Análisis de Datos y Aprendizaje Máquina con Tensorflow 2.0: Redes neuronales convolucionales

2019/09/30

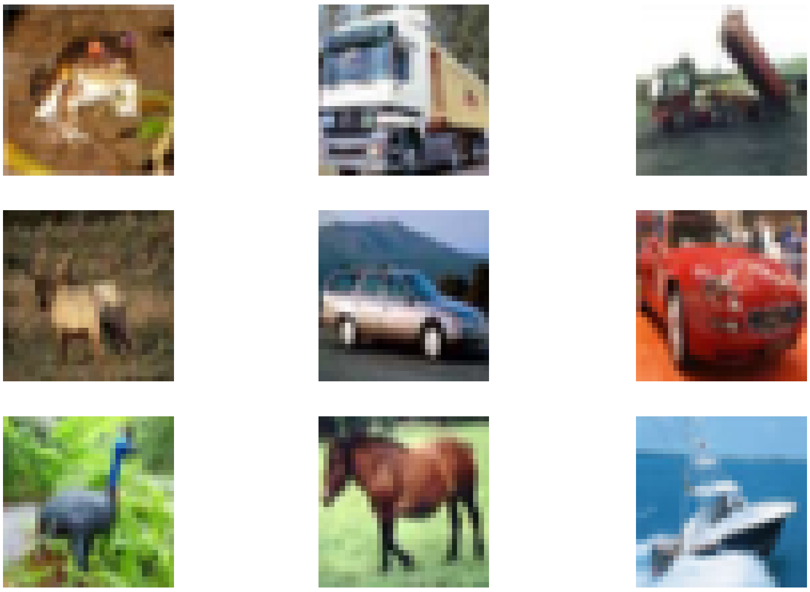
Computer Vision/Clasificación de objetos

https://www.cs.toronto.edu/~kriz/cifar.html

- Objetivo: Programar una red neuronal para reconocer objetos
- CIFAR es un acrónimo que significa Instituto Canadiense de Investigación Avanzada. El conjunto de datos CIFAR-10 consiste en 60000 imágenes a color de 32x32 para 10 clases de elementos.

```
In [9]: import matplotlib.pyplot as plt
        import tensorflow as tf
        from tensorflow import keras
        from tensorflow.keras.datasets import cifar10

        # leer dataset
        (x_train, y_train), (x_test, y_test) = cifar10.load_data()



        print('Train shape', x_train.shape, 'Target shape:',y_train.shape)
        print('Test shape', x_test.shape, 'Target shape:',y_test.shape)
        # plotea imagenes
        for i in range(9):
            # define subplot
            plt.subplot(330 + 1 + i)
            plt.axis('off')
            plt.imshow(x_train[i])



        plt.show()
```

```
Train shape (50000, 32, 32, 3) Target shape: (50000, 1)
Test shape (10000, 32, 32, 3) Target shape: (10000, 1)
```



Normalizar datos

- Número de clases es el número de neuronas que tendrá la capa de salida con 'softmax'

In [10]:
```
import numpy as np
classes = len(np.unique(y_train))
classes
```

Out[10]: 10

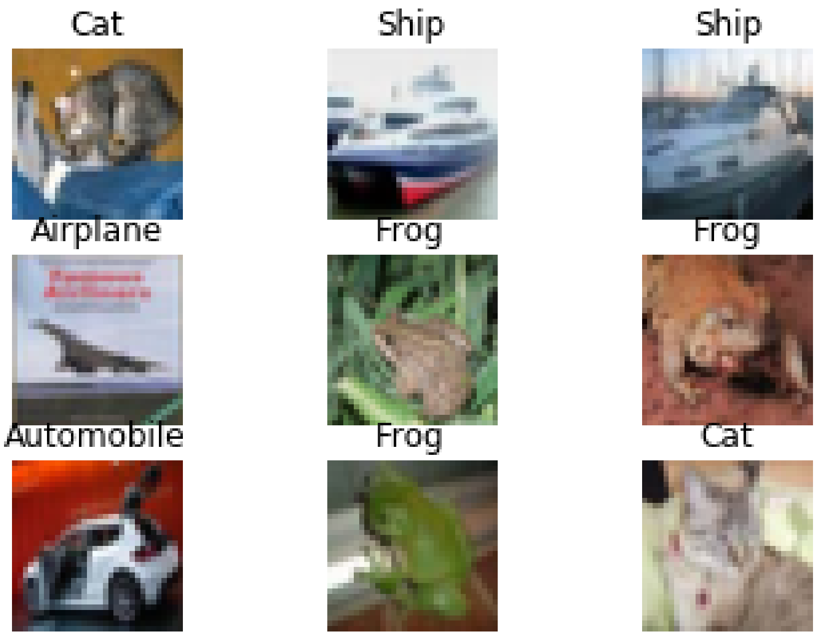
In [11]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.layers import Dropout, BatchNormalization
# con 1: no se cuenta la primera dimensión
x, y, channel = x_train.shape[1:]

input_shape = (x, y, channel)

# escalar entre 0 y 1
# escalar entre 0 y 1
x_train = x_train.reshape(x_train.shape[0], 32, 32, 3).astype('float32') / 255
x_test = x_test.reshape(x_test.shape[0], 32, 32, 3).astype('float32') / 255


print(x_train.shape) # (50000, 32, 32, 3)
print(x_test.shape)  # (10000, 32, 32, 3)
```

```
(50000, 32, 32, 3)
(10000, 32, 32, 3)
```

In [12]:
```python
int(y_test[i])
```

Out[12]:
```
3
```

In [13]:
```python
dict = {0:'Airplane', 1:'Automobile', 2:'Bird',
        3:'Cat', 4:'Deer', 5:'Dog', 6:'Frog', 7:'Horse',
        8:'Ship', 9:'Truck'}
for i in range(9):
    # define subplot
    plt.subplot(330 + 1 + i)
    plt.imshow(x_test[i])
    plt.axis('off')
    plt.title( dict[ int(y_test[i]) ] )


plt.show()
```



In [14]:
```python
epoch = 35
verbose = 1
batch = 50
```

Crear Modelo

```
In [15]: def deep_cnn():
             model = Sequential()

             model.add(Conv2D(20, (2, 2), padding='same', activation='relu',
                              input_shape=input_shape))
             model.add(Conv2D(20, (2, 2), activation='relu'))
             model.add(BatchNormalization(momentum=0.5))
             model.add(MaxPooling2D(pool_size=(2, 2)))
             model.add(Dropout(0.3))

             model.add(Conv2D(60, (2, 2), padding='same', activation='relu'))
             model.add(Conv2D(60, (2, 2), activation='relu'))
             model.add(BatchNormalization(momentum=0.5))
             model.add(MaxPooling2D(pool_size=(2, 2)))
             model.add(Dropout(0.3))

             model.add(Conv2D(120, (2, 2), padding='same', activation='relu'))
             model.add(Conv2D(120, (2, 2), activation='relu'))
             model.add(BatchNormalization(momentum=0.5))
             model.add(MaxPooling2D(pool_size=(2, 2)))
             model.add(Dropout(0.3))

             model.add(Flatten())
             model.add(Dense(64, activation='relu'))
             model.add(BatchNormalization(momentum=0.5))
             model.add(Dense(10, activation='softmax'))

             model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['ac

             return model
```

Entrenar modelo

- Nota: El entrenamiento necesita mucho poder de cómputo

```
In [16]: model1 = deep_cnn()
         model1.summary()

         history = model1.fit(x_train, y_train, batch_size = batch, validation_split = 0.3,
                              epochs = epoch, verbose = verbose)
```

```
Model: "sequential_1"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 32, 32, 20)        260
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 31, 31, 20)        1620
```

```
_________________________________________________________________
batch_normalization_4 (Batch (None, 31, 31, 20)        80
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 15, 15, 20)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 15, 15, 20)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 15, 15, 60)        4860
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 14, 14, 60)        14460
_________________________________________________________________
batch_normalization_5 (Batch (None, 14, 14, 60)        240
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 60)          0
_________________________________________________________________
dropout_4 (Dropout)          (None, 7, 7, 60)          0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 7, 7, 120)         28920
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 6, 6, 120)         57720
_________________________________________________________________
batch_normalization_6 (Batch (None, 6, 6, 120)         480
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 3, 3, 120)         0
_________________________________________________________________
dropout_5 (Dropout)          (None, 3, 3, 120)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1080)              0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                69184
_________________________________________________________________
batch_normalization_7 (Batch (None, 64)                256
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 178,730
Trainable params: 178,202
Non-trainable params: 528
_________________________________________________________________
Train on 35000 samples, validate on 15000 samples
Epoch 1/35
35000/35000 [==============================] - 11s 303us/sample - loss: 1.6156 - accuracy: 0.4233
Epoch 2/35
35000/35000 [==============================] - 8s 237us/sample - loss: 1.2252 - accuracy: 0.5632
Epoch 3/35
35000/35000 [==============================] - 8s 233us/sample - loss: 1.0634 - accuracy: 0.6223
Epoch 4/35
35000/35000 [==============================] - 8s 237us/sample - loss: 0.9754 - accuracy: 0.6570
Epoch 5/35
```

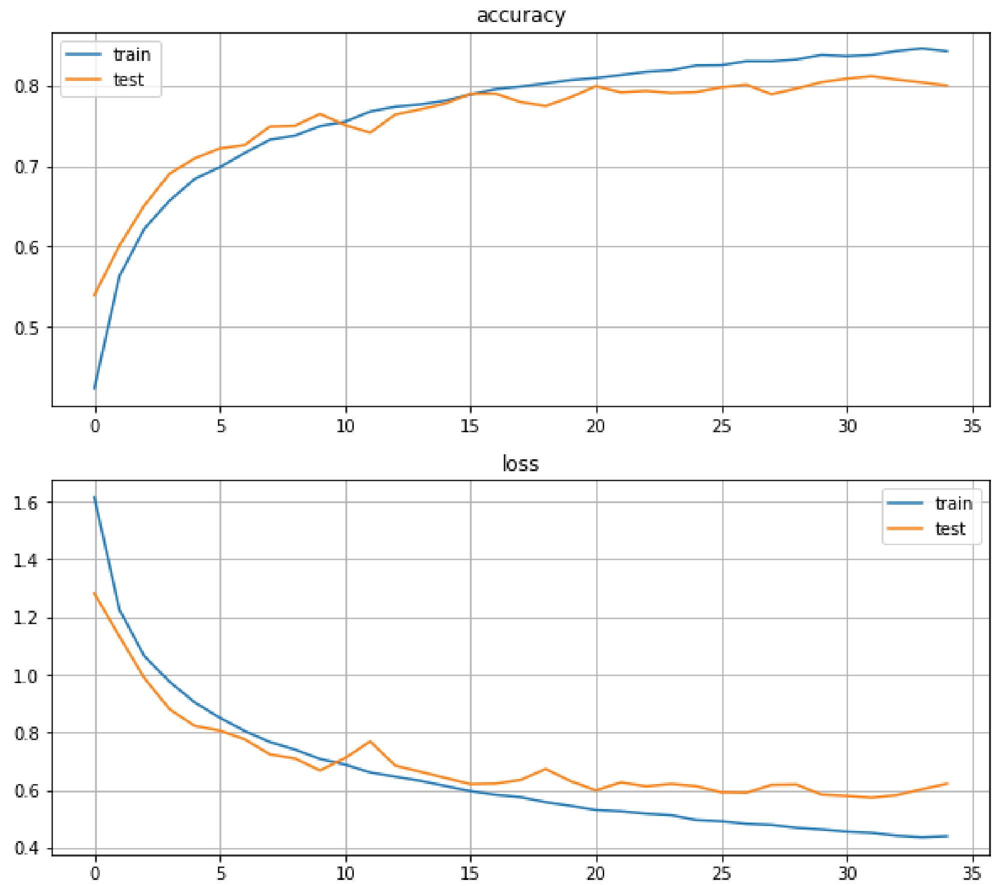
```
35000/35000 [==============================] - 8s 232us/sample - loss: 0.9038 - accuracy: 0.6839
Epoch 6/35
35000/35000 [==============================] - 8s 232us/sample - loss: 0.8506 - accuracy: 0.6985
Epoch 7/35
35000/35000 [==============================] - 8s 231us/sample - loss: 0.8043 - accuracy: 0.7163
Epoch 8/35
35000/35000 [==============================] - 8s 230us/sample - loss: 0.7660 - accuracy: 0.7327
Epoch 9/35
35000/35000 [==============================] - 8s 230us/sample - loss: 0.7400 - accuracy: 0.7378
Epoch 10/35
35000/35000 [==============================] - 8s 229us/sample - loss: 0.7072 - accuracy: 0.7497
Epoch 11/35
35000/35000 [==============================] - 8s 229us/sample - loss: 0.6877 - accuracy: 0.7552
Epoch 12/35
35000/35000 [==============================] - 8s 231us/sample - loss: 0.6605 - accuracy: 0.7678
Epoch 13/35
35000/35000 [==============================] - 8s 238us/sample - loss: 0.6458 - accuracy: 0.7737
Epoch 14/35
35000/35000 [==============================] - 8s 236us/sample - loss: 0.6313 - accuracy: 0.7766
Epoch 15/35
35000/35000 [==============================] - 8s 232us/sample - loss: 0.6134 - accuracy: 0.7813
Epoch 16/35
35000/35000 [==============================] - 8s 234us/sample - loss: 0.5958 - accuracy: 0.7891
Epoch 17/35
35000/35000 [==============================] - 8s 232us/sample - loss: 0.5828 - accuracy: 0.7956
Epoch 18/35
35000/35000 [==============================] - 8s 242us/sample - loss: 0.5747 - accuracy: 0.7988
Epoch 19/35
35000/35000 [==============================] - 9s 244us/sample - loss: 0.5575 - accuracy: 0.8028
Epoch 20/35
35000/35000 [==============================] - 8s 232us/sample - loss: 0.5447 - accuracy: 0.8070
Epoch 21/35
35000/35000 [==============================] - 8s 232us/sample - loss: 0.5302 - accuracy: 0.8095
Epoch 22/35
35000/35000 [==============================] - 8s 229us/sample - loss: 0.5255 - accuracy: 0.8133
Epoch 23/35
35000/35000 [==============================] - 8s 234us/sample - loss: 0.5174 - accuracy: 0.8172
Epoch 24/35
35000/35000 [==============================] - 8s 231us/sample - loss: 0.5120 - accuracy: 0.8191
Epoch 25/35
35000/35000 [==============================] - 8s 235us/sample - loss: 0.4956 - accuracy: 0.8251
Epoch 26/35
35000/35000 [==============================] - 8s 231us/sample - loss: 0.4912 - accuracy: 0.8255
Epoch 27/35
35000/35000 [==============================] - 8s 228us/sample - loss: 0.4824 - accuracy: 0.8304
Epoch 28/35
35000/35000 [==============================] - 8s 227us/sample - loss: 0.4786 - accuracy: 0.8303
Epoch 29/35
35000/35000 [==============================] - 8s 225us/sample - loss: 0.4684 - accuracy: 0.8327
```

```
Epoch 30/35
35000/35000 [==============================] - 8s 229us/sample - loss: 0.4626 - accuracy: 0.8383
Epoch 31/35
35000/35000 [==============================] - 8s 236us/sample - loss: 0.4549 - accuracy: 0.8368
Epoch 32/35
35000/35000 [==============================] - 8s 231us/sample - loss: 0.4508 - accuracy: 0.8383
Epoch 33/35
35000/35000 [==============================] - 8s 233us/sample - loss: 0.4411 - accuracy: 0.8430
Epoch 34/35
35000/35000 [==============================] - 8s 233us/sample - loss: 0.4355 - accuracy: 0.8463
Epoch 35/35
35000/35000 [==============================] - 8s 237us/sample - loss: 0.4394 - accuracy: 0.8428
```

```
In [17]: test_loss, test_acc = model1.evaluate(x_test, y_test, verbose = 0)

         print('\nTest acccuracy:', test_acc)
```

```
Test acccuracy: 0.7944
```

```
In [18]: #plot
         plt.figure(figsize=(10,9))

         plt.subplot(211)
         plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.title('accuracy')
         plt.legend(['train', 'test'])
         plt.grid()
         plt.subplot(212)
         plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title('loss')
         plt.legend(['train', 'test'])
         plt.grid()




         plt.show()
```



Usar red neuronal para reconocer objetos

```
In [19]: x_test[[0]].shape
```

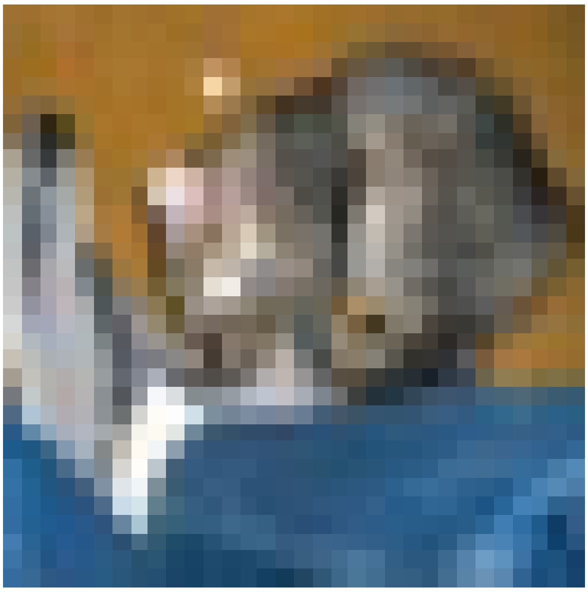
```
Out[19]: (1, 32, 32, 3)
```

```
In [20]: np.argmax( model1.predict(x_test[[0]]) )
```

```
Out[20]: 3
```

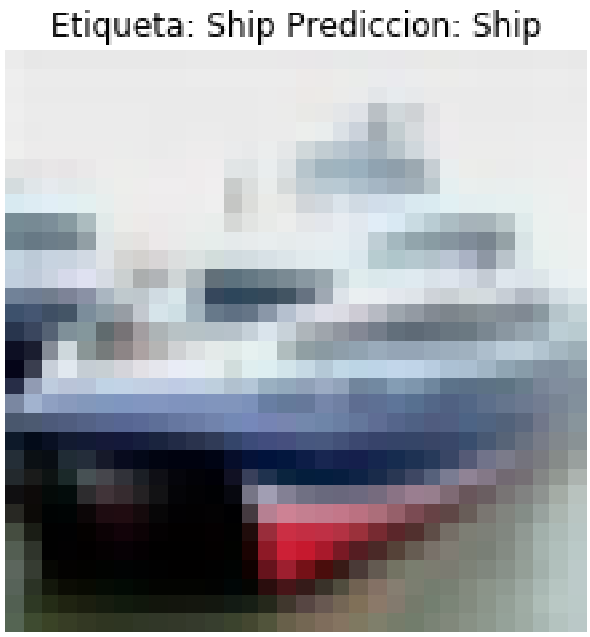
```
In [21]: plt.imshow(x_test[0])
         plt.axis('off')
         plt.title( 'Etiqueta: ' + dict[ int(y_test[0])] + ' '
                    'Prediccion: ' + dict[ np.argmax( model1.predict(x_test[[0]]) ) ] )
```

```
Out[21]: Text(0.5, 1.0, 'Etiqueta: Cat Prediccion: Cat')
```



In [22]:
```
plt.imshow(x_test[1])
plt.axis('off')
plt.title( 'Etiqueta: ' + dict[ int(y_test[1])] + ' '
           'Prediccion: ' + dict[ np.argmax( model1.predict(x_test[[1]]) ) ] )
```

Out[22]:
```
Text(0.5, 1.0, 'Etiqueta: Ship Prediccion: Ship')
```



- Experimentar con la arquitectura de la red, agregando neuronas y capas
- Agregar y modificar la regularización para entrenar en menor tiempo
- Mejorar test accuracy