Caracterización e Identificación de Vehículos Automotores Por Imagen Para la Estimación de Tráfico Urbano (noviembre de 2021)

Sierra G. Federico, Ríos L. Nicolas, y Betancourt A. Miguel

Resumen - En este documento se describe una solución al problema de caracterizar e identificar vehículos haciendo uso de algoritmos de extracción de características particularmente histograma de color y su posterior clasificación usando inteligencia artificial.

Índice de Términos – Inteligencia artificial, mejora de imágenes, procesamiento de imágenes, vehículos automotores.

I. Planteamiento del problema

Desde hace unos años, la movilidad se ha convertido en una de las mayores problemáticas en la ciudad de Bogotá, llegando a ocupar el tercer puesto a nivel mundial en el ranking de las ciudades con peor movilidad. Dentro de las posibles soluciones a este problema se encuentra la identificación y clasificación de vehículos en determinadas áreas de alta congestión, esto con el fin de tener un registro del tipo de carro (público, particular y moto) y calcular la cantidad de cada clase de vehículo que circula por una determinada vía y así realizar un posterior estudio de movilidad urbana que permita a entidades gubernamentales utilizarla como herramienta eficaz para la planeación de proyectos viales y de tránsito.

La extracción de características realizada sobre un conjunto de datos permite relacionar información y determinar la ocurrencia de un fenómeno. Muchas veces los datos entregados por dos fuentes de información no se encuentran relacionados como muchas otras veces si pero se pasan por alto. Teniendo en cuenta que el problema que se quiere solucionar realizando la propuesta descrita en este documento es el siguiente:

- Se quiere clasificar una vía según él volumen de tráfico que fluye por ella en tres diferentes clases, alto, medio y bajo. Es necesario tomar una medición que me permita conocer el número y clase de vehículos en un cierto lapso que cruzan por un tramo de la vía.

Como primer paso a la solución del problema se propone la creación o búsqueda de un conjunto de datos que nos permita posteriormente establecer una cantidad destinada para el entrenamiento, otra para la validación y pruebas de la red neuronal que a la salida del sistema entregue un valor estimado del tráfico urbano.
Definición del método de extracción de características que se aplicará para obtener el vector de variables altamente descriptivas de la información almacenada en el conjunto de datos. Para plantear un método de extracción de características es necesario analizar la data que se tomó o se encontró de alguna fuente.
Ordenar y normalizar la data extraída como vector y ajustar la red neuronal teniendo en cuenta el número de clases y límites entre las mismas. Por último se entrega una estimación del tráfico urbano teniendo en cuenta que debe estar definida por las etiquetas de salida ‘Tráfico alto’, ‘Tráfico medio’ y ‘Tráfico bajo’.

II. Solución Propuesta

La solución está repartida en tres grandes bloques, el primero de ellos es la parte de mejora de imagen, el segundo se trata del histograma y la obtención de un vector de características a partir de él y finalmente un bloque menos relevante para el caso, pero que nos ayudará a ver los resultados que es el bloque del algoritmo de inteligencia artificial.

Para el primer bloque lo que se realizó fue el desarrollo y aprovechamiento de algunas funciones con el fin de realizar una interpolación bicúbica a las imágenes básicas, para así aumentar el tamaño de estas, esto se realizó debido a que algunas imágenes tenían un tamaño y resolución insuficientes para obtener buenos resultados al aplicar el histograma de color. La codificación de estas funciones es la siguiente:

```
#Funciones para realizar la interpolacion

# Kernel de interpolacion
def u(s, a):
    if (abs(s) >= 0) & (abs(s) <= 1):
        return (a+2)*(abs(s)**3)-(a+3)*(abs(s)**2)+1
    elif (abs(s) > 1) & (abs(s) <= 2):
        return a*(abs(s)**3)-(5*a)*(abs(s)**2)+(8*a)*abs(s)-4*a
    return 0


# Padding
def padding(img, H, W, C):
    zimg = np.zeros((H+4, W+4, C))
    zimg[2:H+2, 2:W+2, :C] = img

    # Rellenar la primera/ultima fila y columna
    zimg[2:H+2, 0:2, :C] = img[:, 0:1, :C]
    zimg[H+2:H+4, 2:W+2, :] = img[H-1:H, :, :]
    zimg[2:H+2, W+2:W+4, :] = img[:, W-1:W, :]
    zimg[0:2, 2:W+2, :C] = img[0:1, :, :C]

    # Rellenar los puntos perdidos
    zimg[0:2, 0:2, :C] = img[0, 0, :C]
    zimg[H+2:H+4, 0:2, :C] = img[H-1, 0, :C]
```

```
    zimg[H+2:H+4, W+2:W+4, :C] = img[H-1, W-1, :C]
    zimg[0:2, W+2:W+4, :C] = img[0, W-1, :C]
    return zimg


# Operacion Bicubica
def bicubic(img, ratio, a):

    # Tamanño de la imagen
    H, W, C = img.shape

    # Here H = Height, W = weight,
    # C = Numero de canales si la imagen esta a color
    img = padding(img, H, W, C)

    # Crer nueva imagen
    dH = math.floor(H*ratio)
    dW = math.floor(W*ratio)

    # Convertir en matriz
    dst = np.zeros((dH, dW, 3))

    h = 1/ratio

    print('Start bicubic interpolation')
    print('It will take a little while...')
    inc = 0

    for c in range(C):
        for j in range(dH):
            for i in range(dW):

                # Obtener las coordinadas de los valores cercanos
                x, y = i * h + 2, j * h + 2

                x1 = 1 + x - math.floor(x)
                x2 = x - math.floor(x)
                x3 = math.floor(x) + 1 - x
                x4 = math.floor(x) + 2 - x

                y1 = 1 + y - math.floor(y)
                y2 = y - math.floor(y)
                y3 = math.floor(y) + 1 - y
                y4 = math.floor(y) + 2 - y

                # Considerando todos los 16 valores cercanos
                mat_l = np.matrix([[u(x1, a), u(x2, a), u(x3, a),
u(x4, a)]])
                mat_m = np.matrix([[img[int(y-y1), int(x-x1), c],
                                    img[int(y-y2), int(x-x1), c],
                                    img[int(y+y3), int(x-x1), c],
                                    img[int(y+y4), int(x-x1), c]],
                                   [img[int(y-y1), int(x-x2), c],
                                    img[int(y-y2), int(x-x2), c],
                                    img[int(y+y3), int(x-x2), c],
                                    img[int(y+y4), int(x-x2), c]],
                                   [img[int(y-y1), int(x+x3), c],
                                    img[int(y-y2), int(x+x3), c],
                                    img[int(y+y3), int(x+x3), c],
                                    img[int(y+y4), int(x+x3), c]],
                                   [img[int(y-y1), int(x+x4), c],
                                    img[int(y-y2), int(x+x4), c],
                                    img[int(y+y3), int(x+x4), c],
                                    img[int(y+y4), int(x+x4), c]]])
                mat_r = np.matrix(
                    [[u(y1, a)], [u(y2, a)], [u(y3, a)], [u(y4, a)]])

                # Producto punto entre las dos matrices
                dst[j, i, c] = np.dot(np.dot(mat_l, mat_m), mat_r)

    sys.stderr.write('\n')

    # Flushing the buffer
    sys.stderr.flush()
       return dst
```

Adicionalmente, este bloque cuenta con un par de instrucciones más que debemos añadir al importar la imagen correspondiente, que son:

```
# Adentrando la imagen a la funcion bicubica
dst = bicubic(img, ratio, a)
print(fname)

#Guardando la imagen nueva
cv2.imwrite('bicubic.png', dst)
bicubicImg = cv2.imread('bicubic.png')
img = cv.imread('bicubic.png')
```

Nuestro segundo bloque es un poco más corto en cuanto a codificación sin embargo es el bloque más importante, dado que es el que fundamenta toda la solución y la hace realmente viable, este lo primero que hace es obtener el histograma de color de la imagen y separa este en una matriz de 7x8 lo que nos da 56 características por imagen, cada una de estas características es un valor de 0-256 en el espacio de color hsv, una vez calculado el histograma propiamente dicho simplemente se recorre este y se guardan las características en un vector para su posterior uso. Esta codificación queda:

```
  # Calcula histograma
  hsv = cv.cvtColor(img,cv.COLOR_BGR2HSV)
  hist = cv.calcHist( [hsv], [0, 1], None, [7, 8], [0, 180, 0, 256] )
  plt.imshow(hist,interpolation = 'nearest')
  plt.show()

  for i in range(len(hist[0])): #Se recorren las columnas
    for j in range(len(hist)): ##Se recorren las filas
      #print(hist[j][i])
        x.append(hist[j][i])   #Se   llena   el   vector   con   las
características
```

Finalmente llegamos al último gran bloque, en este caso se usará un algoritmo sencillo de árbol de decisión, el cual tomará los datos etiquetados y realizará un entrenamiento y posterior verificación de estos, con el fin de conocer la matriz de confusión y el accuracy esto con el fin de identificar si las características extraídas de la imagen son útiles o no finalmente para dar solución al problema. Por lo que la codificación nos queda:

```
clases_carros = pd.read_csv('caracteristicas.csv')
x = clases_carros.iloc[:,:-1].values
y = clases_carros.iloc[:,-1].values
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size =
0.3, random_state = 21)
###############################Decision
tree###################################
tree = DecisionTreeClassifier(criterion = "entropy", random_state =
21)
#Train
tree.fit(x_train,y_train)

y_pred = tree.predict(x_test)
print(y_pred)
treeConfMat = confusion_matrix(y_test,y_pred)
sns.heatmap(treeConfMat,annot=True)
plt.xlabel("Predicted label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show
print("accuracy")
print(accuracy_score(y_test,y_pred))
```

Sin embargo estos bloques por separado no dan solución a ningún problema, para lograr nuestro cometido es necesario unir estos de manera correcta, y aquí es donde entra la importancia de tener un dataset de imágenes correctamente separadas y para el caso identificables en cuanto a su nombre se refiere, esto para poder realizar el correcto etiquetado de las muestras, partiendo de esto la codificación del algoritmo completo queda:

```
import cv2
import math
import sys
import time
import glob
import numpy as np
import seaborn as sns
import cv2 as cv
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

#Funciones para realizar la interpolacion

# Kernel de interpolacion
def u(s, a):
    if (abs(s) >= 0) & (abs(s) <= 1):
```

```
        return (a+2)*(abs(s)**3)-(a+3)*(abs(s)**2)+1
    elif (abs(s) > 1) & (abs(s) <= 2):
        return a*(abs(s)**3)-(5*a)*(abs(s)**2)+(8*a)*abs(s)-4*a
    return 0


# Padding
def padding(img, H, W, C):
    zimg = np.zeros((H+4, W+4, C))
    zimg[2:H+2, 2:W+2, :C] = img

    # Rellenar la primera/ultima fila y columna
    zimg[2:H+2, 0:2, :C] = img[:, 0:1, :C]
    zimg[H+2:H+4, 2:W+2, :] = img[H-1:H, :, :]
    zimg[2:H+2, W+2:W+4, :] = img[:, W-1:W, :]
    zimg[0:2, 2:W+2, :C] = img[0:1, :, :C]

    # Rellenar los puntos perdidos
    zimg[0:2, 0:2, :C] = img[0, 0, :C]
    zimg[H+2:H+4, 0:2, :C] = img[H-1, 0, :C]
    zimg[H+2:H+4, W+2:W+4, :C] = img[H-1, W-1, :C]
    zimg[0:2, W+2:W+4, :C] = img[0, W-1, :C]
    return zimg


# Operacion Bicubica
def bicubic(img, ratio, a):

    # Tamanño de la imagen
    H, W, C = img.shape

    # Here H = Height, W = weight,
    # C = Numero de canales si la imagen esta a color
    img = padding(img, H, W, C)

    # Crer nueva imagen
    dH = math.floor(H*ratio)
    dW = math.floor(W*ratio)

    # Convertir en matriz
    dst = np.zeros((dH, dW, 3))

    h = 1/ratio

    print('Start bicubic interpolation')
    print('It will take a little while...')
    inc = 0

    for c in range(C):
        for j in range(dH):
            for i in range(dW):

                # Obtener las coordinadas de los valores cercanos
                x, y = i * h + 2, j * h + 2

                x1 = 1 + x - math.floor(x)
                x2 = x - math.floor(x)
                x3 = math.floor(x) + 1 - x
                x4 = math.floor(x) + 2 - x

                y1 = 1 + y - math.floor(y)
                y2 = y - math.floor(y)
                y3 = math.floor(y) + 1 - y
                y4 = math.floor(y) + 2 - y

                # Considerando todos los 16 valores cercanos
                mat_l = np.matrix([[u(x1, a), u(x2, a), u(x3, a),
u(x4, a)]])
                mat_m = np.matrix([[img[int(y-y1), int(x-x1), c],
                                    img[int(y-y2), int(x-x1), c],
                                    img[int(y+y3), int(x-x1), c],
                                    img[int(y+y4), int(x-x1), c]],
                                   [img[int(y-y1), int(x-x2), c],
                                    img[int(y-y2), int(x-x2), c],
                                    img[int(y+y3), int(x-x2), c],
                                    img[int(y+y4), int(x-x2), c]],
                                   [img[int(y-y1), int(x+x3), c],
                                    img[int(y-y2), int(x+x3), c],
                                    img[int(y+y3), int(x+x3), c],
                                    img[int(y+y4), int(x+x3), c]],
                                   [img[int(y-y1), int(x+x4), c],
                                    img[int(y-y2), int(x+x4), c],
                                    img[int(y+y3), int(x+x4), c],
                                    img[int(y+y4), int(x+x4), c]]])
                mat_r = np.matrix(
                    [[u(y1, a)], [u(y2, a)], [u(y3, a)], [u(y4, a)]])

                # Producto punto entre las dos matrices
                dst[j, i, c] = np.dot(np.dot(mat_l, mat_m), mat_r)

    sys.stderr.write('\n')

    # Flushing the buffer
    sys.stderr.flush()
    return dst

x=[] #Crear vector de caracteristicas
res=[]
z=[]
#Factor de escala
ratio = 2

# Coeficiente
a = -1/2

imagesMoto = glob.glob('Moto*.JPG')

for fname in imagesMoto:

  # Leer la imagen
  img = cv2.imread(fname)
  # Adentrando la imagen a la funcion bicubica
  dst = bicubic(img, ratio, a)
  print(fname)

  #Guardando la imagen nueva
  cv2.imwrite('bicubic.png', dst)
  bicubicImg = cv2.imread('bicubic.png')
  img = cv.imread('bicubic.png')
  # Calcula histograma
  hsv = cv.cvtColor(img,cv.COLOR_BGR2HSV)
  hist = cv.calcHist( [hsv], [0, 1], None, [7, 8], [0, 180, 0, 256] )
  plt.imshow(hist,interpolation = 'nearest')
  plt.show()

  for i in range(len(hist[0])): #Se recorren las columnas
    for j in range(len(hist)): ##Se recorren las filas
      #print(hist[j][i])
      x.append(hist[j][i]) #Se llena el vector con las
caracteristicas
  x.append('2')

imagesPart = glob.glob('Particular*.JPG')

for fname in imagesPart:

  # Leer la imagen
  img = cv2.imread(fname)
  # Adentrando la imagen a la funcion bicubica
  dst = bicubic(img, ratio, a)
  print(fname)

  #Guardando la imagen nueva
  cv2.imwrite('bicubic.png', dst)
  bicubicImg = cv2.imread('bicubic.png')
  img = cv.imread('bicubic.png')
  # Calcula histograma
  hsv = cv.cvtColor(img,cv.COLOR_BGR2HSV)
  hist = cv.calcHist( [hsv], [0, 1], None, [7, 8], [0, 180, 0, 256] )
  plt.imshow(hist,interpolation = 'nearest')
  plt.show()

  for i in range(len(hist[0])): #Se recorren las columnas
    for j in range(len(hist)): ##Se recorren las filas
      #print(hist[j][i])
      x.append(hist[j][i]) #Se llena el vector con las
caracteristicas
  x.append('1')

imagesPub = glob.glob('Publico*.JPG')

for fname in imagesPub:

  # Leer la imagen
  img = cv2.imread(fname)
  # Adentrando la imagen a la funcion bicubica
  dst = bicubic(img, ratio, a)
  print(fname)

  #Guardando la imagen nueva
  cv2.imwrite('bicubic.png', dst)
  bicubicImg = cv2.imread('bicubic.png')
  img = cv.imread('bicubic.png')
  # Calcula histograma
  hsv = cv.cvtColor(img,cv.COLOR_BGR2HSV)
  hist = cv.calcHist( [hsv], [0, 1], None, [7, 8], [0, 180, 0, 256] )
  plt.imshow(hist,interpolation = 'nearest')
  plt.show()

  for i in range(len(hist[0])): #Se recorren las columnas
    for j in range(len(hist)): ##Se recorren las filas
      #print(hist[j][i])
      x.append(hist[j][i]) #Se llena el vector con las
caracteristicas
  x.append('0')

res=
np.array(x).reshape(len(imagesPart)+len(imagesPub)+len(imagesMoto),(l
en(hist[0])*len(hist))+1)
carros=pd.DataFrame(res)
carros.to_csv('caracteristicas.csv')

clases_carros = pd.read_csv('caracteristicas.csv')
x = clases_carros.iloc[:,:-1].values
y = clases_carros.iloc[:,-1].values
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3,
random_state = 21)
################################Decision
tree###################################
tree = DecisionTreeClassifier(criterion = "entropy", random_state =
21)
#Train
tree.fit(x_train,y_train)
```

```
y_pred = tree.predict(x_test)
print(y_pred)
treeConfMat = confusion_matrix(y_test,y_pred)
sns.heatmap(treeConfMat,annot=True)
plt.xlabel("Predicted label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show
print("accuracy")
print(accuracy_score(y_test,y_pred))
```

III. Resultados

Una vez visto detalladamente cada uno de los bloques del algoritmo y su codificación. Y aplicando este algoritmo a un dataset de aproximadamente 70 imágenes (las cuales fueron previamente renombradas y clasificadas manualmente), se obtiene la siguiente matriz de confusión:

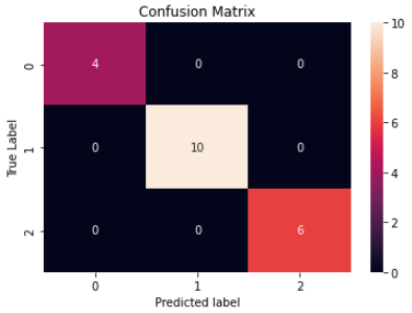

Imagen 1. Matriz de confusión.

Además, se obtuvo un vector de predicción: [2 1 2 1 0 2 2 1 1 1 2 2 1 0 1 0 1 1 1 0] el cual combinado con el accuracy obtenido el cual es de 1.0, podemos afirmar que el algoritmo clasificó correctamente las 20 imágenes elegidas aleatoriamente del vector de muestra donde encontró 6 motos, 10 carros públicos y 6 carros particulares.

Podemos también destacar que el algoritmo nos permite ver el histograma obtenido en el proceso de extracción de este, como se observa en las imágenes 2-4.

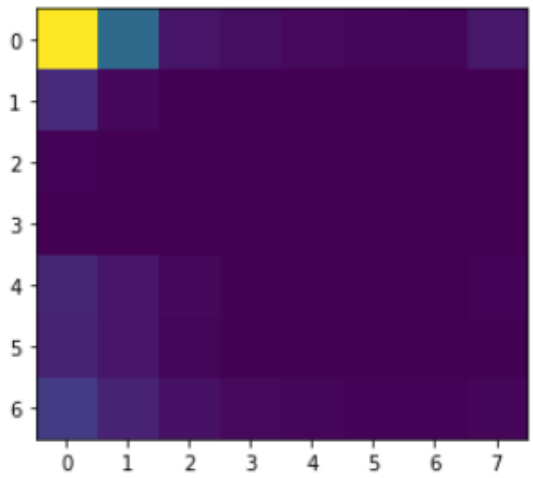

Imagen 2. Histograma 1 (Moto).

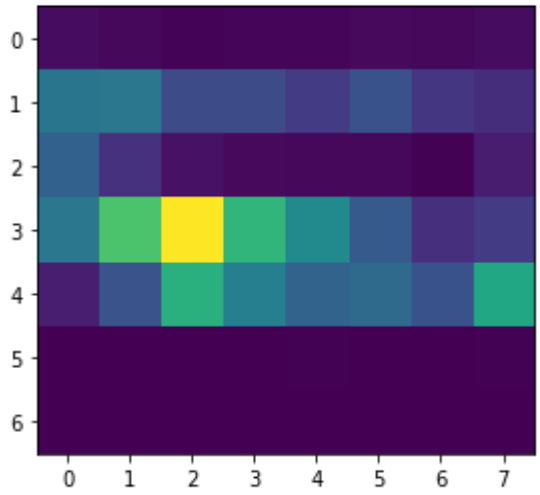

Imagen 3. Histograma 2 (Público).

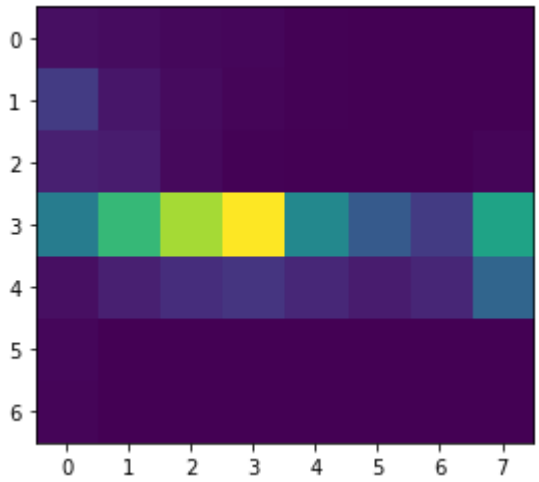

Imagen 4. Histograma 3 (Particular).

IV. Conclusiones

- Basados en los resultados obtenidos, podemos afirmar finalmente que la extracción del histograma de color fue una buena escogencia como método de extracción de características, ya que permitió la correcta separación de las clases propuestas en el planteamiento del problema.
- Los algoritmos de mejora de imágenes cobran mayor importancia cuando nos vemos enfrentados a problemas reales, esto debido a que muchas veces son necesarios para poder aplicar de manera correcta los algoritmos de extracción de características.
- Aunque es menos relevante para el problema, se hace necesario en muchos casos unir el procesamiento de imágenes con la inteligencia artificial, particularmente en este caso fue necesario para corroborar o desmentir la relevancia de las características extraídas.