Introducción a Tensor Flow

In [1]:

```
#import tensorflow as tf
import tensorflow.compat.v1 as tf
```

In [2]:

```
tf.disable_v2_behavior() #Deshabilitammos el comportamiento de la última vers de Tensor Flow
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\compat\v
2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_scope) is de
precated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

In [3]:

```
x1 = tf.constant([1,2,3,4,5])
x2= tf.constant([6,7,8,9,10])
```

In [4]:

```
res = tf.multiply(x1,x2)
print(res)
```

```
Tensor("Mul:0", shape=(5,), dtype=int32)
```

In [5]:

```
sess = tf.Session()
print(sess.run(res))
sess.close()
```

```
[ 6 14 24 36 50]
```

In [6]:

```
with tf.Session() as sess:
    output = sess.run(res)
    print(output)
```

```
[ 6 14 24 36 50]
```

In [7]:

```
config = tf.ConfigProto(log_device_placement = True)
config = tf.ConfigProto(allow_soft_placement = True) #Ambas instrucciones permiten trabajar c
```

Aprendizaje neuronal de las señales de tráfico

In [8]:

```
import os
from skimage.io import imread #librería dedicada para tratamiento de imágenes
```

In [9]:

```
def load_ml_data(data_directory):
    """Cargar los directorios de una base de datos distribuída"""
    dirs = [d for d in os.listdir(data_directory)
           if os.path.isdir(os.path.join(data_directory, d))]
    labels = []
    images = []
    for d in dirs:
        label_dir = os.path.join(data_directory,d) # Empalma el subdirectorio con el path ppa
        file_names = [os.path.join(label_dir, f)
                      for f in os.listdir(label_dir)
                      if f.endswith(".ppm")] # f es el nombre del fichero
        for f in file_names:
            images.append(imread(f))
```

```python
                labels.append(int(d))

    return images, labels

    print(dirs)
```

In [10]:
```python
main_dir = "../../Data-Sets/datasets/belgian/"
train_data_dir = os.path.join(main_dir, "Training")
test_data_dir = os.path.join(main_dir, "Testing")
```

In [11]:
```python
images, labels = load_ml_data(train_data_dir)
```

In [12]:
```python
import numpy as np
```

In [13]:
```python
images = np.array(images, dtype=object)
```

In [14]:
```python
len(images) #Tenemos unas 4575 imágenes
```

Out[14]:
```
4575
```

In [15]:
```python
labels = np.array(labels, dtype=object)
```

In [16]:
```python
images.ndim #vemos las dimensiones del objeto
```

Out[16]:
```
1
```

In [17]:
```python
images.size #Vemos el tamaño o cantidad de imágenes
```

Out[17]:
```
4575
```

In [18]:
```python
images[0] #primera foto
#Tres canales, son Rojo, Verde y Azul que corresponde a cada columna.
#Estar atento a esta configuración.
```

Out[18]:
```
array([[[210, 249, 232],
        [204, 249, 208],
        [197, 198, 155],
        ...,
        [ 51,  60,  40],
        [ 54,  64,  44],
        [ 57,  66,  46]],

       [[209, 250, 236],
        [212, 255, 217],
        [200, 196, 156],
        ...,
        [ 49,  57,  38],
        [ 51,  59,  41],
        [ 53,  60,  42]],

       [[203, 246, 236],
        [207, 246, 213],
        [202, 192, 156],
        ...,
```

```
         [ 47,  53,  35],
         [ 48,  54,  36],
         [ 48,  55,  37]],

        ...,

        [[  2,  22,  25],
         [ 26,  56,  77],
         [ 71, 140, 159],
         ...,
         [ 84,  77,  50],
         [ 68,  66,  41],
         [ 56,  64,  44]],

        [[  0,  22,  32],
         [ 30,  75, 106],
         [ 87, 176, 198],
         ...,
         [ 86,  80,  52],
         [ 68,  66,  41],
         [ 55,  63,  42]],

        [[  0,  32,  50],
         [ 42, 101, 135],
         [121, 217, 239],
         ...,
         [ 87,  80,  52],
         [ 70,  68,  43],
         [ 58,  66,  46]]], dtype=uint8)
```

In [19]:

```
labels.ndim
```

Out[19]:

```
1
```

In [20]:

```
labels.size
```

Out[20]:

```
4575
```

In [21]:

```
len(set(labels)) #Sólo contabilizará las etiquetas diferentes, es como unique
```

Out[21]:

```
62
```

In [22]:

```
images.flags
```

Out[22]:

```
  C_CONTIGUOUS : True
  F_CONTIGUOUS : True
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
  UPDATEIFCOPY : False
```

In [23]:

```
images.itemsize #8-bits
```

Out[23]:

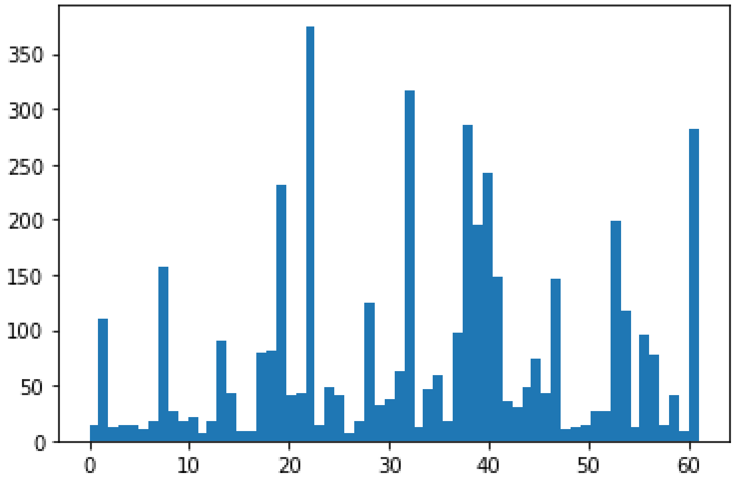
```
8
```

In [24]:

```
images.nbytes #Consultamos la cantidad de bytes utilizados por la ram
```

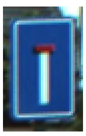
Out[24]:

```
36600
```

In [25]:
```python
images.nbytes/images.itemsize #Esto es la cantidad de bits que estoy utilizando
```
Out[25]:
```
4575.0
```
In [26]:
```python
import matplotlib.pyplot as plt
```
In [27]:
```python
plt.hist(labels, len(set(labels)))
plt.show()
```


No todo los tipos de imágenes están representadas igualmente en el dataset.

In [28]:
```python
import random
```
In [29]:
```python
rand_signs = random.sample(range(0, len(labels)), 6)
rand_signs
```
Out[29]:
```
[4312, 4041, 3320, 3269, 1139, 1719]
```
In [30]:
```python
for i in range(len(rand_signs)):
    temp_im = images[rand_signs[i]]
    plt.subplot(1,6,i+1)
    plt.axis("off") #desactivamos los axis (métricas en cada imágen)
    plt.imshow(images[rand_signs[i]])
    plt.subplots_adjust(wspace=0.5)
    plt.show()
    print("Forma:{0}, min:{1}, max{2}".format(temp_im.shape,
                                              temp_im.min(),
                                              temp_im.max())) #esto se acerca más a django
```


```
Forma:(57, 56, 3), min:12, max201
```


```
Forma:(99, 63, 3), min:0, max255
```

```
Forma:(148, 103, 3), min:3, max255
```

```
Forma:(139, 140, 3), min:11, max255
```

```
Forma:(109, 104, 3), min:18, max255
```

```
Forma:(172, 166, 3), min:0, max255
```

In [31]:

```
unique_labels = set(labels)
plt.figure(figsize=(16,16))
i = 1
for label in unique_labels:
    temp_im = images[list(labels).index(label)] #devuelve la posición que devuelve la etiquet
    plt.subplot(8,8,i) #matriz de 8 x 8 porque son 61 labels independientes
    plt.axis("off")
    plt.title("Clase {0} ({1})".format(label, list(labels).count(label)))
    i += 1
    plt.imshow(temp_im)

plt.show()
```

In [32]:

```
type(labels)
```

Out[32]:

```
numpy.ndarray
```

Modelo de Red Neuronal con TensorFlow

- Las imágenes no todas son del mismo tamaño.
- Hay 62 clases de imágenes (desde la 0 hasta la 61)
- La distribucipon de señales de tráfico no es uniforme (algunas salen más veces que otras).

A la hora de tratar con las formas de las imágenes se recomienda siempre pasar a **escalas de grises**, pues la iluminación de las imágenes puede influir mucho a la hora de determinar la forma, entonces para la red neuronal se le hace muchísimo más fácil determinar formas antes que colores cuando la imágen se encuentra en escala de grises y **reescalar** es también súper importante.

In [33]:

```
from skimage import transform
```

In [34]:

```
w=99999
```

```python
h=9999
for image in images:
    if image.shape[0] < h:
        w = image.shape[0]
    if image.shape[1] < w:
        h = image.shape[1]
print("Tamaño mínimo : {0}x{1}".format(h,w))
```

```
Tamaño mínimo : 21x22
```

In [35]:

```python
images30 = [transform.resize(image, (50,50)) for image in images]
```

In [36]:

```python
images30[0]#cada imágen es un conjunto de datos de 300 elementos
```

Out[36]:

```
array([[[0.80065569, 0.90670588, 0.78211765],
        [0.62728784, 0.47709804, 0.32407216],
        [0.54145098, 0.43150588, 0.30737255],
        ...,
        [0.19108235, 0.2080549 , 0.13746667],
        [0.18160157, 0.20494745, 0.13427451],
        [0.19317647, 0.22454902, 0.15360784]],

       [[0.7786149 , 0.85766118, 0.76253647],
        [0.57839059, 0.41433255, 0.27329255],
        [0.46836078, 0.35687843, 0.24930196],
        ...,
        [0.18717647, 0.22255686, 0.14741176],
        [0.18611765, 0.21792   , 0.14801882],
        [0.18588863, 0.20678431, 0.14011765]],

       [[0.78932549, 0.84336471, 0.73848627],
        [0.55738824, 0.40112157, 0.25292549],
        [0.46529412, 0.30623529, 0.20713725],
        ...,
        [0.18894118, 0.23078431, 0.15364706],
        [0.19701961, 0.23301176, 0.16130196],
        [0.2       , 0.22568627, 0.16064314]],

       ...,

       [[0.10068235, 0.10090196, 0.04876078],
        [0.05360784, 0.14072941, 0.11523137],
        [0.25105882, 0.32886275, 0.28709804],
        ...,
        [0.10776471, 0.35478431, 0.55529412],
        [0.36592157, 0.50463529, 0.45640784],
        [0.29654902, 0.2914902 , 0.21368627]],

       [[0.08189647, 0.1356298 , 0.14284549],
        [0.39479529, 0.56247059, 0.58      ],
        [0.84752157, 0.8957098 , 0.90178824],
        ...,
        [0.09831373, 0.3551451 , 0.55478431],
        [0.36956863, 0.50017569, 0.45252863],
        [0.27941176, 0.2732549 , 0.18526118]],

       [[0.15620706, 0.35450353, 0.43731608],
        [0.6940251 , 0.91894588, 0.93337725],
        [0.62929412, 0.62858039, 0.70258039],
        ...,
        [0.09882353, 0.35415686, 0.55543529],
        [0.3594902 , 0.49497412, 0.44658824],
        [0.27262745, 0.2732549 , 0.17552941]]])
```

In [37]:
```python
rand_signs = random.sample(range(0, len(labels)), 6)
for i in range(len(rand_signs)):
    temp_im = images30[rand_signs[i]]
    plt.subplot(1,6,i+1)
    plt.axis("off") #desactivamos los axis (métricas en cada imágen)
    plt.imshow(images30[rand_signs[i]])
    plt.subplots_adjust(wspace=0.5)
    plt.show()
    print("Forma:{0}, min:{1}, max{2}".format(temp_im.shape,
                                              temp_im.min(),
                                              temp_im.max())) #esto se acerca más a django
```

```
Forma:(50, 50, 3), min:0.016645490196078937, max1.0
```

```
Forma:(50, 50, 3), min:0.05339607843137306, max0.9911152941176473
```

```
Forma:(50, 50, 3), min:0.068538431372548, max1.0
```

```
Forma:(50, 50, 3), min:0.028031372549019464, max0.9554470588235283
```


```
Forma:(50, 50, 3), min:0.09517647058823553, max0.9990313725490196
```


```
Forma:(50, 50, 3), min:0.07945098039215663, max0.3245803921568625
```

Las imágenes están normalizadas, pues el mínimo valor de color comienza en 0 y finaliza en 1

In [38]:
```python
from skimage.color import rgb2gray
```

In [39]:
```python
images30 = np.array(images30)
```

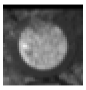
In [40]:
```python
images30 = rgb2gray(images30)
```

In [41]:
```python
rand_signs = random.sample(range(0, len(labels)), 6)
for i in range(len(rand_signs)):
    temp_im = images30[rand_signs[i]]
    plt.subplot(1,6,i+1)
    plt.axis("off") #desactivamos los axis (métricas en cada imágen)
    plt.imshow(temp_im, cmap="gray") #Tenemos que colocar el cmap, porque sino no funciona cc
    plt.subplots_adjust(wspace=0.5)
    plt.show()
    print("Forma:{0}, min:{1}, max{2}".format(temp_im.shape,
                                              temp_im.min(),
                                              temp_im.max())) #esto se acerca más a django
```

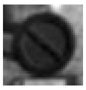
```
Forma:(50, 50), min:0.09511853733333335, max0.46479749984313734
```

```
Forma:(50, 50), min:0.028825995098039205, max0.9929980392156863
```

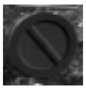
```
Forma:(50, 50), min:0.03772658039215684, max0.9997889873333334
```



```
Forma:(50, 50), min:0.0, max0.8483955732549021
```



```
Forma:(50, 50), min:0.04320876392156835, max0.42123836078431387
```



```
Forma:(50, 50), min:0.04895110023529336, max0.4578013568627425
```

In [42]:

```
## ADAPTACIÓN A LA VERSIÓN ANTIGUA

tf.compat.v1.disable_eager_execution()

#1) Inicializamos los placeholders.
x = tf.compat.v1.placeholder(dtype = tf.float32, shape = [None, 50,50])
y = tf.compat.v1.placeholder(dtype = tf.int32, shape = [None])

#2) Aplanamos la matriz de 30 x 30 a una lista directa
images_flat = tf.compat.v1.layers.flatten(x)

#3) Creamos la red neuronal
logits = tf.compat.v1.layers.dense(images_flat, 62, tf.nn.relu)

#4) Establecemos la función de pérdida
loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(labels = y, logits = log
```

```
C:\Users\Kevin\AppData\Local\Temp/ipykernel_12932/143546519.py:10: UserWarning: `tf.layers.fl
atten` is deprecated and will be removed in a future version. Please use `tf.keras.layers.Fla
tten` instead.
  images_flat = tf.compat.v1.layers.flatten(x)
C:\Users\Kevin\AppData\Local\Temp/ipykernel_12932/143546519.py:13: UserWarning: `tf.layers.de
nse` is deprecated and will be removed in a future version. Please use `tf.keras.layers.Dense
` instead.
  logits = tf.compat.v1.layers.dense(images_flat, 62, tf.nn.relu)
```

In [43]:

```
#5) Creamos la red neuronal
train_opt = tf.compat.v1.train.AdamOptimizer(learning_rate = 0.001).minimize(loss)

final_pred = tf.argmax(logits,1)

accuracy = tf.reduce_mean(tf.cast(final_pred, tf.float32))
```

In [44]:

```
images_flat
```

Out[44]: <tf.Tensor 'flatten/Reshape:0' shape=(?, 2500) dtype=float32>

In [45]:
```
final_pred
```

Out[45]: <tf.Tensor 'ArgMax:0' shape=(?,) dtype=int64>

In [46]:
```
tf.set_random_seed(1234)

sess = tf.Session()

sess.run(tf.global_variables_initializer())

for i in range(301):
    _, accuracy_val = sess.run([train_opt, accuracy],
                              feed_dict={
                                  x:images30,
                                  y:list(labels)
                              })
    _, loss_val = sess.run([train_opt, loss],
                              feed_dict={
                                  x:images30,
                                  y:list(labels)
                              })

    if i%100 == 0:
        print("EPOCH", i)
        print("Accuracy: ", accuracy_val)
        print("Loss", loss_val)
    #print("Fin del Epoch ", i)
```

```
EPOCH 0
Accuracy:  13.029509
Loss 3.844643
EPOCH 100
Accuracy:  34.300545
Loss 1.8875837
EPOCH 200
Accuracy:  34.093334
Loss 1.7296242
EPOCH 300
Accuracy:  33.240875
Loss 1.6545857
```

Evaluación de la red neuronal

In [47]:
```
sample_idx = random.sample(range(len(images30)), 40) #tomamos 16 imágenes aleatorias
sample_images = [images30[i] for i in sample_idx] #extraemos las imágenes
sample_labels = [labels[i] for i in sample_idx] #extraemos los labels de las imágenes.
```

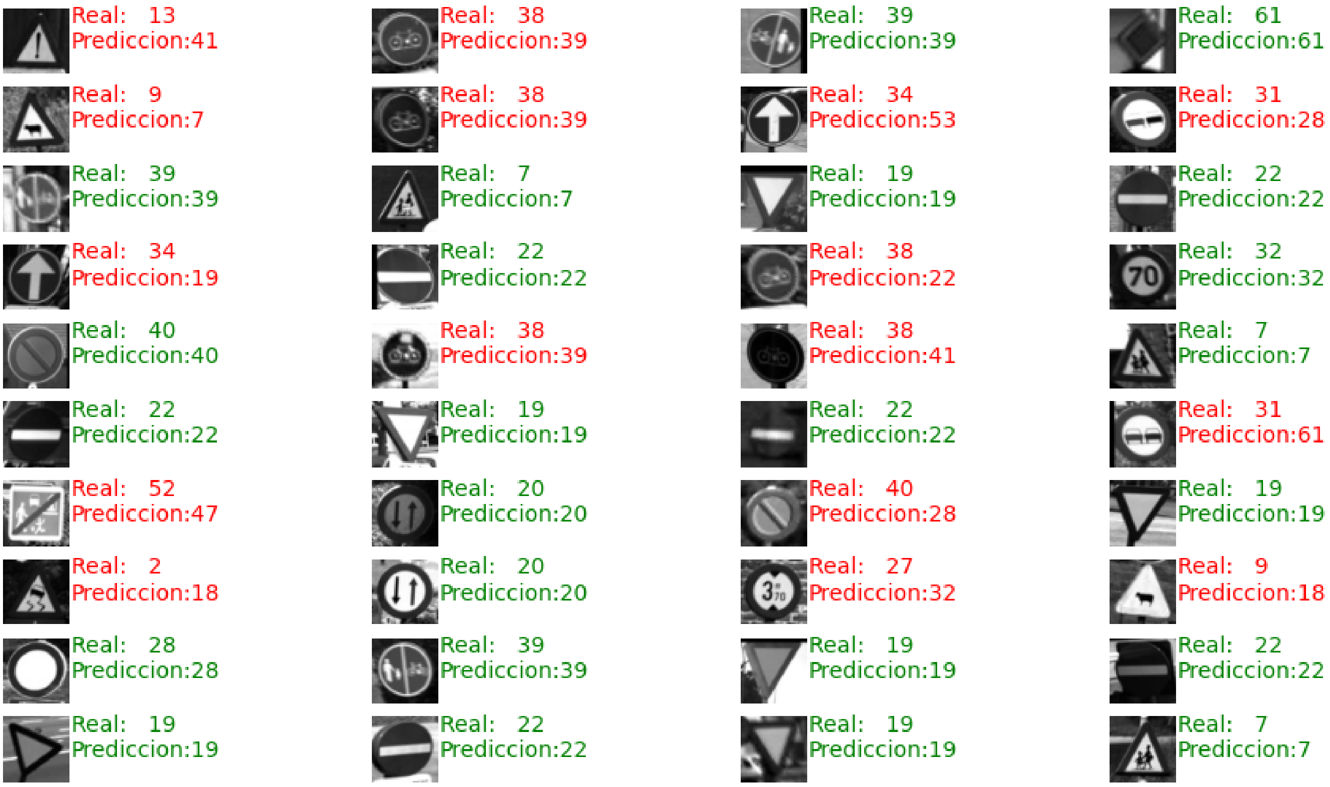
In [48]:
```
prediction = sess.run([final_pred], feed_dict={x:sample_images})[0]
```

In [49]:
```
prediction
```

Out[49]:
```
array([41, 39, 39, 61,  7, 39, 53, 28, 39,  7, 19, 22, 19, 22, 22, 32, 40,
       39, 41,  7, 22, 19, 22, 61, 47, 20, 28, 19, 18, 20, 32, 18, 28, 39,
       19, 22, 19, 22, 19,  7], dtype=int64)
```

In [50]:
```
plt.figure(figsize=(16,9))
for i in range(len(sample_images)):
```

```python
        truth = sample_labels[i]
        predi = prediction[i]
        plt.subplot(10,4,i+1) #empezamos en la 0
        plt.axis("off")
        color = "green" if truth == predi else "red"
        plt.text(51,30, "Real:   {0}\nPrediccion:{1}".format(truth, predi),
                 fontsize = 14, color = color)
        plt.imshow(sample_images[i], cmap="gray")
    plt.show()
```



In [51]:

```python
unique_labels = set(labels)
plt.figure(figsize=(16,16))
i = 1
for label in unique_labels:
    temp_im = images[list(labels).index(label)] #devuelve la posición que devuelve la etiquet
    plt.subplot(8,8,i) #matriz de 8 x 8 porque son 61 labels independientes
    plt.axis("off")
    plt.title("Clase {0} ({1})".format(label, list(labels).count(label)))
    i += 1
    plt.imshow(temp_im)

plt.show()
```

In [52]:
```
test_images, test_labels = load_ml_data(test_data_dir)
```

In [53]:
```
test_images30 = [transform.resize(im, (50, 50)) for im in test_images]
```

In [54]:
```
len(test_images30)
```

Out[54]:
```
2520
```

In [55]:
```
test_images30 = rgb2gray(np.array(test_images30))
```

In [56]:
```
prediction = sess.run([final_pred], feed_dict={x:test_images30})[0]
```

In [57]:
```
match_count = sum([int(l0==lp) for l0, lp in zip(test_labels, prediction)])
match_count
```

Out[57]:
```
1290
```

In [58]:

```
acc = match_count/len(test_labels)*100
print("Eficacia de la red neuronal: {:.2f}".format(acc))
```

```
Eficacia de la red neuronal: 51.19
```