# Medición de flujo vehicular mediante una CNN

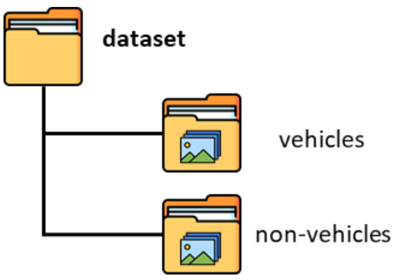
## Bibliotecas generales

Se importan bibliotecas generales que serán utilizadas durante todo el proyecto.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

## Lectura de datos

Para la lectura de datos, se obtienen las rutas de acceso a cada una de las imágenes siguiendo la siguiente estrucutra:




Posteriormente se crea un dataframe con la ruta y su respectiva etiqueta.

In [ ]:
```
import os

classLabel = ""
imagePath = []
labels = []
categories = {"non-vehicles": 0, "vehicles": 1}

# Escanea los directores dentro de la carpeta dataset
for dirname, _, filenames in os.walk("dataset/"):
    print(dirname)
    if (dirname.split("/")[-1] != "dataset"):
        classLabel = dirname.split("/")[-1]

    for filename in filenames:
        imagePath.append(dirname + "/" + filename)
        labels.append(categories[classLabel])
```

```
dataset/
dataset/non-vehicles
dataset/vehicles
```

## Dataframe

Se obtiene un dataframe con todas las rutas de acceso a las imágenes y con su respectiva etiqueta.

In [ ]:
```
dataset = pd.DataFrame({"path": imagePath, "label": labels})
dataset
```

Out[ ]:

| | path | label |
|---|---|---|
| 0 | dataset/non-vehicles/extra1.png | 0 |
| 1 | dataset/non-vehicles/extra10.png | 0 |
| 2 | dataset/non-vehicles/extra100.png | 0 |
| 3 | dataset/non-vehicles/extra1000.png | 0 |
| 4 | dataset/non-vehicles/extra1001.png | 0 |
| ... | ... | ... |
| 17755 | dataset/vehicles/right (95).png | 1 |
| 17756 | dataset/vehicles/right (96).png | 1 |
| 17757 | dataset/vehicles/right (97).png | 1 |
| 17758 | dataset/vehicles/right (98).png | 1 |
| 17759 | dataset/vehicles/right (99).png | 1 |

17760 rows × 2 columns

### Remuestreo de datos

Se ordena aleatoriamente las imágenes para evitar sesgo durante la etapa de entrenamiento.

In [ ]:
```
shuffle_dataset = dataset.sample(frac=1).reset_index()
shuffle_dataset
```

Out[ ]:

| | index | path | label |
|---|---|---|---|
| 0 | 10687 | dataset/vehicles/2549.png | 1 |
| 1 | 1901 | dataset/non-vehicles/extra3100.png | 0 |
| 2 | 1531 | dataset/non-vehicles/extra2684.png | 0 |
| 3 | 1206 | dataset/non-vehicles/extra238.png | 0 |
| 4 | 12610 | dataset/vehicles/428.png | 1 |
| ... | ... | ... | ... |
| 17755 | 1806 | dataset/non-vehicles/extra2948.png | 0 |
| 17756 | 13002 | dataset/vehicles/4632.png | 1 |
| 17757 | 8499 | dataset/non-vehicles/image577.png | 0 |
| 17758 | 16626 | dataset/vehicles/left (872).png | 1 |
| 17759 | 1953 | dataset/non-vehicles/extra3148.png | 0 |

17760 rows × 3 columns

## Preprocesado de datos

Para el procesado se realizarán las siguientes transformaciones:

- Se redimensionarán las imágenes a un tamaño fijo de (32, 32, 3).
- Los valores de las imágenes se normalizarán en una escala de 0 a 1.

In [ ]:
```
from skimage.transform import resize

images = []
labels = shuffle_dataset["label"]

for path in shuffle_dataset["path"]:
    image = plt.imread(path)
    resizedImage = resize(image, (32,32,3))
    images.append(resizedImage)

images = np.array(images)
images = images/255
```

In [ ]:
```
images.shape
```

Out[ ]:
```
(17760, 32, 32, 3)
```

## Limpeado de memoria

Se realiza un limpiado de memoria para liberar memoria cache.

In [ ]:
```
import gc

del dataset
gc.collect()
```

Out[ ]:
```
16
```

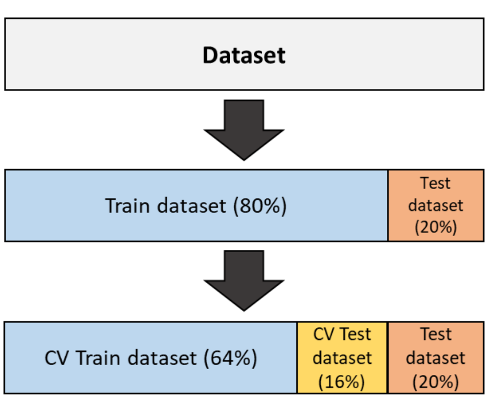
## Imágenes procesadas

Se imprime un conjunto de 9 imágenes ya procesadas con su etiqueta.

In [ ]:
```
plt.figure(figsize=(9,10))
for i in range(9):
    plt.subplot(3,3,i+1)
    #print(shuffle_dataset["label"][i])
    im = plt.imread(shuffle_dataset["path"][i])
    plt.title("Label "+str(shuffle_dataset["label"][i]))
    plt.imshow(im)
plt.show()
```


### Preparación de datos de entrada

Para la fase de entrenamiento, el set de datos se partió en tres partes como se muestra en la siguiente figura.




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

# Aquí se parte el set de datos, siendo el train-set el 80% y el test-set 20%.
X_train,X_test,y_train,y_test = train_test_split(images,labels.values,test_size=0.2)
```

In [ ]:
```
from tensorflow.keras.utils import to_categorical

y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)
y_train_one_hot
```

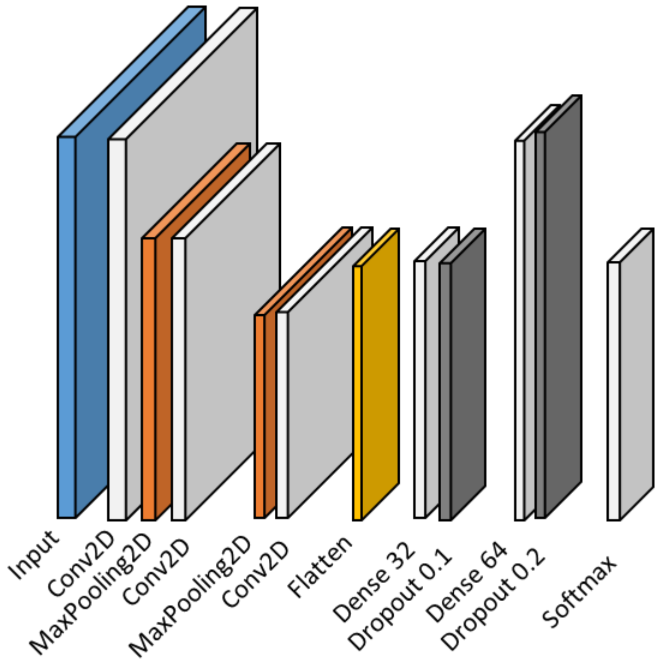
Out[ ]:
```
array([[1., 0.],
       [0., 1.],
       [1., 0.],
       ...,
       [1., 0.],
       [1., 0.],
       [1., 0.]], dtype=float32)
```

In [ ]:
```
X_train.shape, y_train.shape
```

Out[ ]:
```
((14208, 32, 32, 3), (14208,))
```

## CNN

La siguiente parte consiste en la creación de la Red Neural Convolucional, para este proyecto se eligirá la siguiente arquitectura:




### Bibliotecas

In [ ]:
```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense,Flatten,Conv2D,MaxPooling2D,Dropout
```

### Construcción del Modelo

El modelo propuesto anteriormente se construirá en el siguiente bloque.

In [ ]:
```
model = Sequential()

# Bloque 1
model.add(Conv2D(32,(5,5),activation="relu",input_shape=(32,32,3)))
model.add(MaxPooling2D(pool_size=(2,2)))

# Bloque 2
model.add(Conv2D(32,(5,5),activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

# Capa flatten
model.add(Flatten())

# Block 3
model.add(Dense(32,activation="relu"))
model.add(Dropout(0.1))
model.add(Dense(64,activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(2,activation="softmax"))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_2 (Conv2D)           (None, 28, 28, 32)        2432      
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 14, 14, 32)       0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (None, 10, 10, 32)        25632     
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 5, 5, 32)         0         
 2D)                                                             
                                                                 
 flatten_1 (Flatten)         (None, 800)               0         
                                                                 
 dense_3 (Dense)             (None, 32)                25632     
                                                                 
 dropout_2 (Dropout)         (None, 32)                0         
                                                                 
 dense_4 (Dense)             (None, 64)                2112      
                                                                 
 dropout_3 (Dropout)         (None, 64)                0         
                                                                 
 dense_5 (Dense)             (None, 2)                 130       
                                                                 
=================================================================
Total params: 55,938
Trainable params: 55,938
Non-trainable params: 0
_________________________________________________________________
```

### Compilado del Modelo

In [ ]:
```
model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

### Entrenamiento del Modelo

In [ ]:
```
hist = model.fit(X_train,y_train_one_hot,
                 epochs=50,
                 validation_split=0.2,
                 use_multiprocessing=True)
```

```
Epoch 1/50
356/356 [==============================] - 8s 21ms/step - loss: 0.6784 - accuracy: 0.5525 - val_loss: 0.5696 - val_accuracy: 0.7287
Epoch 2/50
356/356 [==============================] - 7s 21ms/step - loss: 0.4344 - accuracy: 0.8080 - val_loss: 0.3277 - val_accuracy: 0.8466
Epoch 3/50
356/356 [==============================] - 7s 20ms/step - loss: 0.3085 - accuracy: 0.8692 - val_loss: 0.2675 - val_accuracy: 0.8874
Epoch 4/50
356/356 [==============================] - 7s 20ms/step - loss: 0.2755 - accuracy: 0.8863 - val_loss: 0.2464 - val_accuracy: 0.8941
Epoch 5/50
356/356 [==============================] - 7s 20ms/step - loss: 0.2456 - accuracy: 0.8982 - val_loss: 0.2253 - val_accuracy: 0.9085
Epoch 6/50
356/356 [==============================] - 7s 20ms/step - loss: 0.2293 - accuracy: 0.9074 - val_loss: 0.2155 - val_accuracy: 0.9141
Epoch 7/50
356/356 [==============================] - 7s 20ms/step - loss: 0.2165 - accuracy: 0.9116 - val_loss: 0.2163 - val_accuracy: 0.9025
Epoch 8/50
356/356 [==============================] - 7s 20ms/step - loss: 0.2019 - accuracy: 0.9184 - val_loss: 0.1734 - val_accuracy: 0.9314
Epoch 9/50
356/356 [==============================] - 7s 20ms/step - loss: 0.1869 - accuracy: 0.9279 - val_loss: 0.1663 - val_accuracy: 0.9356
Epoch 10/50
356/356 [==============================] - 7s 20ms/step - loss: 0.1803 - accuracy: 0.9294 - val_loss: 0.1892 - val_accuracy: 0.9201
Epoch 11/50
356/356 [==============================] - 7s 21ms/step - loss: 0.1707 - accuracy: 0.9344 - val_loss: 0.1516 - val_accuracy: 0.9419
Epoch 12/50
356/356 [==============================] - 7s 20ms/step - loss: 0.1587 - accuracy: 0.9367 - val_loss: 0.1449 - val_accuracy: 0.9423
Epoch 13/50
356/356 [==============================] - 7s 21ms/step - loss: 0.1455 - accuracy: 0.9448 - val_loss: 0.1337 - val_accuracy: 0.9483
Epoch 14/50
356/356 [==============================] - 7s 20ms/step - loss: 0.1448 - accuracy: 0.9447 - val_loss: 0.1225 - val_accuracy: 0.9543
Epoch 15/50
356/356 [==============================] - 7s 20ms/step - loss: 0.1333 - accuracy: 0.9470 - val_loss: 0.1265 - val_accuracy: 0.9497
Epoch 16/50
356/356 [==============================] - 7s 20ms/step - loss: 0.1336 - accuracy: 0.9474 - val_loss: 0.1274 - val_accuracy: 0.9479
Epoch 17/50
356/356 [==============================] - 7s 20ms/step - loss: 0.1266 - accuracy: 0.9516 - val_loss: 0.1078 - val_accuracy: 0.9613
Epoch 18/50
356/356 [==============================] - 7s 20ms/step - loss: 0.1222 - accuracy: 0.9535 - val_loss: 0.1198 - val_accuracy: 0.9553
Epoch 19/50
356/356 [==============================] - 7s 20ms/step - loss: 0.1182 - accuracy: 0.9550 - val_loss: 0.1119 - val_accuracy: 0.9592
Epoch 20/50
356/356 [==============================] - 7s 20ms/step - loss: 0.1090 - accuracy: 0.9577 - val_loss: 0.0994 - val_accuracy: 0.9631
Epoch 21/50
356/356 [==============================] - 7s 20ms/step - loss: 0.1117 - accuracy: 0.9569 - val_loss: 0.0989 - val_accuracy: 0.9659
Epoch 22/50
356/356 [==============================] - 7s 20ms/step - loss: 0.1060 - accuracy: 0.9604 - val_loss: 0.1143 - val_accuracy: 0.9546
Epoch 23/50
356/356 [==============================] - 7s 20ms/step - loss: 0.1080 - accuracy: 0.9602 - val_loss: 0.1125 - val_accuracy: 0.9574
Epoch 24/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0985 - accuracy: 0.9628 - val_loss: 0.1052 - val_accuracy: 0.9595
Epoch 25/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0952 - accuracy: 0.9639 - val_loss: 0.0942 - val_accuracy: 0.9655
Epoch 26/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0960 - accuracy: 0.9632 - val_loss: 0.0893 - val_accuracy: 0.9659
Epoch 27/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0880 - accuracy: 0.9679 - val_loss: 0.0916 - val_accuracy: 0.9673
Epoch 28/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0908 - accuracy: 0.9663 - val_loss: 0.1212 - val_accuracy: 0.9581
Epoch 29/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0863 - accuracy: 0.9675 - val_loss: 0.0884 - val_accuracy: 0.9690
Epoch 30/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0846 - accuracy: 0.9684 - val_loss: 0.0872 - val_accuracy: 0.9687
Epoch 31/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0831 - accuracy: 0.9679 - val_loss: 0.1022 - val_accuracy: 0.9638
Epoch 32/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0798 - accuracy: 0.9723 - val_loss: 0.0894 - val_accuracy: 0.9704
Epoch 33/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0778 - accuracy: 0.9716 - val_loss: 0.0768 - val_accuracy: 0.9750
Epoch 34/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0750 - accuracy: 0.9718 - val_loss: 0.0814 - val_accuracy: 0.9740
Epoch 35/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0738 - accuracy: 0.9729 - val_loss: 0.0854 - val_accuracy: 0.9680
Epoch 36/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0731 - accuracy: 0.9722 - val_loss: 0.0897 - val_accuracy: 0.9690
Epoch 37/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0682 - accuracy: 0.9742 - val_loss: 0.0786 - val_accuracy: 0.9743
Epoch 38/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0735 - accuracy: 0.9715 - val_loss: 0.0810 - val_accuracy: 0.9729
Epoch 39/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0640 - accuracy: 0.9768 - val_loss: 0.0765 - val_accuracy: 0.9750
Epoch 40/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0687 - accuracy: 0.9744 - val_loss: 0.0809 - val_accuracy: 0.9715
Epoch 41/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0678 - accuracy: 0.9740 - val_loss: 0.0718 - val_accuracy: 0.9757
Epoch 42/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0642 - accuracy: 0.9755 - val_loss: 0.0837 - val_accuracy: 0.9726
Epoch 43/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0596 - accuracy: 0.9797 - val_loss: 0.0737 - val_accuracy: 0.9743
Epoch 44/50
356/356 [==============================] - 7s 21ms/step - loss: 0.0636 - accuracy: 0.9760 - val_loss: 0.0781 - val_accuracy: 0.9736
Epoch 45/50
356/356 [==============================] - 7s 21ms/step - loss: 0.0603 - accuracy: 0.9771 - val_loss: 0.0745 - val_accuracy: 0.9740
Epoch 46/50
356/356 [==============================] - 7s 20ms/step - loss: 0.0608 - accuracy: 0.9765 - val_loss: 0.1086 - val_accuracy: 0.9616
Epoch 47/50
356/356 [==============================] - 8s 21ms/step - loss: 0.0591 - accuracy: 0.9777 - val_loss: 0.0747 - val_accuracy: 0.9729
Epoch 48/50
356/356 [==============================] - 8s 22ms/step - loss: 0.0575 - accuracy: 0.9796 - val_loss: 0.0692 - val_accuracy: 0.9768
Epoch 49/50
356/356 [==============================] - 8s 21ms/step - loss: 0.0572 - accuracy: 0.9772 - val_loss: 0.0750 - val_accuracy: 0.9761
Epoch 50/50
356/356 [==============================] - 7s 21ms/step - loss: 0.0569 - accuracy: 0.9787 - val_loss: 0.0773 - val_accuracy: 0.9757
```

### Evaluación del modelo

#### Accuracy

Para evaluar el desempeño del modelo se usa la métrica accuracy.

In [ ]:
```
print("Accuracy = ",np.round(model.evaluate(X_test,y_test_one_hot,verbose=0)[1],2)*100,"%")
```

```
Accuracy =  97.0 %
```

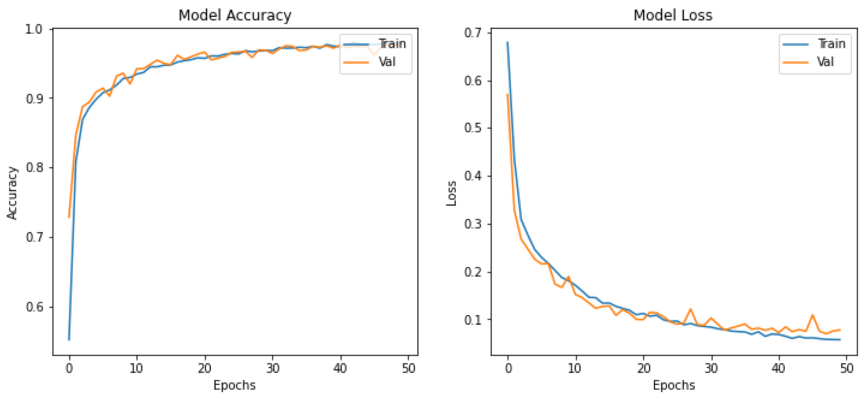
#### Curvas de Aprendizaje

En esta sección se muestran las curvas de aprendizaje del modelo entrenado.

In [ ]:
```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(hist.history["accuracy"])
plt.plot(hist.history["val_accuracy"])
```

```
plt.title("Model Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Epochs")
plt.legend(["Train","Val"],loc="upper right")

plt.subplot(1,2,2)
plt.plot(hist.history["loss"])
plt.plot(hist.history["val_loss"])
plt.title("Model Loss")
plt.ylabel("Loss")
plt.xlabel("Epochs")
plt.legend(["Train", "Val"], loc="upper right")
plt.show()
```




## Exportar modelo

In [ ]:
```
model.save("modelo_car.h5")
```