Maestría en Inteligencia Artificial Aplicada

Curso: Navegación autónoma

Tecnológico de Monterrey

Dr. David Antonio Torres

Actividad 4.2 - Detección de Señales de Tránsito

Nombres y matrículas de los integrantes del equipo:

- Julio Cesar Lynn Jimenez A01793660
- Francisco Javier Parga García A01794380
- Carlos Roberto Torres Ferguson A01215432
- Fernando Sebastian Sanchez Cardona A01687530

Clasificación de señales de transito con CNN

> Mediante un modelo convolucional, se creó u clasificador para identificar el tipo de señalización, utilizando los datos para entrenamiento de GTRSB

In [1]:
```
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense, Dropout, BatchNormalization, ReLU
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import load_model
import pandas as pd
import numpy as np
import tensorflow as tf

import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['figure.figsize'] = (5,3)
```

Datos para entrenamiento y validación

In [2]:
```
train_directory = 'data/Train'
test_directory = 'data/Test'
```

In [3]:
```
# Crea un generador con aumento de datos
train_datagen = ImageDataGenerator(rescale=1./255,
                                   horizontal_flip=True,
                                   vertical_flip=False,
                                   fill_mode='nearest',
                                   validation_split=0.25)
```

In [4]:
```
# Definicion de variables para batch y tamaño de imagen a usar
batch_size = 128
image_size = (28,28)

# Keras ImageDataGenerator proporciona un método flow_from_directory para cargar imágenes en lotes
# Esto es útil para conjuntos de datos grandes que no caben en la memoria
train_generator = train_datagen.flow_from_directory(train_directory,
                                                    target_size=image_size,
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    classes= list(map(str,range(0,43))),
                                                    subset='training')  # establece como datos de entrenamiento

validation_generator = train_datagen.flow_from_directory(train_directory,  # mismo directorio que los datos de entrenamiento
                                                         target_size=image_size,
                                                         batch_size=batch_size,
                                                         class_mode='categorical',
                                                         classes=  list(map(str,range(0,43))),
                                                         subset='validation')  # establece como datos de validación
```

```
Found 29416 images belonging to 43 classes.
Found 9793 images belonging to 43 classes.
```

Modelo CNN para clsificación

In [5]:

```
#Modelo CNN
def initialize_model():
    model = Sequential()

    ### Primera convolución y MaxPooling
    model.add(Conv2D(32, (3,3),padding = "same", input_shape=image_size+(3,), activation="relu"))
    model.add(Conv2D(32, (3,3), activation="relu"))
    model.add(MaxPool2D(pool_size=(2,2)))
    #model.add(BatchNormalization())
    model.add(Dropout(0.25))

    ### Tercera convolución
    model.add(Conv2D(64, (3,3), padding='same', activation="relu"))
    model.add(Conv2D(64, (3,3), activation="relu"))
    model.add(MaxPool2D(pool_size=(2,2)))
    #model.add(BatchNormalization())
    model.add(Dropout(0.25))

    ### Aplanamiento
    model.add(Flatten())

    ### Una capa "Dense"
    model.add(Dense(392, activation='relu'))
    model.add(Dropout(0.5))


    ### Última capa con 43 salidas
    model.add(Dense(43, activation='softmax')) #Softmax(43)
    return model


def compile_model(model):

    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy', 'Recall', 'Precision'])
    return model
```

In [6]:

```
model_keras = initialize_model()
model_keras = compile_model(model_keras)

model_keras.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        896       
                                                                 
 conv2d_1 (Conv2D)           (None, 26, 26, 32)        9248      
                                                                 
 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0         
 )                                                               
                                                                 
 dropout (Dropout)           (None, 13, 13, 32)        0         
                                                                 
 conv2d_2 (Conv2D)           (None, 13, 13, 64)        18496     
                                                                 
 conv2d_3 (Conv2D)           (None, 11, 11, 64)        36928     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0         
 2D)                                                             
                                                                 
 dropout_1 (Dropout)         (None, 5, 5, 64)          0         
                                                                 
 flatten (Flatten)           (None, 1600)              0         
                                                                 
 dense (Dense)               (None, 392)               627592    
                                                                 
 dropout_2 (Dropout)         (None, 392)               0         
                                                                 
 dense_1 (Dense)             (None, 43)                16899     
                                                                 
=================================================================
Total params: 710,059
Trainable params: 710,059
Non-trainable params: 0
_________________________________________________________________
```

In [7]:

```
# Crea un callback de EarlyStopping que detiene el entrenamiento cuando el accuracy de validación deja de mejorar
early_stop = EarlyStopping(monitor = 'val_accuracy',
                           mode = 'max',
                           patience = 7,
                           restore_best_weights = True,
                           verbose=1)

# Crea un callback de ModelCheckpoint que guarda el modelo con la mejor accuracy de validación
model_checkpoint = ModelCheckpoint('best_model.h5',
                                   monitor='val_accuracy',
                                   mode='max',
```

```
                                      verbose=1,
                                      save_best_only=True)
```

Entrenamientoo de modelo CNN

In [8]:
```
%%time

history = model_keras.fit(train_generator,
                              epochs=101,
                              validation_data=validation_generator,
                              callbacks=[early_stop, model_checkpoint])
```

```
Epoch 1/101
229/230 [============================>.] - ETA: 0s - loss: 2.2237 - accuracy: 0.3723 - recall: 0.2011 - precision: 0.7761
Epoch 1: val_accuracy improved from -inf to 0.65006, saving model to best_model.h5
230/230 [==============================] - 17s 60ms/step - loss: 2.2190 - accuracy: 0.3734 - recall: 0.2024 - precision: 0.7761 - val_loss:
1.1673 - val_accuracy: 0.6501 - val_recall: 0.4419 - val_precision: 0.8339
Epoch 2/101
230/230 [==============================] - ETA: 0s - loss: 0.7952 - accuracy: 0.7392 - recall: 0.6337 - precision: 0.8521
Epoch 2: val_accuracy improved from 0.65006 to 0.78679, saving model to best_model.h5
230/230 [==============================] - 13s 57ms/step - loss: 0.7952 - accuracy: 0.7392 - recall: 0.6337 - precision: 0.8521 - val_loss:
0.6740 - val_accuracy: 0.7868 - val_recall: 0.7178 - val_precision: 0.8693
Epoch 3/101
229/230 [============================>.] - ETA: 0s - loss: 0.4520 - accuracy: 0.8499 - recall: 0.8032 - precision: 0.8981
Epoch 3: val_accuracy improved from 0.78679 to 0.84571, saving model to best_model.h5
230/230 [==============================] - 13s 56ms/step - loss: 0.4525 - accuracy: 0.8496 - recall: 0.8030 - precision: 0.8979 - val_loss:
0.5391 - val_accuracy: 0.8457 - val_recall: 0.8028 - val_precision: 0.8942
Epoch 4/101
229/230 [============================>.] - ETA: 0s - loss: 0.3291 - accuracy: 0.8910 - recall: 0.8612 - precision: 0.9219
Epoch 4: val_accuracy improved from 0.84571 to 0.85980, saving model to best_model.h5
230/230 [==============================] - 13s 55ms/step - loss: 0.3290 - accuracy: 0.8910 - recall: 0.8612 - precision: 0.9220 - val_loss:
0.5034 - val_accuracy: 0.8598 - val_recall: 0.8250 - val_precision: 0.8999
Epoch 5/101
230/230 [==============================] - ETA: 0s - loss: 0.2578 - accuracy: 0.9136 - recall: 0.8942 - precision: 0.9338
Epoch 5: val_accuracy improved from 0.85980 to 0.86878, saving model to best_model.h5
230/230 [==============================] - 12s 53ms/step - loss: 0.2578 - accuracy: 0.9136 - recall: 0.8942 - precision: 0.9338 - val_loss:
0.4518 - val_accuracy: 0.8688 - val_recall: 0.8416 - val_precision: 0.9062
Epoch 6/101
230/230 [==============================] - ETA: 0s - loss: 0.2201 - accuracy: 0.9263 - recall: 0.9117 - precision: 0.9424
Epoch 6: val_accuracy improved from 0.86878 to 0.88145, saving model to best_model.h5
230/230 [==============================] - 12s 51ms/step - loss: 0.2201 - accuracy: 0.9263 - recall: 0.9117 - precision: 0.9424 - val_loss:
0.4409 - val_accuracy: 0.8814 - val_recall: 0.8616 - val_precision: 0.9089
Epoch 7/101
230/230 [==============================] - ETA: 0s - loss: 0.1803 - accuracy: 0.9398 - recall: 0.9290 - precision: 0.9520
Epoch 7: val_accuracy improved from 0.88145 to 0.88400, saving model to best_model.h5
230/230 [==============================] - 12s 52ms/step - loss: 0.1803 - accuracy: 0.9398 - recall: 0.9290 - precision: 0.9520 - val_loss:
0.4177 - val_accuracy: 0.8840 - val_recall: 0.8669 - val_precision: 0.9110
Epoch 8/101
230/230 [==============================] - ETA: 0s - loss: 0.1634 - accuracy: 0.9460 - recall: 0.9364 - precision: 0.9560
Epoch 8: val_accuracy improved from 0.88400 to 0.89053, saving model to best_model.h5
230/230 [==============================] - 12s 52ms/step - loss: 0.1634 - accuracy: 0.9460 - recall: 0.9364 - precision: 0.9560 - val_loss:
0.4098 - val_accuracy: 0.8905 - val_recall: 0.8756 - val_precision: 0.9154
Epoch 9/101
229/230 [============================>.] - ETA: 0s - loss: 0.1469 - accuracy: 0.9516 - recall: 0.9434 - precision: 0.9601
Epoch 9: val_accuracy did not improve from 0.89053
230/230 [==============================] - 12s 51ms/step - loss: 0.1466 - accuracy: 0.9517 - recall: 0.9435 - precision: 0.9601 - val_loss:
0.4746 - val_accuracy: 0.8867 - val_recall: 0.8739 - val_precision: 0.9055
Epoch 10/101
229/230 [============================>.] - ETA: 0s - loss: 0.1327 - accuracy: 0.9563 - recall: 0.9488 - precision: 0.9640
Epoch 10: val_accuracy improved from 0.89053 to 0.90473, saving model to best_model.h5
```

```
230/230 [==============================] - 12s 54ms/step - loss: 0.1327 - accuracy: 0.9564 - recall: 0.9489 - precision: 0.9640 - val_loss:
0.3591 - val_accuracy: 0.9047 - val_recall: 0.8915 - val_precision: 0.9267
Epoch 11/101
230/230 [==============================] - ETA: 0s - loss: 0.1196 - accuracy: 0.9590 - recall: 0.9536 - precision: 0.9660
Epoch 11: val_accuracy did not improve from 0.90473
230/230 [==============================] - 13s 54ms/step - loss: 0.1196 - accuracy: 0.9590 - recall: 0.9536 - precision: 0.9660 - val_loss:
0.3925 - val_accuracy: 0.8998 - val_recall: 0.8896 - val_precision: 0.9182
Epoch 12/101
230/230 [==============================] - ETA: 0s - loss: 0.1093 - accuracy: 0.9633 - recall: 0.9584 - precision: 0.9692
Epoch 12: val_accuracy improved from 0.90473 to 0.90851, saving model to best_model.h5
230/230 [==============================] - 12s 54ms/step - loss: 0.1093 - accuracy: 0.9633 - recall: 0.9584 - precision: 0.9692 - val_loss:
0.3908 - val_accuracy: 0.9085 - val_recall: 0.9011 - val_precision: 0.9237
Epoch 13/101
229/230 [============================>.] - ETA: 0s - loss: 0.1051 - accuracy: 0.9651 - recall: 0.9611 - precision: 0.9713
Epoch 13: val_accuracy did not improve from 0.90851
230/230 [==============================] - 13s 55ms/step - loss: 0.1051 - accuracy: 0.9651 - recall: 0.9611 - precision: 0.9713 - val_loss:
0.4064 - val_accuracy: 0.9015 - val_recall: 0.8907 - val_precision: 0.9212
Epoch 14/101
229/230 [============================>.] - ETA: 0s - loss: 0.0958 - accuracy: 0.9693 - recall: 0.9652 - precision: 0.9737
Epoch 14: val_accuracy improved from 0.90851 to 0.90922, saving model to best_model.h5
230/230 [==============================] - 13s 55ms/step - loss: 0.0957 - accuracy: 0.9693 - recall: 0.9653 - precision: 0.9737 - val_loss:
0.3793 - val_accuracy: 0.9092 - val_recall: 0.8986 - val_precision: 0.9252
Epoch 15/101
229/230 [============================>.] - ETA: 0s - loss: 0.0943 - accuracy: 0.9692 - recall: 0.9650 - precision: 0.9734
Epoch 15: val_accuracy improved from 0.90922 to 0.91525, saving model to best_model.h5
230/230 [==============================] - 13s 55ms/step - loss: 0.0943 - accuracy: 0.9691 - recall: 0.9649 - precision: 0.9733 - val_loss:
0.3837 - val_accuracy: 0.9152 - val_recall: 0.9093 - val_precision: 0.9272
Epoch 16/101
230/230 [==============================] - ETA: 0s - loss: 0.0863 - accuracy: 0.9713 - recall: 0.9677 - precision: 0.9750
Epoch 16: val_accuracy improved from 0.91525 to 0.92137, saving model to best_model.h5
230/230 [==============================] - 13s 55ms/step - loss: 0.0863 - accuracy: 0.9713 - recall: 0.9677 - precision: 0.9750 - val_loss:
0.3321 - val_accuracy: 0.9214 - val_recall: 0.9147 - val_precision: 0.9355
Epoch 17/101
229/230 [============================>.] - ETA: 0s - loss: 0.0806 - accuracy: 0.9727 - recall: 0.9691 - precision: 0.9760
Epoch 17: val_accuracy did not improve from 0.92137
230/230 [==============================] - 13s 56ms/step - loss: 0.0807 - accuracy: 0.9726 - recall: 0.9691 - precision: 0.9760 - val_loss:
0.3914 - val_accuracy: 0.9102 - val_recall: 0.9028 - val_precision: 0.9239
Epoch 18/101
230/230 [==============================] - ETA: 0s - loss: 0.0752 - accuracy: 0.9751 - recall: 0.9725 - precision: 0.9780
Epoch 18: val_accuracy did not improve from 0.92137
230/230 [==============================] - 13s 57ms/step - loss: 0.0752 - accuracy: 0.9751 - recall: 0.9725 - precision: 0.9780 - val_loss:
0.3759 - val_accuracy: 0.9120 - val_recall: 0.9046 - val_precision: 0.9274
Epoch 19/101
229/230 [============================>.] - ETA: 0s - loss: 0.0730 - accuracy: 0.9751 - recall: 0.9731 - precision: 0.9783
Epoch 19: val_accuracy did not improve from 0.92137
230/230 [==============================] - 13s 57ms/step - loss: 0.0734 - accuracy: 0.9750 - recall: 0.9729 - precision: 0.9782 - val_loss:
0.3318 - val_accuracy: 0.9181 - val_recall: 0.9095 - val_precision: 0.9290
Epoch 20/101
```

```
230/230 [==============================] - ETA: 0s - loss: 0.0715 - accuracy: 0.9761 - recall: 0.9732 - precision: 0.9795
Epoch 20: val_accuracy did not improve from 0.92137
230/230 [==============================] - 13s 57ms/step - loss: 0.0715 - accuracy: 0.9761 - recall: 0.9732 - precision: 0.9795 - val_loss:
0.4076 - val_accuracy: 0.9086 - val_recall: 0.9022 - val_precision: 0.9249
Epoch 21/101
229/230 [============================>.] - ETA: 0s - loss: 0.0683 - accuracy: 0.9770 - recall: 0.9744 - precision: 0.9797
Epoch 21: val_accuracy improved from 0.92137 to 0.92362, saving model to best_model.h5
230/230 [==============================] - 13s 57ms/step - loss: 0.0685 - accuracy: 0.9768 - recall: 0.9743 - precision: 0.9796 - val_loss:
0.3607 - val_accuracy: 0.9236 - val_recall: 0.9199 - val_precision: 0.9326
Epoch 22/101
229/230 [============================>.] - ETA: 0s - loss: 0.0647 - accuracy: 0.9779 - recall: 0.9758 - precision: 0.9809
Epoch 22: val_accuracy did not improve from 0.92362
230/230 [==============================] - 13s 57ms/step - loss: 0.0648 - accuracy: 0.9778 - recall: 0.9758 - precision: 0.9808 - val_loss:
0.3718 - val_accuracy: 0.9193 - val_recall: 0.9136 - val_precision: 0.9293
Epoch 23/101
230/230 [==============================] - ETA: 0s - loss: 0.0618 - accuracy: 0.9790 - recall: 0.9771 - precision: 0.9812
Epoch 23: val_accuracy did not improve from 0.92362
230/230 [==============================] - 14s 59ms/step - loss: 0.0618 - accuracy: 0.9790 - recall: 0.9771 - precision: 0.9812 - val_loss:
0.4598 - val_accuracy: 0.9147 - val_recall: 0.9079 - val_precision: 0.9243
Epoch 24/101
230/230 [==============================] - ETA: 0s - loss: 0.0578 - accuracy: 0.9805 - recall: 0.9788 - precision: 0.9830
Epoch 24: val_accuracy did not improve from 0.92362
230/230 [==============================] - 14s 59ms/step - loss: 0.0578 - accuracy: 0.9805 - recall: 0.9788 - precision: 0.9830 - val_loss:
0.3831 - val_accuracy: 0.9174 - val_recall: 0.9123 - val_precision: 0.9288
Epoch 25/101
229/230 [============================>.] - ETA: 0s - loss: 0.0608 - accuracy: 0.9797 - recall: 0.9776 - precision: 0.9819
Epoch 25: val_accuracy improved from 0.92362 to 0.92495, saving model to best_model.h5
230/230 [==============================] - 14s 59ms/step - loss: 0.0609 - accuracy: 0.9795 - recall: 0.9775 - precision: 0.9818 - val_loss:
0.3512 - val_accuracy: 0.9249 - val_recall: 0.9209 - val_precision: 0.9351
Epoch 26/101
229/230 [============================>.] - ETA: 0s - loss: 0.0551 - accuracy: 0.9819 - recall: 0.9803 - precision: 0.9840
Epoch 26: val_accuracy did not improve from 0.92495
230/230 [==============================] - 14s 60ms/step - loss: 0.0553 - accuracy: 0.9819 - recall: 0.9803 - precision: 0.9839 - val_loss:
0.4730 - val_accuracy: 0.9122 - val_recall: 0.9080 - val_precision: 0.9221
Epoch 27/101
230/230 [==============================] - ETA: 0s - loss: 0.0538 - accuracy: 0.9820 - recall: 0.9802 - precision: 0.9839
Epoch 27: val_accuracy improved from 0.92495 to 0.92944, saving model to best_model.h5
230/230 [==============================] - 14s 60ms/step - loss: 0.0538 - accuracy: 0.9820 - recall: 0.9802 - precision: 0.9839 - val_loss:
0.3188 - val_accuracy: 0.9294 - val_recall: 0.9231 - val_precision: 0.9392
Epoch 28/101
230/230 [==============================] - ETA: 0s - loss: 0.0565 - accuracy: 0.9802 - recall: 0.9789 - precision: 0.9826
Epoch 28: val_accuracy did not improve from 0.92944
230/230 [==============================] - 14s 59ms/step - loss: 0.0565 - accuracy: 0.9802 - recall: 0.9789 - precision: 0.9826 - val_loss:
0.4257 - val_accuracy: 0.9180 - val_recall: 0.9137 - val_precision: 0.9252
Epoch 29/101
230/230 [==============================] - ETA: 0s - loss: 0.0534 - accuracy: 0.9825 - recall: 0.9811 - precision: 0.9841
Epoch 29: val_accuracy did not improve from 0.92944
230/230 [==============================] - 14s 59ms/step - loss: 0.0534 - accuracy: 0.9825 - recall: 0.9811 - precision: 0.9841 - val_loss:
```

```
0.4267 - val_accuracy: 0.9125 - val_recall: 0.9089 - val_precision: 0.9215
Epoch 30/101
229/230 [============================>.] - ETA: 0s - loss: 0.0521 - accuracy: 0.9830 - recall: 0.9816 - precision: 0.9848
Epoch 30: val_accuracy did not improve from 0.92944
230/230 [==============================] - 14s 60ms/step - loss: 0.0523 - accuracy: 0.9830 - recall: 0.9816 - precision: 0.9848 - val_loss:
0.3900 - val_accuracy: 0.9261 - val_recall: 0.9236 - val_precision: 0.9323
Epoch 31/101
230/230 [==============================] - ETA: 0s - loss: 0.0461 - accuracy: 0.9842 - recall: 0.9831 - precision: 0.9860
Epoch 31: val_accuracy did not improve from 0.92944
230/230 [==============================] - 14s 60ms/step - loss: 0.0461 - accuracy: 0.9842 - recall: 0.9831 - precision: 0.9860 - val_loss:
0.3319 - val_accuracy: 0.9284 - val_recall: 0.9233 - val_precision: 0.9362
Epoch 32/101
230/230 [==============================] - ETA: 0s - loss: 0.0524 - accuracy: 0.9833 - recall: 0.9821 - precision: 0.9852
Epoch 32: val_accuracy improved from 0.92944 to 0.93414, saving model to best_model.h5
230/230 [==============================] - 14s 60ms/step - loss: 0.0524 - accuracy: 0.9833 - recall: 0.9821 - precision: 0.9852 - val_loss:
0.2969 - val_accuracy: 0.9341 - val_recall: 0.9291 - val_precision: 0.9407
Epoch 33/101
229/230 [============================>.] - ETA: 0s - loss: 0.0510 - accuracy: 0.9836 - recall: 0.9823 - precision: 0.9849
Epoch 33: val_accuracy did not improve from 0.93414
230/230 [==============================] - 14s 60ms/step - loss: 0.0509 - accuracy: 0.9837 - recall: 0.9824 - precision: 0.9850 - val_loss:
0.3916 - val_accuracy: 0.9227 - val_recall: 0.9181 - val_precision: 0.9321
Epoch 34/101
230/230 [==============================] - ETA: 0s - loss: 0.0506 - accuracy: 0.9835 - recall: 0.9822 - precision: 0.9854
Epoch 34: val_accuracy did not improve from 0.93414
230/230 [==============================] - 15s 64ms/step - loss: 0.0506 - accuracy: 0.9835 - recall: 0.9822 - precision: 0.9854 - val_loss:
0.3737 - val_accuracy: 0.9253 - val_recall: 0.9213 - val_precision: 0.9318
Epoch 35/101
230/230 [==============================] - ETA: 0s - loss: 0.0421 - accuracy: 0.9856 - recall: 0.9845 - precision: 0.9870
Epoch 35: val_accuracy did not improve from 0.93414
230/230 [==============================] - 14s 63ms/step - loss: 0.0421 - accuracy: 0.9856 - recall: 0.9845 - precision: 0.9870 - val_loss:
0.4074 - val_accuracy: 0.9254 - val_recall: 0.9217 - val_precision: 0.9311
Epoch 36/101
230/230 [==============================] - ETA: 0s - loss: 0.0459 - accuracy: 0.9848 - recall: 0.9835 - precision: 0.9858
Epoch 36: val_accuracy did not improve from 0.93414
230/230 [==============================] - 14s 62ms/step - loss: 0.0459 - accuracy: 0.9848 - recall: 0.9835 - precision: 0.9858 - val_loss:
0.4383 - val_accuracy: 0.9122 - val_recall: 0.9069 - val_precision: 0.9239
Epoch 37/101
230/230 [==============================] - ETA: 0s - loss: 0.0441 - accuracy: 0.9849 - recall: 0.9840 - precision: 0.9862
Epoch 37: val_accuracy did not improve from 0.93414
230/230 [==============================] - 14s 62ms/step - loss: 0.0441 - accuracy: 0.9849 - recall: 0.9840 - precision: 0.9862 - val_loss:
0.3715 - val_accuracy: 0.9262 - val_recall: 0.9227 - val_precision: 0.9331
Epoch 38/101
230/230 [==============================] - ETA: 0s - loss: 0.0435 - accuracy: 0.9851 - recall: 0.9841 - precision: 0.9869
Epoch 38: val_accuracy did not improve from 0.93414
230/230 [==============================] - 14s 61ms/step - loss: 0.0435 - accuracy: 0.9851 - recall: 0.9841 - precision: 0.9869 - val_loss:
0.3656 - val_accuracy: 0.9305 - val_recall: 0.9277 - val_precision: 0.9372
Epoch 39/101
229/230 [============================>.] - ETA: 0s - loss: 0.0434 - accuracy: 0.9857 - recall: 0.9848 - precision: 0.9869Restoring model we
```

```
ights from the end of the best epoch: 32.

Epoch 39: val_accuracy did not improve from 0.93414
230/230 [==============================] - 13s 56ms/step - loss: 0.0435 - accuracy: 0.9856 - recall: 0.9848 - precision: 0.9869 - val_loss: 0.4024 - val_accuracy: 0.9252 - val_recall: 0.9223 - val_precision: 0.9323
Epoch 39: early stopping
CPU times: total: 12min 40s
Wall time: 8min 38s
```

Resultados de entrenamiento

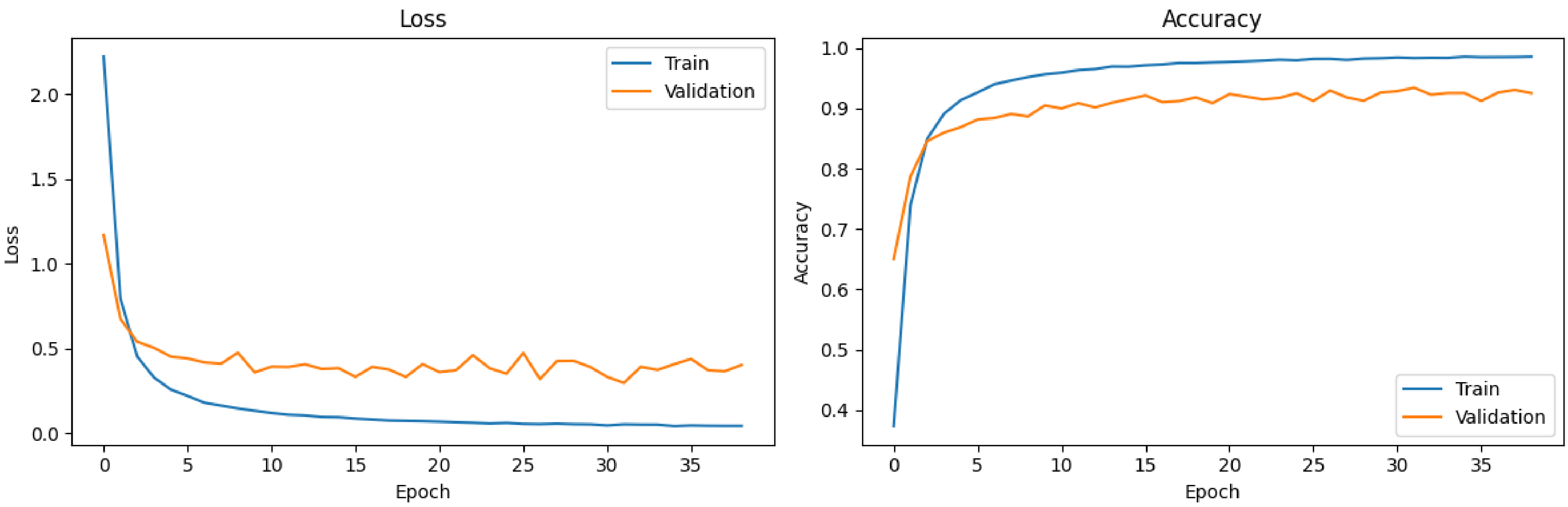
In [9]:

```python
# Plot the loss and accuracy curves
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])

plt.tight_layout()
plt.show()
```



Evalución de modelo de clasificación

In [6]:

```
# Cargar la tabla pandas con las rutas de las imágenes y las clases
df = pd.read_csv("data/Test.csv")

# Definir una función que carga una imagen desde una ruta y la preprocesa
def load_image(path):
  # Leer el archivo de imagen
  image = tf.io.read_file(path)
  # Decodificar la imagen como JPEG
  image = tf.image.decode_png(image, channels=3)
  # Cambiar el tamaño de la imagen a un tamaño fijo
  image = tf.image.resize(image, image_size)
  # Normalizar los valores de píxeles al rango [0, 1]
  image = image / 255.0
  return image

# Definir una función que carga un lote de imágenes y etiquetas desde la tabla pandas
def load_batch(df, batch_size):
  # Barajar la tabla
  df = df.sample(frac=1).reset_index(drop=True)
  # Recorrer la tabla en lotes
  for i in range(0, len(df), batch_size):
    # Obtener el lote de rutas y clases
    batch_paths = df["Path"][i:i+batch_size]
    batch_classes = df["ClassId"][i:i+batch_size]
```

```python
    # Cargar el lote de imágenes
    batch_images = [load_image('data/' + path) for path in batch_paths]
    # Convertir el lote de imágenes y etiquetas en tensores
    batch_images = tf.stack(batch_images)
    # Convertir el lote de etiquetas en tensores categóricos
    batch_labels = tf.keras.utils.to_categorical(batch_classes, num_classes=43)
    yield batch_images, batch_labels

# Definir una función de cargador de datos que usa la función load_batch
def data_loader(df, batch_size):
  # Crear un conjunto de datos a partir de la función load_batch
  dataset = tf.data.Dataset.from_generator(
                                        lambda: load_batch(df, batch_size),
                                        output_types=(tf.float32, tf.float32),
                                        output_shapes=([None, image_size[0], image_size[1], 3], [None, None])
  )
  return dataset

# Crear un cargador de datos con un tamaño de lote de 32
data_loader = data_loader(df, 32)
```

In [7]:
```python
# Cargar el mejor modelo
model = tf.keras.models.load_model("best_model.h5")

# Evlauar el modelo con los datos de Test
model.evaluate(data_loader)
```

```
395/395 [==============================] - 113s 277ms/step - loss: 0.2606 - accuracy: 0.9435 - recall: 0.9408 - precision: 0.9494
```

Out[7]:
```
[0.2606205344200134, 0.943467915058136, 0.9407759308815002, 0.949420690536499]
```

Modelo para detección de señales en el video

> Para complementar la clasificación de las señales de tránsito, se utilizó el set de datos de GTSDB (German Traffic Sign Detection Benchmark) el cual es una evaluación de detección de imágenes individuales para investigadores con interés en el campo de la visión por computadora, el reconocimiento de patrones y la asistencia al conductor basada en imágenes. Se supone que se presentará en la IEEE International Joint Conference on Neural Networks 2013:
>
> https://benchmark.ini.rub.de/gtsdb_news.html

Datos de entrenamiento

In [5]:

```
path_dir="data/YOLO_data/FullIJCNN2013/"

data = pd.read_csv(path_dir+'gt.txt',sep=';',names=['path','left','top','right','bottom','id'])
data.head()
```

Out[5]:

	path	left	top	right	bottom	id
0	00000.ppm	774	411	815	446	11
1	00001.ppm	983	388	1024	432	40
2	00001.ppm	386	494	442	552	38
3	00001.ppm	973	335	1031	390	13
4	00002.ppm	892	476	1006	592	39

In [7]:

```
from PIL import Image
#opening image and converting it into numpy array and checking the size of array.
img = Image.open(path_dir+data['path'][0])
img=np.array(img)
print(img.shape)

img = Image.open(path_dir+data['path'][1])
img=np.array(img)
plt.axis('off')
plt.imshow(img[:,:,:])
plt.show()
```

```
(800, 1360, 3)
```

In [8]:

```
#copiando la columna id de los datos en df
df = data['id'].copy()
```

```python
data ['Object Name'] = data['id']
#asignando una sola etiqueta para señal de trafico
for i in range(len(df)):
  if(df[i] in range(0,43)):
    df.loc[i]=0
    data['Object Name'].loc[i]='trafficsign'
  else:
    df.loc[i]=-1
data.head()
```

```
C:\Users\fgarcia24\AppData\Local\Temp\ipykernel_44008\1312309559.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['Object Name'].loc[i]='trafficsign'
```

Out[8]:

	path	left	top	right	bottom	id	Object Name
0	00000.ppm	774	411	815	446	11	trafficsign
1	00001.ppm	983	388	1024	432	40	trafficsign
2	00001.ppm	386	494	442	552	38	trafficsign
3	00001.ppm	973	335	1031	390	13	trafficsign
4	00002.ppm	892	476	1006	592	39	trafficsign

Ajustes de dato para el entrenamiento

In [9]:

```python
import cv2

#Remover la extension en la columna de path
data['path'] = data['path'].str.slice(0, -4)

# Usando os.walk para recorrer todos los directorios
# y archivos en ellos desde el directorio actual
path_dir="data/YOLO_data/FullIJCNN2013/"
final_df=pd.DataFrame(columns=data.columns)
df_list = [] # crear una lista vacía de DataFrames
for current_dir, dirs, files in os.walk(path_dir):
    for f in files:
        #print(f)
        #Comprobar si un archivo termina con formato .ppm o no
        if f.endswith('.ppm'):
            image_name=f[:-4]
            img=cv2.imread(path_dir + f)
```

```python
            single_yolo_dat = data.loc[data['path'] == image_name].copy()
            #print(single_yolo_dat)
            # y, de esta manera, el dataFrame inicial no se cambiará
            # Comprobando si no hay ninguna anotación para la imagen actual
            if single_yolo_dat.isnull().values.all():
                # Eliminando esta imagen de los datos de entrenamiento
                # print(f)
                os.remove(path_dir + f)

            #Ahora guardar el resulted_frame en una carpeta dentro de path_dir
            else:
              df_list.append(single_yolo_dat) # añadir a la lista de DataFrames
              #Ahora escribiendo y guardando la imagen de formato ppm a formato jpg usando OpenCV
              save_path = 'data/YOLO_data/jpeg_files/' + image_name + '.jpg'
              cv2.imwrite(save_path, img)

final_df = pd.concat(df_list) # concatenar la lista de DataFrames
final_df = final_df[~final_df.index.duplicated(keep='first')]
final_df.sort_index(inplace=True)
```

División del dato para entrenamiento y validación

In [11]:
```python
from sklearn.model_selection import train_test_split

train, test = train_test_split(final_df['path'].unique(), test_size=0.2, random_state=37)
train.shape, test.shape
```

Out[11]:
```
((592,), (149,))
```

Creación de etiquetas para entrenamiento y validación

In [ ]:
```python
labels_path = 'data/YOLO_data/labels/'

# Crear una carpeta para almacenar los archivos de etiquetas
os.makedirs(labels_path, exist_ok=True)
os.makedirs(labels_path+'train/', exist_ok=True)
os.makedirs(labels_path+'test/', exist_ok=True)

# Recorrer cada imagen en la tabla
for path, group in final_df.groupby('path'):
    # Obtener el nombre de la imagen sin extensión
    if path in train:
        name = labels_path+'train/'+path+'.txt'
    elif path in test:
```

```
        name = labels_path+'test/'+path+'.txt'
    # Abrir un archivo de texto con el mismo nombre que la imagen
    with open(name, 'w') as f:
        # Recorrer cada objeto en la imagen
        for row in group.itertuples():
            # Obtener el id de clase del objeto, las coordenadas izquierda, superior, derecha e inferior
            class_id = 0 #row.id   # modificado para detectar cualquier señal de tráfico
            left = row.left
            top = row.top
            right = row.right
            bottom = row.bottom
            # Calcular el centro normalizado x, y, ancho y alto del cuadro delimitador
            x = round((left + right) / 2 / 1360,2) # ancho de la imagen
            y = round((top + bottom) / 2 / 800,2) # altura de la imagen
            w = round((right - left) / 1360,2)
            h = round((bottom - top) / 800,2)

            # Escribir los datos de la etiqueta en el archivo de texto en formato YOLOv8
            f.write(f'{class_id} {x} {y} {w} {h}\n')
```

División de las imágenes deacuerdo a las listas de entrenamiento y validación

In [13]:
```
import shutil

jpeg_files = 'data/YOLO_data/jpeg_files/'
images_path = 'data/YOLO_data/images/'

# Crear las carpetas de entrenamiento y prueba si no existen
os.makedirs(images_path, exist_ok=True)
os.makedirs(images_path+'train/', exist_ok=True)
os.makedirs(images_path+'test/', exist_ok=True)

# Recorrer el array de entrenamiento y copiar las imágenes jpeg a la carpeta de entrenamiento
for image in train:
    # Obtener la ruta completa del archivo de imagen
    src = os.path.join(jpeg_files, image + '.jpg')
    # Copiar el archivo de imagen a la carpeta de entrenamiento
    shutil.copy(src, images_path+'train')

# Recorrer el array de prueba y copiar las imágenes jpeg a la carpeta de prueba
for image in test:
    # Obtener la ruta completa del archivo de imagen
    src = os.path.join(jpeg_files, image + '.jpg')
    # Copiar el archivo de imagen a la carpeta de prueba
    shutil.copy(src, images_path+'test')
```

Entrenamiento de modelo YOLO

In [14]:
```python
import torch
from ultralytics import YOLO

os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
```

In [15]:
```python
model = YOLO('yolov8s.yaml').load('yolov8s.pt')
```

```
                   from  n    params  module                                       arguments
  0                  -1  1       928  ultralytics.nn.modules.conv.Conv             [3, 32, 3, 2]
  1                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  2                  -1  1     29056  ultralytics.nn.modules.block.C2f             [64, 64, 1, True]
  3                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  4                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  5                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  6                  -1  2    788480  ultralytics.nn.modules.block.C2f             [256, 256, 2, True]
  7                  -1  1   1180672  ultralytics.nn.modules.conv.Conv             [256, 512, 3, 2]
  8                  -1  1   1838080  ultralytics.nn.modules.block.C2f             [512, 512, 1, True]
  9                  -1  1    656896  ultralytics.nn.modules.block.SPPF            [512, 512, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  1    591360  ultralytics.nn.modules.block.C2f             [768, 256, 1]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 16                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 19                  -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  1   1969152  ultralytics.nn.modules.block.C2f             [768, 512, 1]
 22        [15, 18, 21]  1   2147008  ultralytics.nn.modules.head.Detect           [80, [128, 256, 512]]
YOLOv8s summary: 225 layers, 11166560 parameters, 11166544 gradients

Transferred 355/355 items from pretrained weights
```

In [16]:
```python
%%time
results = model.train(data='yolo-trafficsigns.yaml', epochs=83, imgsz=640, verbose=True, patience=13, single_cls=True,
                      conf=0.5, iou=0.7,
                      project='trafficsigns_yolo8s_640', name='train', pretrained=True, optimizer='SGD',
                      dropout=0.1, cls=0.5,
                      device=0, workers=16, batch=32, save_conf=True,
                      augment=True, mosaic=False, mixup=False,
                      degrees=7, flipud=0, scale=0.1,
```

```
                        hsv_v=0.75, hsv_h=0.085, hsv_s=0.75
                       )
```

```
Ultralytics YOLOv8.0.106  Python-3.9.16 torch-2.0.1+cu117 CUDA:0 (NVIDIA RTX A4500 Laptop GPU, 16384MiB)
yolo\engine\trainer: task=detect, mode=train, model=yolov8s.yaml, data=yolo-trafficsigns.yaml, epochs=83, patience=13, batch=32, imgsz=640, save=True, save_period=-1, cache=False, device=0, workers=16, project=trafficsigns_yolo8s_640, name=train, exist_ok=False, pretrained=True, optimizer=SGD, verbose=True, seed=0, deterministic=True, single_cls=True, rect=False, cos_lr=False, close_mosaic=0, resume=False, amp=True, overlap_mask=True, mask_ratio=4, dropout=0.1, val=True, split=val, save_json=False, save_hybrid=False, conf=0.5, iou=0.7, max_det=300, half=False, dnn=False, plots=True, source=None, show=False, save_txt=False, save_conf=True, save_crop=False, show_labels=True, show_conf=True, vid_stride=1, line_width=None, visualize=False, augment=True, agnostic_nms=False, classes=None, retina_masks=False, boxes=True, format=torchscript, keras=False, optimize=False, int8=False, dynamic=False, simplify=False, opset=None, workspace=4, nms=False, lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=7.5, cls=0.5, dfl=1.5, pose=12.0, kobj=1.0, label_smoothing=0.0, nbs=64, hsv_h=0.085, hsv_s=0.75, hsv_v=0.75, degrees=7, translate=0.1, scale=0.1, shear=0.0, perspective=0.0, flipud=0, fliplr=0.5, mosaic=False, mixup=False, copy_paste=0.0, cfg=None, v5loader=False, tracker=botsort.yaml, save_dir=trafficsigns_yolo8s_640\train2
Overriding model.yaml nc=80 with nc=1

                   from  n    params  module                                       arguments
  0                  -1  1       928  ultralytics.nn.modules.conv.Conv             [3, 32, 3, 2]
  1                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  2                  -1  1     29056  ultralytics.nn.modules.block.C2f             [64, 64, 1, True]
  3                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  4                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  5                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  6                  -1  2    788480  ultralytics.nn.modules.block.C2f             [256, 256, 2, True]
  7                  -1  1   1180672  ultralytics.nn.modules.conv.Conv             [256, 512, 3, 2]
  8                  -1  1   1838080  ultralytics.nn.modules.block.C2f             [512, 512, 1, True]
  9                  -1  1    656896  ultralytics.nn.modules.block.SPPF            [512, 512, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  1    591360  ultralytics.nn.modules.block.C2f             [768, 256, 1]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 16                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 19                  -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  1   1969152  ultralytics.nn.modules.block.C2f             [768, 512, 1]
 22        [15, 18, 21]  1   2116435  ultralytics.nn.modules.head.Detect           [1, [128, 256, 512]]
YOLOv8s summary: 225 layers, 11135987 parameters, 11135971 gradients

Transferred 349/355 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir trafficsigns_yolo8s_640\train2', view at http://localhost:6006/
AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
AMP: checks passed
optimizer: SGD(lr=0.01) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias
train: Scanning C:\Users\fgarcia24\OneDrive - Schlumberger\Python\fco-parga\mna\mna-navegacion_autonoma\Actividad-4.2\data\YOLO_data\labels\train.cache... 592 images, 0 backgrounds, 0 corrupt: 100%|██████████| 592/592 [00:00<?, ?it/s]
```

```
train: WARNING  C:\Users\fgarcia24\OneDrive - Schlumberger\Python\fco-parga\mna\mna-navegacion_autonoma\Actividad-4.2\data\YOLO_data\images
\train\00340.jpg: 1 duplicate labels removed
val: Scanning C:\Users\fgarcia24\OneDrive - Schlumberger\Python\fco-parga\mna\mna-navegacion_autonoma\Actividad-4.2\data\YOLO_data\labels\t
est.cache... 149 images, 0 backgrounds, 0 corrupt: 100%|██████████| 149/149 [00:00<?, ?it/s]
Plotting labels to trafficsigns_yolo8s_640\train2\labels.jpg...
Image sizes 640 train, 640 val
Using 16 dataloader workers
Logging results to trafficsigns_yolo8s_640\train2
Starting training for 83 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       1/83      7.47G      2.219      14.68      1.271         20        640: 100%|██████████| 19/19 [00:17<00:00,  1.10it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:00<00:00,  3.99it/s]
                   all        149        243       0.87      0.276      0.576      0.326

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       2/83      7.69G      1.717      1.345      1.071         28        640: 100%|██████████| 19/19 [00:05<00:00,  3.74it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:00<00:00,  4.14it/s]
                   all        149        243      0.896      0.354      0.632      0.374

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       3/83      7.53G      1.695      1.195      1.028         23        640: 100%|██████████| 19/19 [00:05<00:00,  3.66it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:00<00:00,  3.36it/s]
                   all        149        243      0.835      0.708      0.803      0.415

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       4/83      7.55G      1.641      1.215      1.037         27        640: 100%|██████████| 19/19 [00:05<00:00,  3.73it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:00<00:00,  3.02it/s]
                   all        149        243      0.818      0.724      0.817      0.378

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       5/83      7.53G      1.688      1.145      1.032         28        640: 100%|██████████| 19/19 [00:05<00:00,  3.72it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:01<00:00,  2.99it/s]
                   all        149        243      0.836      0.758      0.838      0.407

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       6/83      7.55G      1.749      1.162      1.075         23        640: 100%|██████████| 19/19 [00:05<00:00,  3.72it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:00<00:00,  3.53it/s]
                   all        149        243      0.773      0.564      0.682      0.312

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       7/83      7.68G      1.717      1.133      1.041         30        640: 100%|██████████| 19/19 [00:05<00:00,  3.68it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:00<00:00,  3.13it/s]
                   all        149        243      0.655      0.658      0.688      0.289

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       8/83      7.55G      1.703      1.028      1.042         25        640: 100%|██████████| 19/19 [00:05<00:00,  3.67it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:00<00:00,  3.26it/s]
```

```
                   all        149        243      0.851      0.675      0.788      0.352

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       9/83      7.53G      1.732     0.9913      1.051         29        640: 100%|██████████| 19/19 [00:05<00:00,  3.66it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:01<00:00,  2.93it/s]
                   all        149        243      0.725       0.75      0.761      0.356

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      10/83      7.55G      1.686      1.026      1.028         24        640: 100%|██████████| 19/19 [00:05<00:00,  3.64it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:00<00:00,  3.11it/s]
                   all        149        243      0.737      0.668      0.721      0.361

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      11/83      7.52G      1.715     0.9607       1.03         25        640: 100%|██████████| 19/19 [00:05<00:00,  3.65it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:00<00:00,  3.48it/s]
                   all        149        243      0.934      0.638      0.777      0.396

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      12/83      7.55G      1.699     0.9815      1.038         33        640: 100%|██████████| 19/19 [00:05<00:00,  3.40it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:00<00:00,  3.33it/s]
                   all        149        243      0.823      0.671       0.76      0.359

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      13/83      7.52G      1.679     0.9558      1.027         28        640: 100%|██████████| 19/19 [00:05<00:00,  3.40it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:00<00:00,  3.24it/s]
                   all        149        243      0.792      0.601      0.703      0.348

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      14/83      7.71G       1.69     0.9369      1.041         26        640: 100%|██████████| 19/19 [00:05<00:00,  3.30it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:00<00:00,  3.19it/s]
                   all        149        243      0.897      0.679      0.796      0.391

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      15/83      7.52G      1.659     0.8911      1.037         30        640: 100%|██████████| 19/19 [00:05<00:00,  3.26it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:00<00:00,  3.06it/s]
                   all        149        243      0.883      0.745      0.806      0.385

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      16/83      7.72G      1.651      0.901       1.03         25        640: 100%|██████████| 19/19 [00:05<00:00,  3.18it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:01<00:00,  2.36it/s]
                   all        149        243      0.935      0.593      0.763      0.371
Stopping training early as no improvement observed in last 13 epochs. Best results observed at epoch 3, best model saved as best.pt.
To update EarlyStopping(patience=13) pass a new patience value, i.e. `patience=300` or use `patience=0` to disable EarlyStopping.

16 epochs completed in 0.034 hours.
Optimizer stripped from trafficsigns_yolo8s_640\train2\weights\last.pt, 22.5MB
Optimizer stripped from trafficsigns_yolo8s_640\train2\weights\best.pt, 22.5MB
```

```
Validating trafficsigns_yolo8s_640\train2\weights\best.pt...
Ultralytics YOLOv8.0.106  Python-3.9.16 torch-2.0.1+cu117 CUDA:0 (NVIDIA RTX A4500 Laptop GPU, 16384MiB)
YOLOv8s summary (fused): 168 layers, 11125971 parameters, 0 gradients
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:01<00:00,  2.03it/s]
                   all        149        243      0.835      0.708      0.803      0.416
Speed: 0.7ms preprocess, 2.7ms inference, 0.0ms loss, 0.8ms postprocess per image
Results saved to trafficsigns_yolo8s_640\train2
CPU times: total: 2min 15s
Wall time: 8min 49s
```

In [20]:

```
road_img = cv2.imread('trafficsigns_yolo8s_640/train2/val_batch1_pred.jpg')

plt.figure(figsize=(20,11))
plt.imshow(road_img)
plt.axis('off')
plt.show()
```

Detector de senales de trafico integrado

In [21]:
```
classes = {
    0:'Speed limit (20km/h)',
    1:'Speed limit (30km/h)',
    2:'Speed limit (50km/h)',
    3:'Speed limit (60km/h)',
    4:'Speed limit (70km/h)',
```

```
    5:'Speed limit (80km/h)',
    6:'End of speed limit (80km/h)',
    7:'Speed limit (100km/h)',
    8:'Speed limit (120km/h)',
    9:'No passing',
    10:'No passing veh over 3.5 tons',
    11:'Right-of-way at intersection',
    12:'Priority road',
    13:'Yield',
    14:'Stop',
    15:'No vehicles',
    16:'Veh > 3.5 tons prohibited',
    17:'No entry',
    18:'General caution',
    19:'Dangerous curve left',
    20:'Dangerous curve right',
    21:'Double curve',
    22:'Bumpy road',
    23:'Slippery road',
    24:'Road narrows on the right',
    25:'Road work',
    26:'Traffic signals',
    27:'Pedestrians',
    28:'Children crossing',
    29:'Bicycles crossing',
    30:'Beware of ice/snow',
    31:'Wild animals crossing',
    32:'End speed + passing limits',
    33:'Turn right ahead',
    34:'Turn left ahead',
    35:'Ahead only',
    36:'Go straight or right',
    37:'Go straight or left',
    38:'Keep right',
    39:'Keep left',
    40:'Roundabout mandatory',
    41:'End of no passing',
    42:'End no passing veh > 3.5 tons'
}
```

In [22]:

```
from tqdm.auto import tqdm
from PIL import Image
import tensorflow as tf

class TrafficSignDetector:
    def __init__(self, model_detec, model_classify):
        self.model_detec = model_detec
```

```
        self.model_classify = model_classify

    def process_frame(self, img):
        # Definir los parámetros para deteccin de señales de trafico
        passed_frame = Image.fromarray(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        #Deteccion de se;ales de transito con YOLO
        detected_trafficsigns = self.model_detec.predict(source=passed_frame, save=False, conf=0.1, verbose=False)
        boxes = detected_trafficsigns[0].boxes.xyxy.cpu().data.numpy().astype(int)

        # Definir una función para redimensionar y normalizar una imagen
        def preprocess_image(image):
            image = tf.image.resize(image, (28, 28))
            image = tf.cast(image, tf.float32)
            image = image/255
            return image

        #Función para crear una caja cuadrada en la señal detectadda
        def get_squered_box(box):
            extend = 1
            w = box[2]-box[0]
            h = box[3]-box[1]

            long = (max(w,h)/2)+extend

            center = (np.mean([box[2],box[0]]).astype(int),np.mean([box[3],box[1]]).astype(int))

            square_box = np.array([center[0]-long, center[1]-long,
                                   center[0]+long, center[1]+long]).astype(int)
            return square_box, long

        # Clasificación de cada señal detectada dentro el frame
        if boxes.shape[0] >0:
            for sign in boxes:
                square_box, long = get_squered_box(sign)
                cropped_sign = img[square_box[1]:square_box[3],square_box[0]:square_box[2]]

                # Convertir la región de interés en un tensor de TensorFlow y normalizar
                roi_tensor = preprocess_image(np.array(cropped_sign))
                roi_tensor = tf.expand_dims(roi_tensor, axis=0)

                # Predecir qué tipo de señal de tránsito fue eocntrada en la región de interés
                self.prediction = self.model_classify.predict(roi_tensor, verbose=0)
                self.trafic_class = classes.get(np.argmax(self.prediction))

                # Si se detecta una señal de tránsito, dibujar un rectángulo alrededor de la región de interés
                if tf.reduce_any(tf.greater_equal(self.prediction, 0.9)):
                    cv2.rectangle(img, (sign[0], sign[1]), (sign[2], sign[3]), (0, 255, 0), 2)
                    cv2.putText(img, self.trafic_class, (sign[0], sign[1]-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0,255,0), 2)
```

```
                    #print('Traficsignal')


        return img

    def process_video(self, input_video_path, output_video_path, start_at=3600, fracction_to_process=0.1):
        frame_set_no=start_at
        cap = cv2.VideoCapture(input_video_path)
        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_set_no)

        width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = int(cap.get(cv2.CAP_PROP_FPS))
        total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

        # Calcular el número de fotogramas a procesar (el 10% de los fotogramas totales)
        num_frames_to_process = int(total_frames * fracction_to_process)

        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out = cv2.VideoWriter(output_video_path, fourcc, fps, (width, height))

        pbar = tqdm(total=num_frames_to_process, ncols=80, bar_format='{l_bar}{bar}| {n_fmt}/{total_fmt}',
                    position=0, leave=True)

        frame_count = 0

        while cap.isOpened():
            ret, frame = cap.read()
            if ret:
                try:
                    # Procesar el fotograma con el detector de señales de tráfico
                    result_frame = self.process_frame(frame)
                    # Escribir el fotograma procesado en el archivo de salida
                    out.write(result_frame)
                    #cv2.imshow('Processed Frame', result_frame)

                    pbar.update(1)
                    frame_count += 1

                    if frame_count >= num_frames_to_process:
                        break

                    #if cv2.waitKey(1) & 0xFF == ord('q'):
                    #    break
                except:
                    pass
            else:
                break
```

```python
        pbar.close()

        cap.release()
        out.release()
        #cv2.destroyAllWindows()
```

In [23]:

```python
from ultralytics import YOLO

input_video_path = 'dash_cam.mp4'
output_video_path = 'dash_cams_cnn-Keras.mp4'

model_detec = YOLO('trafficsigns_yolo8s_640/train/weights/best.pt')
model_classify = tf.keras.models.load_model("best_model.h5")
```

In [24]:

```python
%%time
# Crear una instancia de PedestrianDetector con svc_model como el modelo a usar
traffic_sign_detector = TrafficSignDetector(model_detec, model_classify)

# Procesar el archivo de vídeo de entrada y guardar el archivo de vídeo de salida usando el método process_video
traffic_sign_detector.process_video(input_video_path, output_video_path, start_at=3600, fracction_to_process=0.2)
```

```
  0%|                                                                   | 0/3990
CPU times: total: 7min 29s
Wall time: 6min 24s
```

Resultado de video procesado

El resultado procesado puede encontrarse en el link de youtube:

> https://youtu.be/ErCkWxos-EI