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

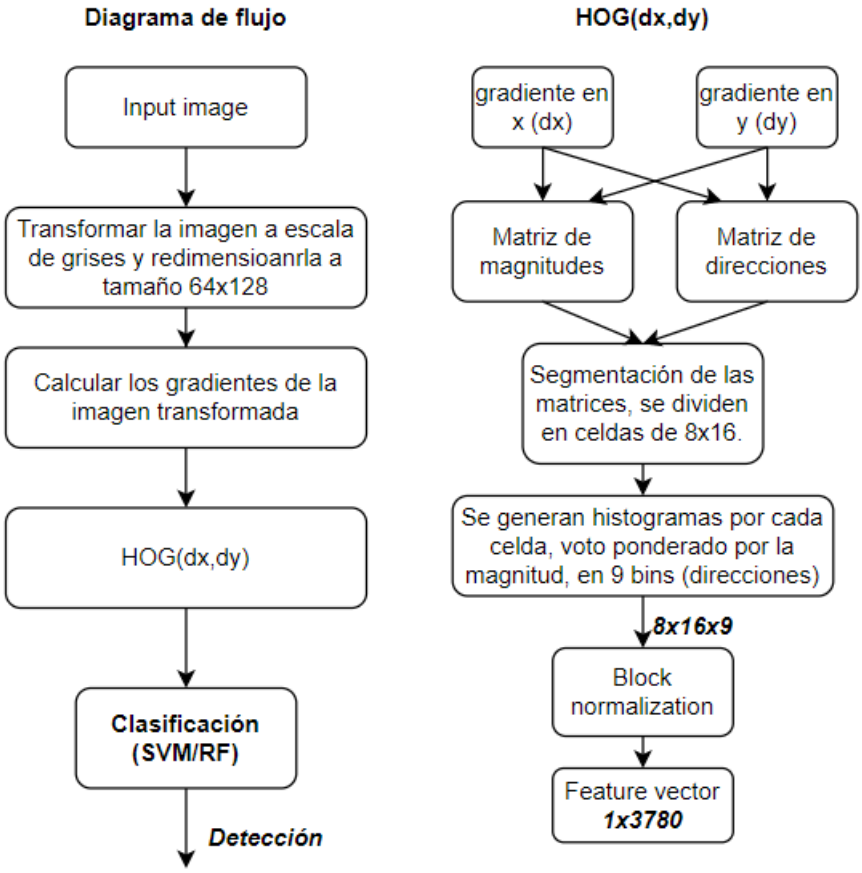

Figura 2: Diagrama de flujo resumido.

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.