Identificación de placas vehiculares en Colombia a partir de reconocimiento óptico de caracteres ROC

Juan M. Aya Perlaza
Departamento de Ingeniería Electrónica
Pontificia Universidad Javeriana

Laura Joaquí Muñoz
Departamento de Ingeniería Electrónica
Pontificia Universidad Javeriana

Victor Múnera Rojas
Departamento de Ingeniería Electrónica
Pontificia Universidad Javeriana

Inteligencia Artificial
11 de septiembre de 2020

1. Introducción

La integración masiva de las tecnologías de la información, bajo diferentes aspectos del mundo moderno, ha llevado a tratar los vehículos como recursos conceptuales en los sistemas de información. Dentro de los equipos inteligentes se menciona el sistema de detección y reconocimiento de matrículas de vehículos.

Este sistema se utiliza para detectar las placas y luego hacer el reconocimiento de la matrícula que consiste en extraer el texto de una imagen y todo ello gracias a los módulos de cálculo que utilizan algoritmos de localización, segmentación de la matrícula y reconocimiento de caracteres. La detección y lectura de matrículas es un tipo de sistema inteligente y es considerable por las posibles aplicaciones en varios sectores, por ejemplo:

- *Cuerpo policial*: Este sistema se utiliza para la detección de vehículos robados y registrados. Las matrículas detectadas se comparan con las de los vehículos denunciados.
- *Gestión de parqueaderos*: La gestión de las entradas y salidas de los vehículos.
- *Seguridad vial*: Este sistema se utiliza para detectar matrículas que superan una cierta velocidad.

2. Objetivos

2.1. Objetivo General

- Implementar un sistema que permita reconocer los caracteres presentes en placas vehiculares colombianas.

2.2. Objetivos Específicos curso INTELIGENCIA ARTIFICIAL

- Escoger las mejores características para el entrenamiento del modelo y posterior predicción de letras.
- Utilizar un algoritmo de reconocimiento óptico de caracteres para obtener las letras y números de la placa.
- Realizar una comparación entre los distintos modelos de predicción para ver con cuál se obtienen mejores resultados.

3. Marco teórico

3.1. Placas vehiculares colombianas

Los residentes en Colombia son requeridos por las autoridades correspondientes para registrar sus vehículos motorizados y exhibir la matrícula automovilística (o Placas, como se les conoce comúnmente). El actual diseño de placas fue introducido en los años 90, aunque algunas matrículas con el diseño anterior (usado entre 1972 y 1990) siguen en uso. [1]

Categoría	Imagen	Tipo	Colores	Formato de serie
Pasajeros		Particular	negro sobre amarillo	ABC·123[1] [2]
		Comercial	negro sobre blanco[1] [2]	

Diseño de placas colombianas. Imagen de Wikipedia.

3.2. Reconocimiento automático de matrículas

El Reconocimiento Automático de Matrículas (ANPR) es un sistema de gran precisión capaz de leer las matrículas de los vehículos sin intervención humana mediante el uso de captura de imágenes a alta velocidad con iluminación de apoyo, detección de caracteres dentro de las imágenes proporcionadas, verificación de las secuencias de caracteres como si fueran de una matrícula de vehículo, reconocimiento de caracteres para convertir la imagen en texto; terminando así con un conjunto de metadatos que identifican una imagen que contiene una matrícula de vehículo y el texto decodificado asociado de esa matrícula. [2]

3.3. Reconocimiento óptico de caracteres ROC

El reconocimiento óptico de caracteres (ROC), generalmente conocido como reconocimiento de caracteres y expresado con frecuencia con la sigla OCR (del inglés Optical Character Recognition), es un proceso dirigido a la digitalización de textos, los cuales identifican automáticamente a partir de una imagen símbolos o caracteres que pertenecen a un determinado alfabeto, para luego almacenarlos en forma de datos. [4]

4. Descripción de la solución

4.1. Propuesta

Se propone diseñar e implementar un sistema de software que permita, a partir de muestras fotográficas tomadas a vehículos en calles colombianas, identificar, segmentar y, para cada letra o número incluido en la placa de tránsito en cuestión, implementar un algoritmo de reconocimiento óptico de caracteres para poder conocer la matrícula del vehículo.

Imagen tomada de https://www.syscomblog.com/2018/10/software-para-reconocimiento-automatico.html

Se procederá a tomar la información de las bases de datos provistas por el docente, para luego realizar la identificación de los objetos en las imágenes y discriminar las placas. Luego de esto, se segmentará

la imagen con el fin de procesar las respectivas partes que la componen, con el fin de conocer la placa del vehículo, empleando un algoritmo de reconocimiento óptico de caracteres.

4.2. Diagrama en Bloques

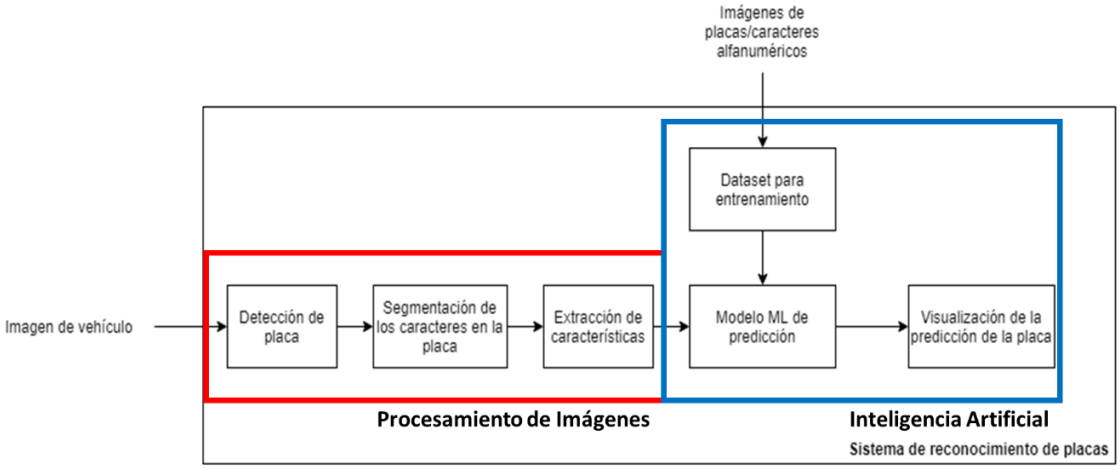


Diagrama en bloques del sistema

Como se puede observar en la imagen anterior, se diseñó un sistema que permita reconocer y predecir los caracteres de una placa vehicular colombiana a partir del procesamiento de una imagen inicial que contenga un vehículo y mediante el entrenamiento de algunos algoritmos de aprendizaje de máquina.
Para conseguir esto, se parte de la imagen inicial en la cual se identifica el lugar en el que se encuentra la placa a través del reconocimiento del contorno con un área y dimensiones relativas al tamaño de la imagen, luego, conociendo el lugar donde está la placa se toma este nuevo recorte de imagen y, mediante el mismo algoritmo, se detectan cada una de las letras/números presentes en ella. Esto permite generar 6 nuevos recortes de imagen, cada uno pasará por un modelo de Aprendizaje de Máquina previamente entrenado para poder predecir a qué caracter alfanumérico hace referencia.

4.3. Diseño de la solución

Para el diseño del sistema, se hizo uso de las librerías de OpenCV para toda la parte de Procesamiento de Imágenes, NumPY y Sklearn para el apartado de Inteligencia Artificial y, por último, glob para el manejo de directorios.

Los diagramas de flujo del sistema se presentan a continuación:

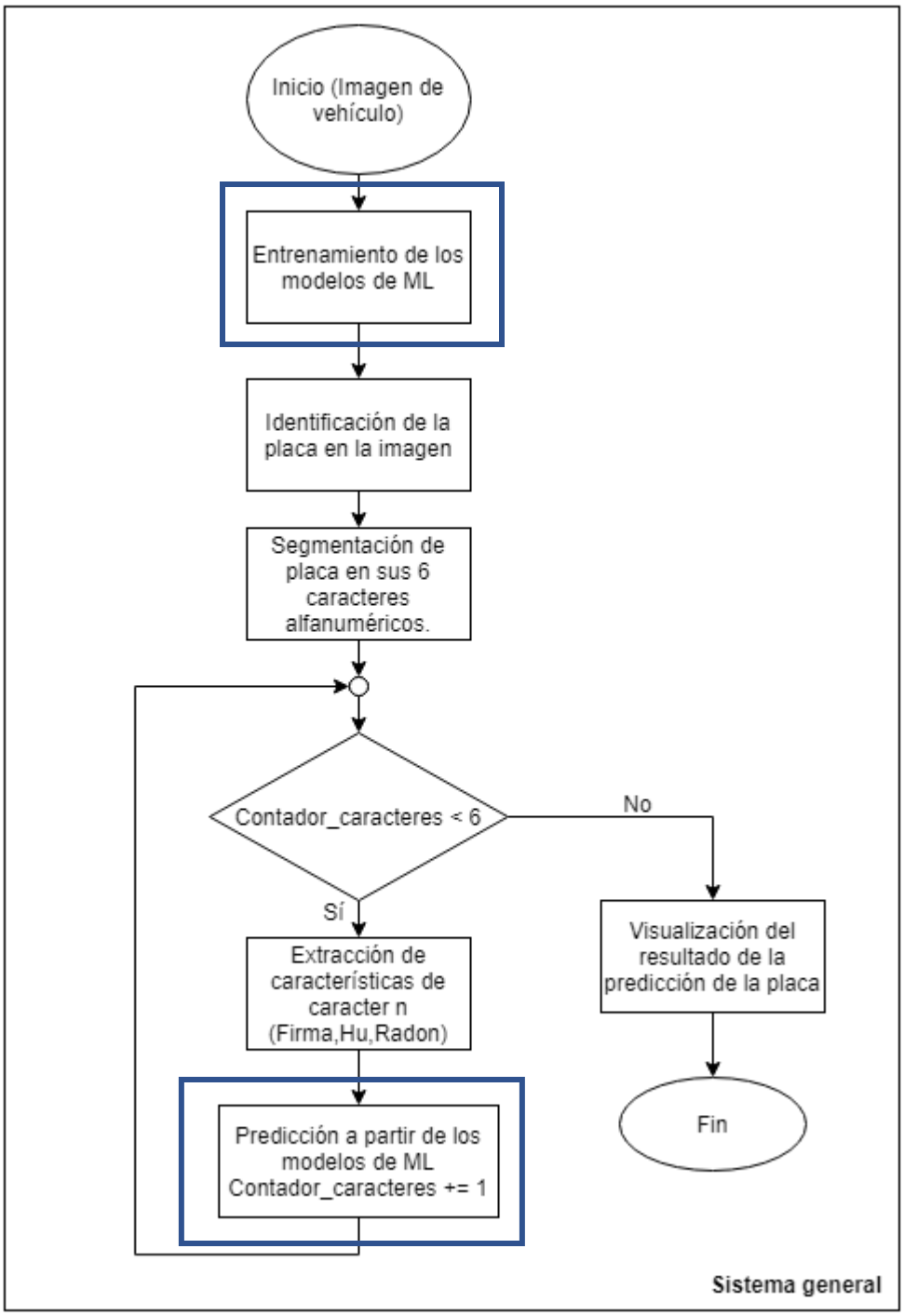


Diagrama de flujo del sistema general. En azul los bloques de IA

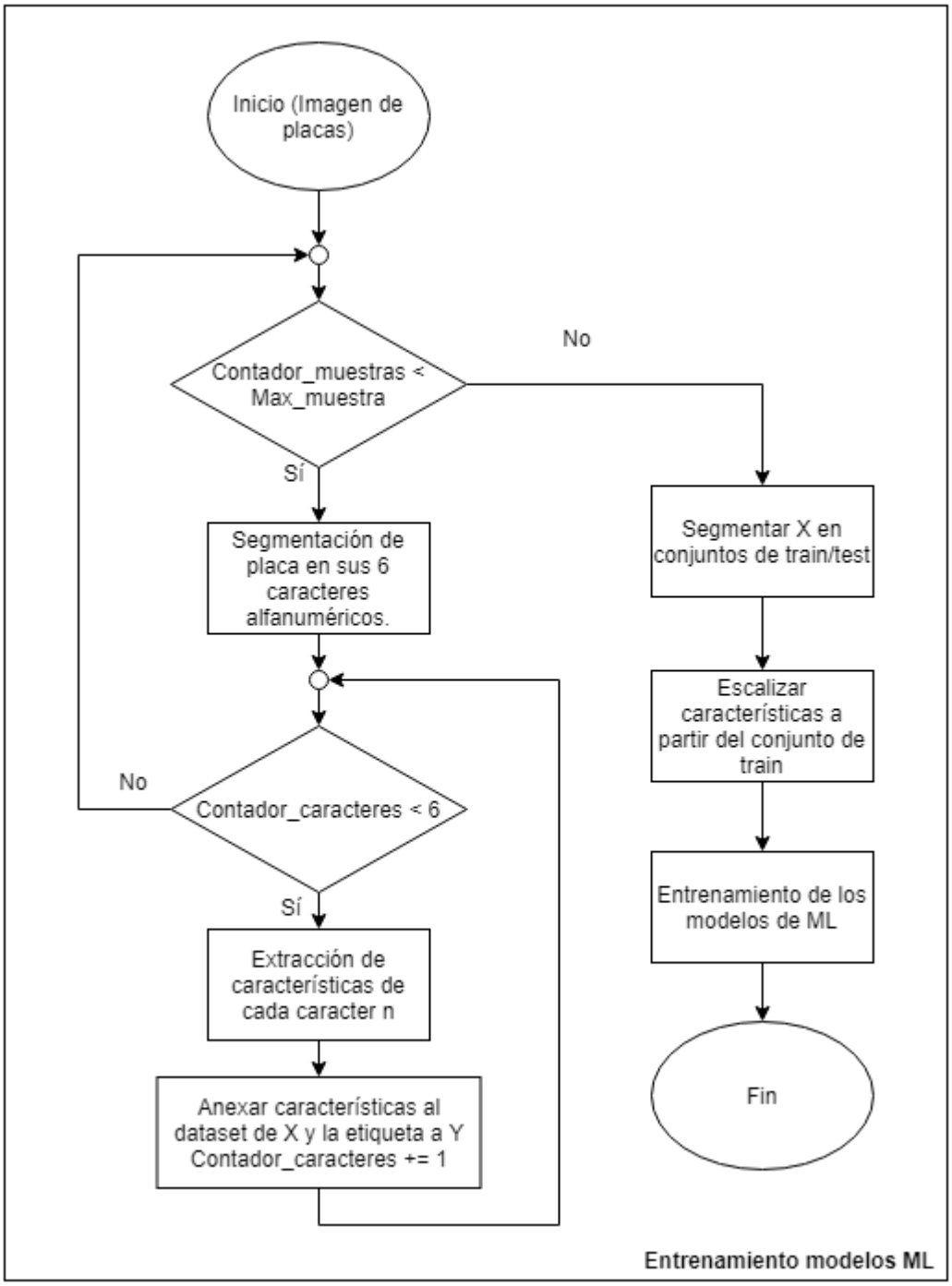
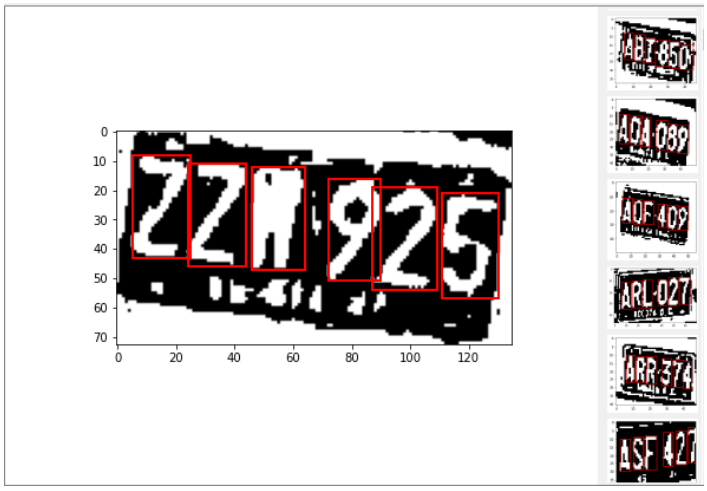


Diagrama de flujo del entrenamiento de los modelos de ML

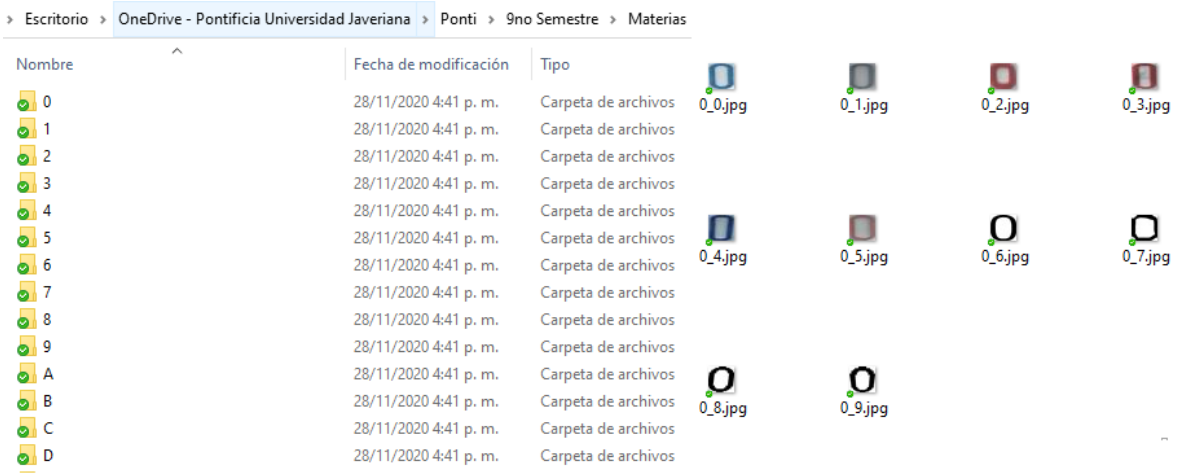
Entrenamiento de los modelos de ML

Para el entrenamiento, se utilizaron dos bases de datos:

Primero, se utilizaron 162 fotos de placas tomadas por cámaras en semáforos de la ciudad de Bogotá por la secretaría de movilidad. Esta base de datos resulta confidencial, por eso no se pueden subir las muestras, sin embargo, el método a utilizar fue segmentar cada imagen en los respectivos caracteres y, por último, a cada carácter sacarle características para conformar el dataset junto a sus etiquetas. Se obtuvieron un total de 972 letras.



Por otra parte, se utilizó un dataset encontrado en github de un proyecto similar, el cual contenía 10 imágenes de 20x20 pixeles para cada carácter alfanumérico.



Validación de modelos.
Se implementaron 3 diferentes algoritmos de aprendizaje de máquina, a saber: KNN, SVM y Red neuronal.
Para cada una, se variaron sus respectivos hiperparámetros para encontrar cuál era el mejor modelo para el problema en cuestión. Finalmente, se obtuvieron los siguientes resultados:

```
El mejor resultado fue: [1, 0.9647058823529412]
[[4 0 0 ... 0 0 0]
 [0 4 0 ... 0 0 0]
 [0 0 2 ... 0 0 0]
 ...
 [0 0 0 ... 3 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 4]]
```

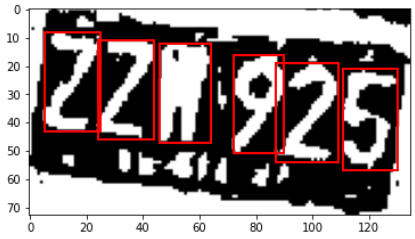
Modelo KNN. Mejor modelo con k=1 y accuracy del 96% en conjunto de validación.

```
El mejor Kernel es linear          con un accuracy de: 0.9882352941176471
Accuracy of SVM classifier on test set: 0.99
```

Modelo SVM. Mejor modelo con kernel lineal y accuracy de 99% en conjunto de validación.

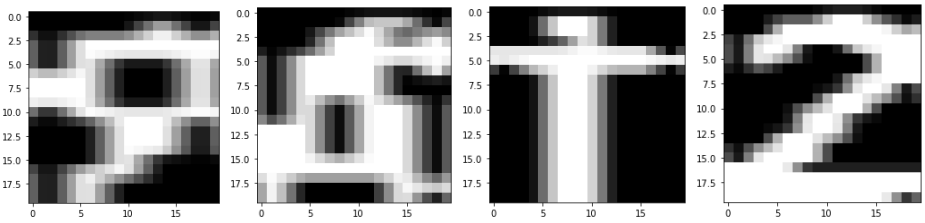
```
Accuracy por Red Neuronal con sklearn en conjunto de entrenamiento: 1.00
Accuracy por Red Neuronal con sklearn en conjunto de validación: 0.72
[[3 0 0 ... 0 0 0]
 [0 2 0 ... 0 0 0]
 [0 0 2 ... 0 0 0]
 ...
 [0 0 0 ... 1 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 4]]
```

Modelo NN. Modelo con 5 capas de (40,40,30,15,8) neuronas y accuracy de 72% en conjunto de validación.



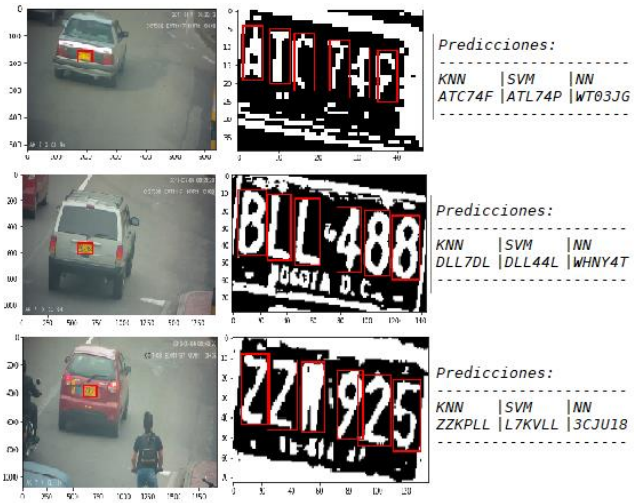
Predicción

Para la predicción, cada una de las letras detectadas se reescalizaban a un tamaño de 20x20, se les sacaba las características (Momentos de Hu, Transformada de Radón y Firma), se escalizaban las características a partir del conjunto de entrenamiento previo y se ingresaban a la función predict de cada modelo.



Resultados

Después de todo el proceso previamente mencionado, se obtuvieron las siguientes predicciones para las respectivas placas:



5. Conclusiones

- Las características tomadas a cada recorte de imagen (caracter) hacían referencia a 3 métodos distintos, los momentos de Hu, Transformada de Radón y la firma de la imagen. Primero, se realizó un entrenamiento con cada uno de los métodos independientemente y, por último, se juntaron todos los métodos en el dataset de características, arrojando los siguientes resultados:

Método	**Accuracy en conjunto de test**
Momentos de Hu	38%
Transformada de Radón	66%
Firma	66%
Concatenación de todos	66%

  A raíz de estos resultados, al ver que juntando todos los métodos no aumentaba la exactitud del modelo, entonces se decidió utilizar únicamente la firma que consiste en aplanar la matriz de imagen a una sola dimensión y así no aumentar el tiempo de procesamiento del algoritmo con los otros métodos que sí implican el llamado a otras funciones.
- Por otra parte, inicialmente se realizó la creación del dataset de entrenamiento a partir de la base de datos dada por el docente, con imágenes confidenciales de placas de la ciudad. Sin embargo, de todo el conjunto de imágenes, solo 162 funcionaron y estas no fueron suficientes para lograr un buen entrenamiento de los modelos, razón por la cual el valor máximo de exactitud al hacer la validación era de 66%. Para dar solución a este inconveniente, se procedió a buscar otro dataset en internet con imágenes de caracteres y se encontró una carpeta en GitHub con 10 muestras para cada letra y número. Al hacer el entrenamiento y validación con este nuevo dataset se obtuvieron exactitudes de 98 y 99%.
- Consecuentemente, la predicción de las letras cuando se hizo la prueba del sistema con diferentes placas fue mucho más acertada, logrando predecir entre 3 y 4 caracteres de los presentes en cada placa.

Referencias

[1] Ministerio de Transporte Colombiano, *Resolución No. 1690 del Ministerio de Transporte de Colombia, "Por la cual se adopta la ficha técnica Placa Única Nacional para vehículos de servicio Diplomático, Consular y de Misiones Especiales acreditados ante el Gobierno de Colombia."* Bogotá, DC.

[2] M. SYSTEMS, "What is Automatic Number Plate Recognition (ANPR)?" https://www.anprcameras.com/about-us/understanding-anpr/ (accessed Sep. 10, 2020).

[3] "Tema 4:Segmentación de imágenes."

[4] "Reconocimiento óptico de caracteres - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Reconocimiento_óptico_de_caracteres (accessed Sep. 10, 2020).

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 26 11:54:16 2020

@author: JuanManuel
"""
from skimage.io import imread
from skimage.filters import threshold_otsu
from skimage.transform import resize
from skimage import measure
from skimage.measure import regionprops
import matplotlib.pyplot as plt
import numpy as np
import cv2
import matplotlib.patches as patches
import os
import glob
from sklearn import svm
from skimage.transform import radon
# Modelos a comparar
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
# Métricas a utilizar
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import matthews_corrcoef

letters = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D','E', 'F', '

def KNN(X_train, X_test, y_train, y_test):
    #Obteniendo el mejor valor de K a partir del conjunto de validación con mejor accuracy
    scores = []
    maxscore = [0,0] # [k,score]
    for k in range(1, 40):
        knn = KNeighborsClassifier(n_neighbors = k,weights='distance',metric='euclidean', m
        knn.fit(X_train, y_train)
        score = knn.score(X_test, y_test)
        scores.append(score)
        if score > maxscore[1]:
            maxscore = [k,score]

    plt.figure()
    plt.xlabel('Parametro k')
    plt.ylabel('Accuracy')
    plt.scatter(range(1,40), scores)

    print('El mejor resultado fue:',maxscore)
    knn = KNeighborsClassifier(n_neighbors = maxscore[0],weights='distance',metric='euclide
    knn.fit(X_train, y_train)

    pred = knn.predict(X_test)
    # Matriz de confusion
    print(confusion_matrix(y_test, pred))
    # Reporte de clasificación
```

```
    print(classification_report(y_test, pred))
    # Matthews
    print('Matthews: ',matthews_corrcoef(y_test, pred))
    return knn

def SVM(X_train, X_test, y_train, y_test):
    kernels=['linear', 'poly', 'rbf', 'sigmoid']
    max_score = 0
    max_kernel = 500
    max_degree = 500
    for Kernel in range(4):
        if Kernel == 1:
            for Degree in range(1,11):
                msv = svm.SVC(kernel=kernels[Kernel],degree=Degree)
                msv.fit(X_train, y_train)
                score = msv.score(X_test, y_test)
                if score > max_score:
                    max_score = score
                    max_kernel = Kernel
                    max_degree = Degree
        else:
            msv = svm.SVC(kernel=kernels[Kernel])
            msv.fit(X_train, y_train)
            score = msv.score(X_test, y_test)
            if score > max_score:
                    max_score = score
                    max_kernel = Kernel

    print('El mejor Kernel es', kernels[max_kernel],'de grado',max_degree,'con un accuracy
    msv = svm.SVC(kernel=kernels[max_kernel])
    msv.fit(X_train, y_train)
    print('Accuracy of SVM classifier on test set: {:.2f}'
         .format(msv.score(X_test, y_test)))
    return msv

def NN(X_train, X_test, y_train, y_test):
    clf = MLPClassifier(hidden_layer_sizes=(40,40,30,15,8),activation='relu',random_state=1
    clf.fit(X_train, y_train)
    print('Accuracy por Red Neuronal con sklearn en conjunto de entrenamiento: {:.2f}'.form
    print('Accuracy por Red Neuronal con sklearn en conjunto de validación: {:.2f}'.format(
    # Se generan las métricas
    pred = clf.predict(X_test)
    # Matriz de confusion
    print(confusion_matrix(y_test, pred))
    # Reporte de clasificación
    print(classification_report(y_test, pred))
    # Matthews
    print('Matthews: ',matthews_corrcoef(y_test, pred))
    return clf

def read_training_data(training_directory):
    image_data = []
    target_data = []
    for each_letter in letters:
        for each in range(10):
            image_path = os.path.join(training_directory, each_letter, each_letter + '_' +
            # read each image of each character
```

```
            img_details = imread(image_path, as_gray=True)
            binary_image = img_details < threshold_otsu(img_details)
            flat_bin_image = binary_image.reshape(-1)
            image_data.append(flat_bin_image)
            target_data.append(each_letter)

    abc = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D',
                   'E', 'F', 'G', 'H','I', 'J', 'K', 'L', 'M', 'N','O', 'P', 'Q', 'R', 'S'
                   'U', 'V', 'W', 'X', 'Y', 'Z']
    nums = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29
    for letra in range(len(target_data)):
        for x in range(len(abc)):
            if target_data[letra] == abc[x]:
                target_data[letra] = nums[x]
    return (np.array(image_data), np.array(target_data))

def Imp_Placa(knn,msv,clf):
    print('\nPredicciones:')
    print('----------------------')
    print('KNN\t   |SVM\t   |NN')
    for x in range(len(knn)):
        print(knn[x],end='')
    print('\t',end='|')
    for x in range(len(knn)):
        print(msv[x],end='')
    print('\t',end='|')
    for x in range(len(knn)):
        print(clf[x],end='')
    print('\n----------------------')

# MAIN
print('Reading data')
#Xtotal, ytotal = Base_Datos()
training_dataset_dir = './training_images'
Xtotal, ytotal = read_training_data(training_dataset_dir)
print('Reading data completed')

# Se generan los conjuntos de entrenamiento y validación
X_train, X_test, y_train, y_test = train_test_split(Xtotal, ytotal,random_state = 1)
scaler= MinMaxScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

knn = KNN(X_train, X_test, y_train, y_test)
msv = SVM(X_train, X_test, y_train, y_test)
clf = NN(X_train, X_test, y_train, y_test)

for name in glob.glob('placas/*'):
    car_image = imread(name)
    gray = cv2.cvtColor(car_image, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray,(1,1),0)
    ret,binaria = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

    threshold_value = threshold_otsu(gray)
    binary_car_image = gray > threshold_value
    # fig, (ax1, ax2) = plt.subplots(1, 2)
```

```
# ax1.imshow(gray, cmap="gray")
# ax2.imshow(binary_car_image, cmap="gray")

# this gets all the connected regions and groups them together
label_image = measure.label(binary_car_image)

plate_dimensions = (0.06*label_image.shape[0], 0.1*label_image.shape[0], 0.06*label_ima
min_height, max_height, min_width, max_width = plate_dimensions
plate_objects_cordinates = []
plate_like_objects = []

fig, (ax1) = plt.subplots(1)
ax1.imshow(car_image, cmap="gray")


for region in regionprops(label_image):
    if region.area <50:
        continue

    min_row, min_col, max_row, max_col = region.bbox
    region_height = max_row - min_row
    region_width = max_col - min_col
    if region_height >= min_height and region_height <= max_height and region_width >=
        plate_like_objects.append(binary_car_image[min_row:max_row,
                                  min_col:max_col])
        plate_objects_cordinates.append((min_row, min_col,
                                         max_row, max_col))
        rectBorder = patches.Rectangle((min_col, min_row), max_col - min_col, max_row -
        ax1.add_patch(rectBorder)
plt.show()


for i in range(len(plate_like_objects)):
    license_plate = np.invert(plate_like_objects[i])
    labelled_plate = measure.label(license_plate)
    character_dimensions = (0.3*license_plate.shape[0], 0.55*license_plate.shape[0], 0.
    min_height, max_height, min_width, max_width = character_dimensions

    fig, ax1 = plt.subplots(1)
    ax1.imshow(license_plate, cmap="gray")

    characters = []
    counter=0
    column_list = []
    for regions in regionprops(labelled_plate):
        y0, x0, y1, x1 = regions.bbox
        y0 -= 1
        region_height = y1 - y0
        region_width = x1 - x0

        if region_height > min_height and region_height < max_height and region_width >
            roi = license_plate[y0:y1, x0:x1]

            rect_border = patches.Rectangle((x0, y0), x1 - x0, y1 - y0, edgecolor="red"
                                            linewidth=2, fill=False)
            ax1.add_patch(rect_border)
```

```
        # resize the characters to 20X20 and then append each character into the ch
        resized_char = resize(roi, (20, 20))
        image_data = resized_char.flatten()
        # Tradon = radon(resized_char, theta=np.arange(0,210,30), circle=True).flat
        # Firma = resized_char.flatten()
        # Hu = cv2.HuMoments(cv2.moments(resized_char)).flatten()
        # image_data = np.concatenate((Tradon, Firma,Hu), axis=None)
        characters.append(image_data)
plt.show()

X = np.array(characters)
X = scaler.transform(X)
knn_pred = list(knn.predict(X))
msv_pred = list(msv.predict(X))
clf_pred = list(clf.predict(X))

letters = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D',
           'E', 'F', 'G', 'H','I', 'J', 'K', 'L', 'M', 'N','O', 'P', 'Q', 'R', 'S'
           'U', 'V', 'W', 'X', 'Y', 'Z']
nums = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,2
for letra in range(len(knn_pred)):
    for x in range(len(nums)):
        if knn_pred[letra] == nums[x]:
            knn_pred[letra] = letters[x]
        if msv_pred[letra] == nums[x]:
            msv_pred[letra] = letters[x]
        if clf_pred[letra] == nums[x]:
            clf_pred[letra] = letters[x]

Imp_Placa(knn_pred,msv_pred,clf_pred)
```