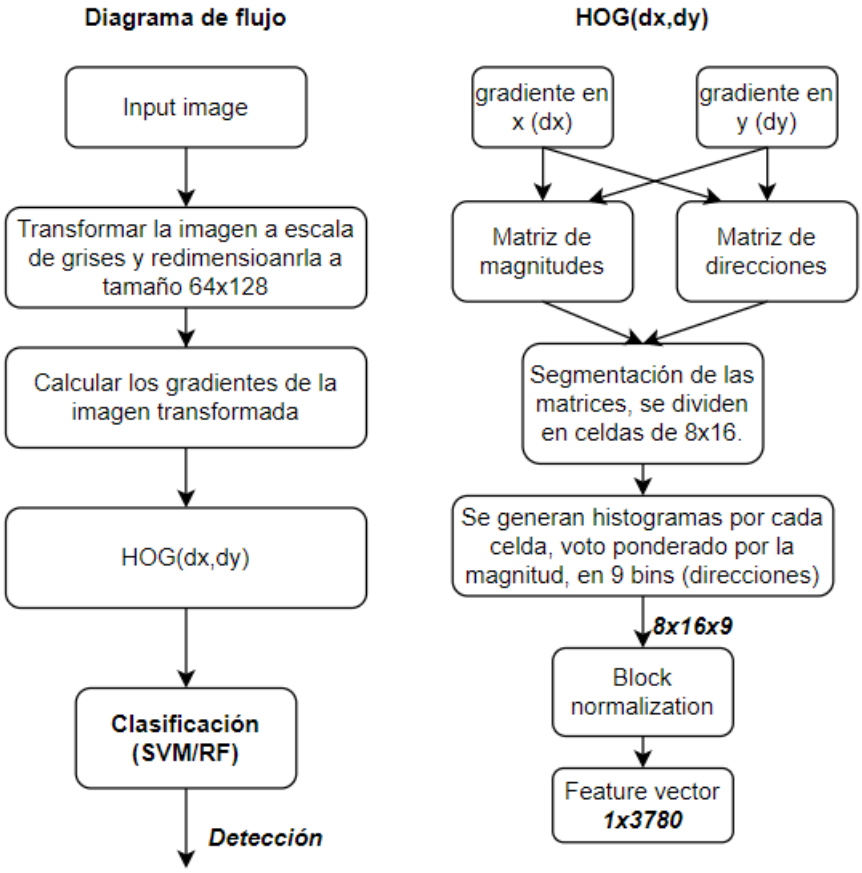


Figura 2: Diagrama de flujo resumido.

En esta sección se describen las demás funciones que representan del item 5 en adelante.

- **Extracción de características Item 5:** Se utilizan la función HOG para extraer las características de todas las imágenes de la base de datos. Además se utiliza una función auxiliar para cargar todas las imágenes, esto se realizó solo para mantener el código más ordenado.

```
    def featureExtractor(data,interpolacion=0):
     """
     Para utilizar la interpolacion normal: Interpolacion = 0
     Para utilizar la interpolacion bilinear: Interpolacion = 1
```

```
    """
    features = []
    for img in data:
        dx = gradx(img)
        dy = grady(img)
        feature = HOG(dx,dy,interpolacion)
        features.append(feature)
    #caracteristicas no normalizadas
    return np.array(features)

```

Dependiendo del tipo de la imagen se le asigno un label, los cuales se muestran en la siguiente tabla:

Tabla 1: Labels para cada clasificación

Clasificación\labels	Cars	Chairs	Pedestrians
Binario	0	0	1
Multiclase	0	1	2

Además de esto, esta base de datos fue separada en 60 % para entrenamiento, 20 % para validación y 20 % para prueba. Esto se muestra a continuación, además se utilizó la función PredefinedSplit para indicar el conjunto de validación a utilizar para encontrar los mejores hiper parámetros.

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import  PredefinedSplit
from sklearn.model_selection import train_test_split
# Conjuntos de Train y Val/Test
X_trainval, X_test, y_trainval, y_test = train_test_split(features, labels, test_size=0.2,
 ↪ shuffle = True, stratify = labels)

X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, shuffle = False,
 ↪ test_size=0.25)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)

X_trainval = scaler.transform(X_trainval)
X_test = scaler.transform(X_test)

split_fold = [-1 for _ in range(int(len(X_trainval)*0.75))]+ [0 for _ in range(int(len(
 ↪ X_trainval)*0.25))]
cv = PredefinedSplit(split_fold)
```

- **Entrenamiento de los clasificadores:** Se utilizarón las implementaciones de Scikit-learn para los clasificadores SVM y Random Forest, en cada uno de ellos se realizó un grid search que se describe a continuación:

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


# Create the parameter grid based on the results of random search

param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': ['scale','auto',1,0.1],
              'kernel': ['linear','rbf', 'poly', 'sigmoid'],
              'class_weight':('balanced',None),
              'decision_function_shape':['ovo','ovr']}
# Instantiate the grid search model
grid = GridSearchCV(SVC(), param_grid,cv=cv, refit=False)


# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [False,True],
    'max_depth': [30, 50, 90, 100],
    'max_features': [5,10,15,20,25,30],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [30,60,100,200]
}
rf = RandomForestClassifier()

# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 3, verbose = 3)
```

- **Hiperparámetros utilizados:**

  Como se observa en estos extractos de código, se exploraron distintos parámetros utilizando gridsearch para estas funciones, finalmente se utilizaron los siguientes parámetros en los clasificadores:

```
    classifier = SVC(C=0.1, class_weight='balanced', decision_function_shape='ovo',
↪ gamma= 'scale',kernel='linear')

```

Código 2: Hiperparámetros utilizados para el clasificador SVM.

```
RandomForestClassifier(bootstrap=False, max_depth=90, max_features=5,
↪ min_samples_leaf=3, min_samples_split=12,n_estimators=60)

```

Código 3: Hiperparámetros utilizados para el clasificador RF.

3. Resultados clasificación

Corresponde a los items 8 y 9 del enunciado. Para la clasificación se utilizaron las imágenes de la base de datos subida a U-Cursos (370 imágenes), la cual incluye 185 imágenes con personas y 185 imágenes sin personas (sillas y autos). Ejemplos de estas imágenes se pueden observar en la figura 3 en donde las imágenes (a) y (b) corresponden a imágenes sin personas y la imagen (c) corresponde a una imagen con una persona.

(a) Car_side (b) Chair (c) Pedestrian

Figura 3: Ejemplos de las imágenes de la base de datos.

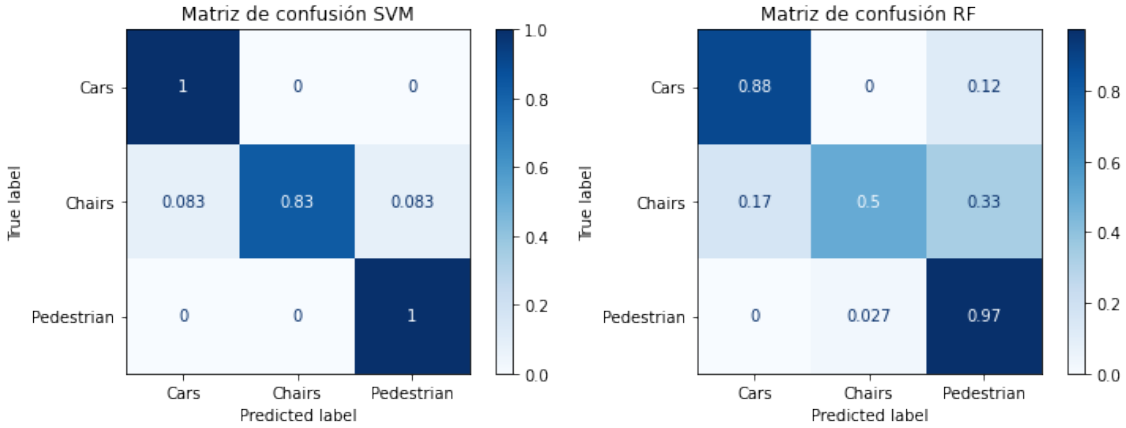
A continuación se muestran los resultados de las clasificaciones utilizando la metodología e hiperparametros comentados en la sección anterior.

3.1. Sin repartir votos entre cells

Este método es bastante simple y corto en cuanto a código, a pesar de esto los resultados son bastante buenos, 97.30 % y 86.49 % de accuracy respectivamente como se puede observar en la tabla 2, llegando a tener un 100 % de clasificaciones correctas para las personas con el clasificador SVM, por otro lado si bien el clasificador RF tiene peores resultados, como se observa la tasa de recall disminuye considerablemente, la detección de personas es del 97 %, a pesar de esto ambos clasificadores confunden sillas con personas y/o autos, esto nos lleva a buscar alternativas. Esto se puede observar en las matrices de confusión de la figura 4.

Tabla 2: Métricas para cada clasificador con los mejores parámetros determinados con el gridsearch.

	Accuracy test	Recall test
SVM	97.30 %	94.44 %
RF	86.49 %	78.43 %



(a) SVM (b) RF

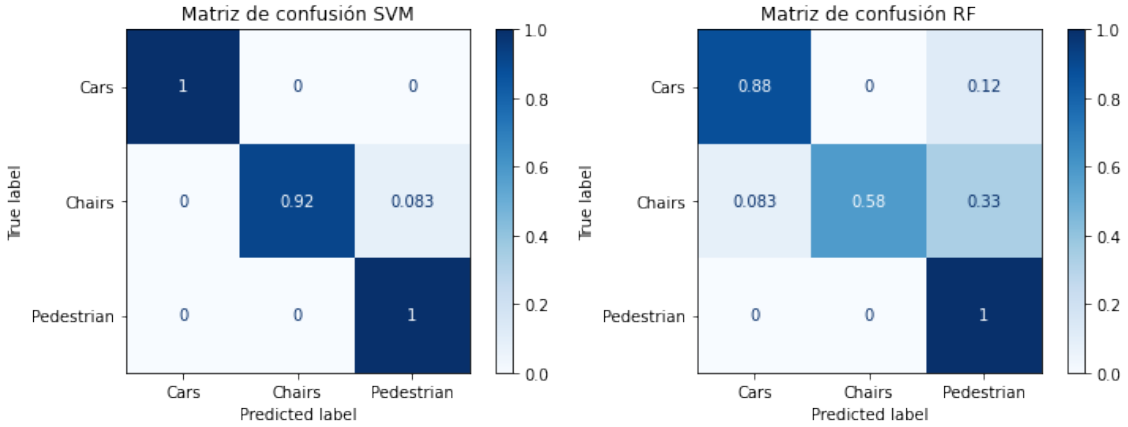
Figura 4: Matrices de confusión normalizadas. Utilizando SVM y Random Forest.

3.2. Repartiendo votos entre cells

Este método es más extenso y especifico, a pesar de esto tiene un tiempo de ejecución similar al método anterior, los resultados al repartir los votos entre cells mejorar considerablemente (en especial utilizando el método SVM), se puede observar que en las matrices de confusión de la figura 5, la detección de personas en ambos clasificadores es del 100 %, además aumenta el *accuracy* y el *recall* en comparación con el método anterior. Es decir, no solo detecta muy bien personas, sino que también se equivoca menos en la detección de otros objetos que no corresponden a personas.

Tabla 3: Métricas para cada clasificador con los mejores parámetros determinados con el gridsearch.

	Accuracy test	Recall test
SVM	98.65 %	97.22 %
RF	91.89 %	86.22 %



(a) SVM (b) RF

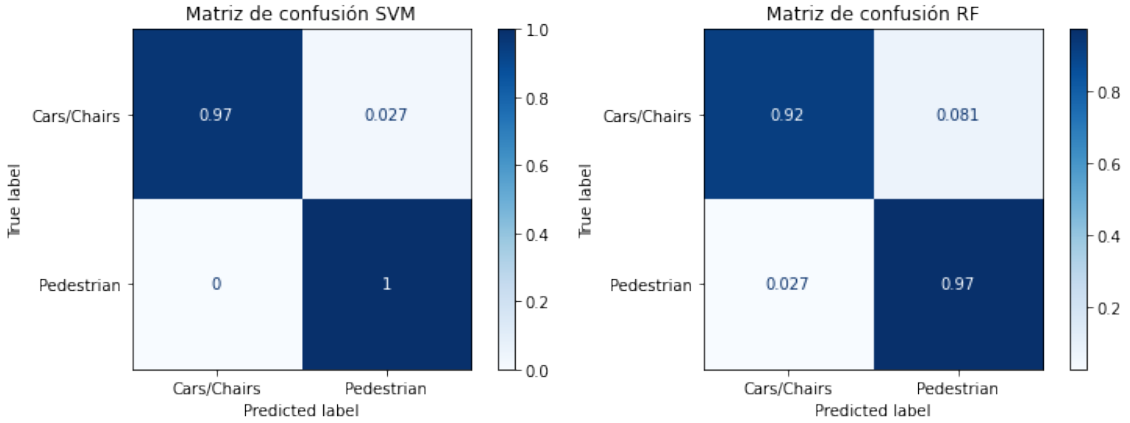
Figura 5: Matrices de confusión normalizadas. Utilizando SVM y Random Forest.

3.3. Clasificación binaria repartiendo votos entre cells

Finalmente se realiza una clasificación binaria con los mejores resultados de la clasificación multiclase anterior,esto pues el objetivo de la tarea es la detección de personas o no, por lo que lo más adecuado corresponde a realizar la clasificación binaria. Esta tiene resultados similares a la sub-sección anterior, pero mejora levemente, en especial en el recall, los clasificadores tienen menos errores y existen menos falsos positivos de personas. Por otro lado, los errores son menores al 2 % siendo muy efectivo el clasificador.

Tabla 4: Métricas para cada clasificador con los mejores parámetros determinados con el gridsearch.

	Accuracy test	Recall test
SVM	98.65 %	98.65 %
RF	95.95 %	95.95 %



(a) SVM (b) RF

Figura 6: Matrices de confusión normalizadas. Utilizando SVM y Random Forest. Resultados binarios.

Los mejores resultados corresponden a la clasificación utilizando el algoritmo SVM y la repartición de votos entre celdas. Por otro lado cabe destacar que el hecho de que el algoritmo funcione tan bien puede deberse a la buena base de datos con la cual se esta trabajando, en donde las personas están recortadas de forma perfecta lo cual en condiciones reales no es tan común.

4. Conclusión

Luego del desarrollo de esta tarea, se logró implementar y desarrollar un extractor de características HOG de forma propia y clasificadores de personas utilizando los algoritmos de SVM y Random Forest los cuales presentaron excelentes resultados. Además fue posible comprender el algoritmo HOG, sus ventajas y se pudo evidenciar su gran utilidad en el reconocimiento de objetos. Estas herramientas son bastante útiles para nuestra formación como ingenieros pues se puede aplicar a un sin fin de proyectos de la vida real (no solo teóricos). Se recomienda el uso de el clasificador SVM en caso de querer realizar un proyecto similar, el cual fue el que obtuvo mejores resultados de clasificación, a pesar de esto no descarto el uso de RF para otros proyectos pues es probable que en peores condiciones este tenga mejores resultados que SVM. Además se recomienda el uso de la interpolación con repartición de votos entre celdas pues permite obtener mejores resultados en un tiempo similar.
Por otro lado algo que fue considerable en esta tarea es el uso de gridsearch para la búsqueda de los mejores hiperparámetros, los cuales son de vital importancia en los resultados de las clasificaciones. Se pusieron en practica los conceptos y técnicas vistas en clases, programarlas en Python manteniendo el uso de librerías al mínimo e incluso programar con Cython lo cual permitió mejorar los tiempos de ejecución del programa. A pesar de esto algunas de las funciones realizadas no fueron optimizadas, por lo que el tiempo de ejecución del código puede mejorar. La mayor dificultad de esta tarea fue el de entender y lograr traspasar los algoritmos de HOG para desarrollar las funciones.

Referencias

[1] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection,"2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 2005, pp. 886-893 vol. 1, doi: 10.1109/CVPR.2005.177.

5. Anexos

Código 4: Código utilizado.

```

# -*- coding: utf-8 -*-
"""Tarea3_imagenes.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1SGB_DErHvyt9tzlXlOkHAYZHcsNMkLaI
"""

!ls

!unzip /content/imagenes_tarea3_2022.zip

!pwd

#    Para utilizar la interpolacion normal: Interpolacion = 0
#    Para utilizar la interpolacion bilinear: Interpolacion = 1

interpolacion = 1
gridsearch = True
binario = True

import cv2
import numpy as np

def transform(img):
    #transforma la imagen a escala de grises, al tipo np.float32
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #redimensiona la imagen a tamaño 64x128
    gray32 = np.float32(gray)
    out = cv2.resize(gray32, (64,128), interpolation = cv2.INTER_AREA)
    return out

example = cv2.imread('pedestrian/1.png')
out = transform(example)
print("shape",out.shape)
out

"""2. Implementar en python una función que reciba una imagen y calcule sus gradientes (se
puede reutilizar código de cálculo de gradientes de la tarea 2).

"""

# Commented out IPython magic to ensure Python compatibility.
# %load_ext Cython
```

```
47
48 # Commented out IPython magic to ensure Python compatibility.
49 # %%cython
50 # import cython
51 # import numpy as np
52 # cimport numpy as np
53 #
54 # cpdef np.ndarray[np.float32_t, ndim=2] gradx(np.ndarray[np.float32_t, ndim=2] input):
55 #   # POR HACER: calcular el gradiente en x
56 #   cdef int rows,cols, i,j
57 #
58 #   cdef np.ndarray[np.float32_t, ndim=2] output=np.zeros([input.shape[0], input.shape[1]], dtype =
       ↪  np.float32)
59 #
60 #   # tamano de la imagen
61 #   rows = input.shape[0]
62 #   cols = input.shape[1]
63 #
64 #   for i in range(rows):
65 #     for j in range(2,cols-2):
66 #         output[i][j-2] = -1*input[i][j-2]+0*input[i][j-1]+1*input[i][j]
67 #
68 #   return output
69
70 # Commented out IPython magic to ensure Python compatibility.
71 # %%cython
72 # import cython
73 # import numpy as np
74 # cimport numpy as np
75 #
76 # cpdef np.ndarray[np.float32_t, ndim=2] grady(np.ndarray[np.float32_t, ndim=2] input):
77 #   # POR HACER: Calcular el gradiente en y
78 #   cdef int rows,cols, i,j
79 #   cdef np.ndarray[np.float32_t, ndim=2] output=np.zeros([input.shape[0], input.shape[1]], dtype =
       ↪  np.float32)
80 #
81 #   # tamano de la imagen
82 #   rows = input.shape[0]
83 #   cols = input.shape[1]
84 #
85 #   for j in range(cols):
86 #     for i in range(2,rows-2):
87 #         output[i-2][j] = -1*input[i-2][j]+0*input[i-1][j]+1*input[i][j]
88 #
89 #   return output
90
91 # Commented out IPython magic to ensure Python compatibility.
92 # %%cython
93 # import cython
94 # import numpy as np
95 # cimport numpy as np
```

```
#
# cpdef MagDir(np.ndarray[np.float32_t, ndim=2] dx,np.ndarray[np.float32_t, ndim=2] dy):
#
#     cdef int rows,cols, i,j
#     cdef np.ndarray[np.float32_t, ndim=2] magnitud = np.zeros([dx.shape[0], dx.shape[1]], dtype =
    ↪  np.float32)
#     cdef np.ndarray[np.float32_t, ndim=2] angulo = np.zeros([dx.shape[0], dx.shape[1]], dtype =
    ↪ np.float32)
#
#     rows = dx.shape[0]
#     cols = dx.shape[1]
#     for i in range(rows):
#         for j in range(cols):
#             magnitud[i][j] = np.sqrt(dx[i][j]**2+dy[i][j]**2)
#             # Angulo en grados para simplificar los futuros histogramas
#             angulo[i][j] = np.degrees(np.arctan2(dx[i][j],dy[i][j]))
#
#             #los angulos deben quedar en el rango de 0° a 180°
#             if angulo[i][j]>=180:
#                 angulo[i][j] = angulo[i][j] - 180
#     return magnitud,angulo

"""4. Implementar en python el block normalization para el histograma. Para implementar esto,
se deben formar bloques de 2x2 celdas (con traslape). Cada bloque se debe transformar en
un vector de 1x36, el cual se debe normalizar. El vector de características final (de tamaño
1x3780) se obtiene concatenando los vectores normalizados de cada bloque.
"""

def block_normalization(histogram):
    """
    Implementar en python el block normalization para el histograma. Para implementar esto,
    se deben formar bloques de 2x2 celdas (con traslape). Cada bloque se debe transformar en
    un vector de 1x36, el cual se debe normalizar. El vector de características final (de tamaño
    1x3780) se obtiene concatenando los vectores normalizados de cada bloque
    """
    output = np.array([], np.float32)
    for i in range(16-1):
        for j in range(8-1):

            L = np.concatenate((histogram[i][j],histogram[i][j+1],histogram[i+1][j],histogram[i+1][j+1]))
            if sum(L)!=0:
                output= np.concatenate((output,L/sum(L)))
            else:
                output= np.concatenate((output,L))

    return np.array(output,np.float32)



def HOG(dx,dy,interpolacion=0):
    """3. Implementar en python una función que, a partir de los gradientes, calcule las
```

```
    características HOG usando 8x16 celdas (la salida debe ser un arreglo de numpy de
    dimensión 8x16x9).

    Para esto:
    1. se determinan 2 matrices, una de la magnitud de las derivadas y otra
    de las direcciones de estas.
    2. Se dividen las matrices en 8x16 celdas.
    3. En cada celda se calcula un histograma de orientación del gradiente
    con 9 componentes, es decir, se realiza un histograma el cual separa los angulos
    en 9 intervalos cubriendo cada uno 20°, luego por cada pixel de la celda se agregan
    los valores de la magnitud proporcionales a la distancia del angulo a los intervalos
    más cercanos, de esta manera los votos que se agregan son proporcionales a las distancias.
    4. Luego se reunen los histogramas de orientación del gradiente de cada celda
    los cuales corresponden a los features que retorna este método. Esto se reune
    en un arrelgo de numpy de dimensión 8x16x9.

    Para utilizar la interpolacion normal: Interpolacion = 0
    Para utilizar la interpolacion bilinear: Interpolacion = 1
    """
    histograms = np.zeros([16,8,9],np.float32)

    #utilizamos una función auxiliar para determinar la magnitud y el angulo
    magnitud, angulo = MagDir(dx,dy)
    rows = dx.shape[0]
    cols = dx.shape[1]

    for i in range(rows):
        for j in range(cols):
            cont1 = i//8
            cont2 = j//8
            #creamos un arreglo que guarde los valores del respectivo histograma
            bins = np.array([0,20,40,60,80,100,120,140,160])
            bin = int(np.floor(angulo[i][j]/20))
            if bin>8:
                bin=8
            p = angulo[i][j]/20-np.floor(angulo[i][j]/20)
            if interpolacion == 0: #lineal

                if angulo[i][j] %20==0:
                    histograms[cont1][cont2][bin] += magnitud[i][j]
                elif bin==8:
                    histograms[cont1][cont2][bin] += (1-p)*magnitud[i][j]
                else:
                    histograms[cont1][cont2][bin] += (1-p)*magnitud[i][j]
                    histograms[cont1][cont2][bin+1] += (p)*magnitud[i][j]

            else: #bilinearly
                """
                Determinamos las posiciones de los centros de las 4 celdas más cercanas que conforman
    ↪ un bloque,
```

```
194         luego a partir de la distancia del pixel con esas 4 celdas se distribuyen 8 votos, los cuales
    ↪ corresponden
195         a 2 votos por celda: 1 voto al bin correspondiente y otro voto al siguiente bin.
196         """
197
198         #listas que guardan las 4 celdas más cercanas y sus respectivos posiciones de sus centros
199         celdas = [ (cont1,cont2),(cont1,cont2+1),(cont1+1,cont2),(cont1+1,cont2+1) ]
200         centros = [((cont1*8+8)/2,(cont2*8+8)/2) ,((cont1*8+8)/2,((cont2+2)*8+8)/2), (((
    ↪ cont1+2)*8+8)/2,(cont2*8+8)/2),(((cont1+2)*8+8)/2,((cont2+2)*8+8)/2)]
201
202         #Zona normal
203         if celdas[0][0]!=15 and celdas[0][1]!=7:
204
205             # si estamos en el centro, todos los votos van para esa celda.
206             if i==centros[0][0] and j==centros[0][1]:
207                 if angulo[i][j] %20==0:
208                     histograms[cont1][cont2][bin] += magnitud[i][j]
209                 elif bin==8:
210                     histograms[cont1][cont2][bin] += (1-p)*magnitud[i][j]
211                 else:
212                     histograms[cont1][cont2][bin] += (1-p)*magnitud[i][j]
213                     histograms[cont1][cont2][bin+1] += (p)*magnitud[i][j]
214
215
216             distancias = []
217             for k in range(4):
218                 #determinamos la distancia al centro de la celda px,py
219                 px=centros[k][0]
220                 py=centros[k][1]
221
222                 d = np.sqrt((i-px)**2+(j-py)**2)
223                 distancias.append(d)
224
225             pd = np.array(distancias,np.float32)/sum(distancias)
226
227             for k in range(4):
228                 c1 = celdas[k][0]
229                 c2 = celdas[k][1]
230                 #print(pd[3-k],celdas[k])
231                 if angulo[i][j] %20==0:
232                     histograms[c1][c2][bin] += magnitud[i][j]*pd[3-k]*0.25
233                 elif bin==8:
234                     histograms[c1][c2][bin] += (1-p)*magnitud[i][j]*pd[3-k]*0.25
235                 else:
236                     histograms[c1][c2][bin] += (1-p)*magnitud[i][j]*pd[3-k]*0.25
237                     histograms[c1][c2][bin+1] += (p)*magnitud[i][j]*pd[3-k]*0.25
238
239         #Zonas bordes
240         else:
241             if angulo[i][j] %20==0:
242                 histograms[cont1][cont2][bin] += magnitud[i][j]
```

```
                elif bin==8:
                    histograms[cont1][cont2][bin] += (1-p)*magnitud[i][j]
                else:
                    histograms[cont1][cont2][bin] += (1-p)*magnitud[i][j]
                    histograms[cont1][cont2][bin+1] += (p)*magnitud[i][j]



    features = block_normalization(histograms)

    return features

#Pruebas
i=125
j=62
cont1 =i//8
cont2 = j//8
print("celda:",cont1,cont2)
celdas = [ (cont1,cont2),(cont1,cont2+1),(cont1+1,cont2),(cont1+1,cont2+1) ]
centros = [((cont1*8+8)/2,(cont2*8+8)/2) ,((cont1*8+8)/2,((cont2+2)*8+8)/2), (((cont1+2)*8+8)
    ↪ /2,(cont2*8+8)/2),(((cont1+2)*8+8)/2,((cont2+2)*8+8)/2)]

distancias = []
for k in range(4):
    #determinamos la distancia al centro de la celda px,py
    px=centros[k][0]
    py=centros[k][1]


    d = np.sqrt((i-px)**2+(j-py)**2)
    distancias.append(d)

p = np.array(distancias,np.float32)/sum(distancias)


for k in range(4):
    print(p[3-k],celdas[k][0],celdas[k][1])

dx = gradx(out)
print(dx.shape)
dy = grady(out)
result = HOG(dx,dy)
result

result.shape

from google.colab.patches import cv2_imshow
cv2_imshow(example)
cv2_imshow(out)
dx = gradx(out)
dy = grady(out)
```

```
magnitud, angulo = MagDir(dx,dy)
cv2_imshow(magnitud)
cv2_imshow(angulo)

example.shape

out.shape

"""5. Extraer características HOG de cada imagen del conjunto de entrenamiento. Se recomienda
aplicar un StandardScaler (usando scikit-learn) a las características para normalizarlas. El
StandardScaler se debe entrenar usando sólo las características de entrenamiento.

"""

import os

def cargarDatos(nombre_carpeta,clase):
    data = []
    contenido = os.listdir(nombre_carpeta)
    label = []
    for nombre_imagen in contenido:
        img = cv2.imread(os.path.join(nombre_carpeta, nombre_imagen))
        imgt = transform(img)
        data.append(imgt)
        label.append(clase)

    return np.array(data),np.array(label)

def featureExtractor(data,interpolacion=0):
    """
    Para utilizar la interpolacion normal: Interpolacion = 0
    Para utilizar la interpolacion bilinear: Interpolacion = 1
    """
    features = []
    for img in data:
        dx = gradx(img)
        dy = grady(img)
        feature = HOG(dx,dy,interpolacion)
        features.append(feature)
    #caracteristicas no normalizadas
    return np.array(features)

#Carga todos los datos de la carpeta "car_side" y le asigna la clase 0
dataCar_side,CSlabel = cargarDatos("car_side",0)
if binario==True:
    label_2 = 0
    text_labels = ["Cars/Chairs","pedestrian"]
else:
    label_2 = 1
    text_labels = ["Cars", "Chairs", "Pedestrian"]

```

```
#Carga todos los datos de la carpeta "chair" y le asigna la clase 1
dataChair,CL = cargarDatos("chair",label_2)

#Carga todos los datos de la carpeta "pedestrian" y le asigna la clase 2
dataPedestrian,PL = cargarDatos("pedestrian",2)

print("car_side.   ","shape = ",dataCar_side.shape, ", clase = ","0")
print("chair.      ","shape = ",dataChair.shape,  " , clase = ","1")
print("pedestrian. ","shape = ",dataPedestrian.shape, ", clase = ","2\n")

#data no normalizada
data = np.concatenate((dataCar_side,dataChair,dataPedestrian))
labels = np.concatenate((CSlabel,CL,PL))
data.shape

#Caracteristicas no normalizadas
features = featureExtractor(data,interpolacion=interpolacion)

from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc, accuracy_score, confusion_matrix
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import  PredefinedSplit

# Conjuntos de Train y Val/Test
X_trainval, X_test, y_trainval, y_test = train_test_split(features, labels, test_size=0.2, shuffle =
    ↪ True, stratify = labels)

X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, shuffle = False,test_size
    ↪ =0.25)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)

X_trainval = scaler.transform(X_trainval)
X_test = scaler.transform(X_test)

split_fold = [-1 for _ in range(int(len(X_trainval)*0.75))]+ [0 for _ in range(int(len(X_trainval)
    ↪ *0.25))]
cv = PredefinedSplit(split_fold)

X_train.shape

X_val.shape

X_test.shape

"""6. Entrenar un SVM (usando scikit-learn) con las características extraídas a cada imagen
redimensionada del conjunto de entrenamiento. Se debe elegir un kernel y usar grid
search, usando el conjunto de validación para encontrar los mejores hiper parámetros (se
```

```
debe usar PredefinedSplit).

# SVM
"""

from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
# Create the parameter grid based on the results of random search

param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': ['scale','auto',1,0.1],
              'kernel': ['linear','rbf', 'poly', 'sigmoid'],
              'class_weight':('balanced',None),
              'decision_function_shape':['ovo','ovr']}
# Instantiate the grid search model
grid = GridSearchCV(SVC(), param_grid,cv=cv, refit=False)

if gridsearch == False:
    classifier = SVC(C=0.1, class_weight='balanced', decision_function_shape='ovo',
    gamma= 'scale',kernel='linear')
else:
    # fitting the model for grid search
    grid.fit(X_trainval, y_trainval)
    print(grid.best_params_)
    classifier = SVC(C=0.1, class_weight='balanced', decision_function_shape='ovo',
    gamma= 'scale',kernel='linear')

#classifier = SVC(C=0.1, class_weight='balanced', decision_function_shape='ovo',
#    gamma= 'scale',kernel='linear')
# Interpolacion normal
#classifier = SVC(C=1, class_weight='balanced', decision_function_shape='ovo',
#    gamma= 'scale',kernel='rbf')

classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(y_pred)

# calculate accuracy
import matplotlib.pyplot as plt
from sklearn.metrics import  accuracy_score, confusion_matrix, recall_score
from sklearn.metrics import ConfusionMatrixDisplay


accuracy = accuracy_score(y_test, y_pred)*100
recall = recall_score(y_test, y_pred, average='macro')*100
print("Classification accuracy is  %2f" %accuracy," %")
print("Classification recall is  %2f" %recall," %\n")

fig = plt.figure(figsize = (12,9))
```

```
443
444 ConfusionMatrixDisplay.from_estimator(classifier, X_test, y_test,display_labels=text_labels,
        ↪ normalize="true",cmap=plt.cm.Blues)
445 t = plt.title("Matriz de confusión SVM")
446
447 """# Random Forest"""
448
449 from sklearn.model_selection import GridSearchCV
450 # Create the parameter grid based on the results of random search
451 param_grid = {
452     'bootstrap': [False,True],
453     'max_depth': [30, 50, 90, 100],
454     'max_features': [5,10,15,20,25,30],
455     'min_samples_leaf': [3, 4, 5],
456     'min_samples_split': [8, 10, 12],
457     'n_estimators': [30,60,100,200]
458 }
459 rf = RandomForestClassifier()
460
461 # Instantiate the grid search model
462 grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
463                            cv = 3, verbose = 3)
464
465 if gridsearch == False:
466     classifierRF = RandomForestClassifier(bootstrap=False, max_depth=90, max_features=30,
467                     min_samples_leaf=3, min_samples_split=8)
468 else:
469     # Fit the grid search to the data
470     grid_search.fit(X_train, y_train)
471     grid_search.best_params_
472     #se revisan los mejores hiperparametros encontrados
473     classifierRF = grid_search.best_estimator_
474
475 #classifierRF = best_grid
476 #RandomForestClassifier(bootstrap=False, max_depth=100, max_features=25,
477 #                       min_samples_leaf=4, min_samples_split=10)
478 #RandomForestClassifier(bootstrap=False, max_depth=90, max_features=30,
479 #                       min_samples_leaf=3, min_samples_split=8)
480 classifierRF.fit(X_train, y_train)
481 y_pred = classifierRF.predict(X_test)
482 print(y_pred)
483
484 # calculate accuracy
485 import matplotlib.pyplot as plt
486 from sklearn.metrics import  accuracy_score, confusion_matrix, recall_score
487 from sklearn.metrics import plot_confusion_matrix
488
489 accuracy = accuracy_score(y_test, y_pred)*100
490 recall = recall_score(y_test, y_pred, average='macro')*100
491 print("Classification accuracy is  %2f" %accuracy," %")
492 print("Classification recall is  %2f" %recall," %")
```

```
493
494 plt.figure(figsize = (12,9))
495 plot_confusion_matrix(classifierRF, X_test, y_test,display_labels=text_labels,cmap=plt.cm.Blues,
        ↪ normalize="true")
496 t = plt.title("Matriz de confusión RF")
```