Michael Woitass

Proyecto 1

Descripción:

Cambio de arquitectura -> añadimos dos capas de convoluciones y una de pooling

```
[5] model = ks.Sequential()

    model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                               padding='same', input_shape=(32,32,3)))
    model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                               padding='same', input_shape=(32,32,3)))
    model.add(ks.layers.MaxPooling2D((2, 2)))
    model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                               padding='same', input_shape=(32,32,3)))
    model.add(ks.layers.MaxPooling2D((2, 2)))

    model.add(ks.layers.Flatten())
    model.add(ks.layers.Dense(32, activation='relu'))
    model.add(ks.layers.Dense(10, activation='softmax'))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 128)       3584

 conv2d_1 (Conv2D)           (None, 32, 32, 64)        73792

 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)       0
 )

 conv2d_2 (Conv2D)           (None, 16, 16, 32)        18464

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 32)         0
 2D)

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 32)                65568

 dense_1 (Dense)             (None, 10)                330

=================================================================
Total params: 161,738
Trainable params: 161,738
Non-trainable params: 0
_________________________________________________________________
```

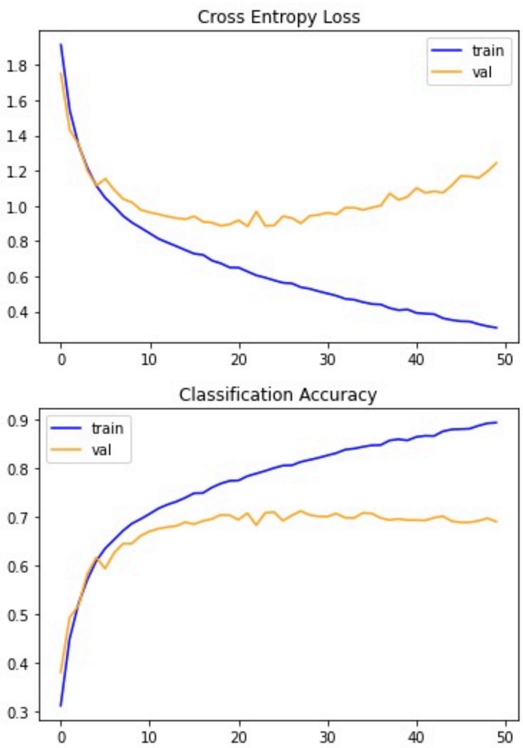
Modelo:

```
# Fijamos epochs 50 y batch size 512
history = model.fit(x_train_scaled, y_train, epochs=50,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

Entrenamiento:

```
[20] _, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
     print('> %.3f' % (acc * 100.0))

     > 67.760
```

Evaluación del resultado:



Conclusiones:

- Overfitting -> cambiamos pacience a 5.

- Accuracy -> añadimos más capas de convoluciones y pooling para conseguir aumentar el valor.

Proyecto 2

Descripción:

Cambio de arquitectura -> añadimos dos capas de convoluciones y una de pooling

Modelo:

```
[3]  model = ks.Sequential()

     model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                                padding='same', input_shape=(32,32,3)))
     model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                                padding='same', input_shape=(32,32,3)))
     model.add(ks.layers.MaxPooling2D((2, 2)))

     model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                                padding='same', input_shape=(32,32,3)))
     model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                                padding='same', input_shape=(32,32,3)))
     model.add(ks.layers.MaxPooling2D((2, 2)))

     model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                                padding='same', input_shape=(32,32,3)))
     model.add(ks.layers.MaxPooling2D((2, 2)))

     model.add(ks.layers.Flatten())
     model.add(ks.layers.Dense(32, activation='relu'))
     model.add(ks.layers.Dense(10, activation='softmax'))
```

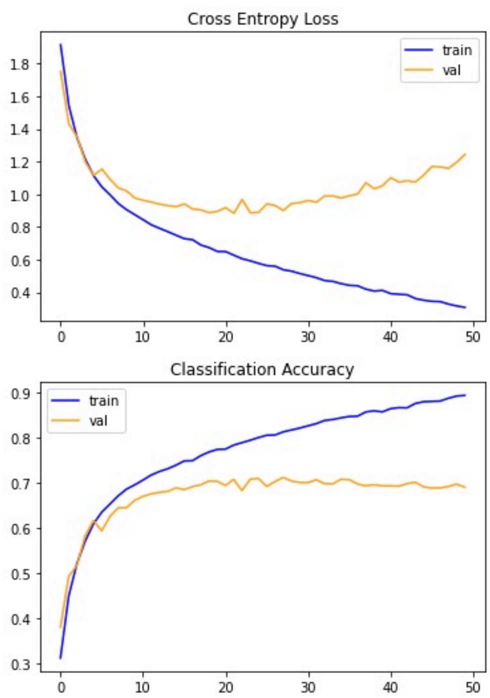
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 128)       3584

 conv2d_1 (Conv2D)           (None, 32, 32, 128)       147584

 max_pooling2d (MaxPooling2D  (None, 16, 16, 128)      0
 )

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        73792

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 32)          18464

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 32)         0
 2D)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 32)                16416

 dense_1 (Dense)             (None, 10)                330

=================================================================
Total params: 297,098
Trainable params: 297,098
Non-trainable params: 0
_________________________________________________________________
```

Entrenamiento:

```
# Fijamos epochs 50 y batch size 512
history = model.fit(x_train_scaled, y_train, epochs=50,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

Evaluación del resultado:

```
[16] _, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
     print('> %.3f' % (acc * 100.0))

     > 72.480
```



Conclusiones:

- Overfitting -> subimos paciencie a 8, dejando un poco más de margen.
- Accuracy -> añadimos una primera capa de 32 neuronas así como dos capas de convoluciones y pooling al final para conseguir aumentar el valor, reordenamos las primeras capas para que el número de neuronas suba paulatinamente.

Proyecto 3

Descripción:

Cambio de arquitectura -> capas de convoluciones y de pooling, aumentando sucesivamente el número de neuronas

Modelo:

```
[4]  model = ks.Sequential()

     model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                                padding='same', input_shape=(32,32,3)))
     model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                                padding='same', input_shape=(32,32,3)))
     model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                                padding='same', input_shape=(32,32,3)))
     model.add(ks.layers.MaxPooling2D((2, 2)))

     model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                                padding='same', input_shape=(32,32,3)))
     model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                                padding='same', input_shape=(32,32,3)))
     model.add(ks.layers.MaxPooling2D((2, 2)))

     model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                                padding='same', input_shape=(32,32,3)))
     model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                                padding='same', input_shape=(32,32,3)))
     model.add(ks.layers.MaxPooling2D((2, 2)))

     model.add(ks.layers.Flatten())
     model.add(ks.layers.Dense(32, activation='relu'))
     model.add(ks.layers.Dense(10, activation='softmax'))
```

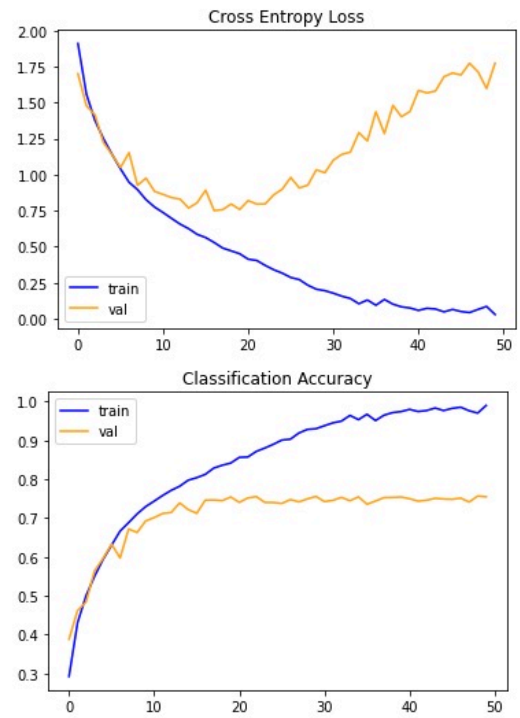
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 conv2d_1 (Conv2D)           (None, 32, 32, 64)        18496

 conv2d_2 (Conv2D)           (None, 32, 32, 64)        36928

 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)       0
 )

 conv2d_3 (Conv2D)           (None, 16, 16, 128)       73856

 conv2d_4 (Conv2D)           (None, 16, 16, 128)       147584

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 conv2d_5 (Conv2D)           (None, 8, 8, 32)          36896

 conv2d_6 (Conv2D)           (None, 8, 8, 32)          9248

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 32)         0
 2D)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 32)                16416

 dense_1 (Dense)             (None, 10)                330

=================================================================
Total params: 340,650
Trainable params: 340,650
Non-trainable params: 0
_________________________________________________________________
```

Entrenamiento:

```
# Fijamos epochs 50 y batch size 512
history = model.fit(x_train_scaled, y_train, epochs=50,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

Evaluación del resultado:

```
[ ] _, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
    print('> %.3f' % (acc * 100.0))

    > 75.510
```



Conclusiones:

- Accuracy -> la red necesita más capas con un aumento considerable de neuronas.

Proyecto 4

Descripción:

Cambio de arquitectura -> capas de convoluciones y de pooling, aumentando sucesivamente el número de neuronas hasta 512

Modelo:

```python
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(32, activation='relu'))
model.add(ks.layers.Dense(10, activation='softmax'))
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 32, 32, 32)        896

 conv2d_9 (Conv2D)           (None, 32, 32, 64)        18496

 max_pooling2d_4 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 conv2d_10 (Conv2D)          (None, 16, 16, 128)       73856

 conv2d_11 (Conv2D)          (None, 16, 16, 128)       147584

 max_pooling2d_5 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 conv2d_12 (Conv2D)          (None, 8, 8, 256)         295168

 conv2d_13 (Conv2D)          (None, 8, 8, 256)         590080

 max_pooling2d_6 (MaxPooling  (None, 4, 4, 256)        0
 2D)

 conv2d_14 (Conv2D)          (None, 4, 4, 512)         1180160

 conv2d_15 (Conv2D)          (None, 4, 4, 512)         2359808

 max_pooling2d_7 (MaxPooling  (None, 2, 2, 512)        0
 2D)

 flatten_1 (Flatten)         (None, 2048)              0

 dense_2 (Dense)             (None, 32)                65568

 dense_3 (Dense)             (None, 10)                330

=================================================================
Total params: 4,731,946
Trainable params: 4,731,946
Non-trainable params: 0
```

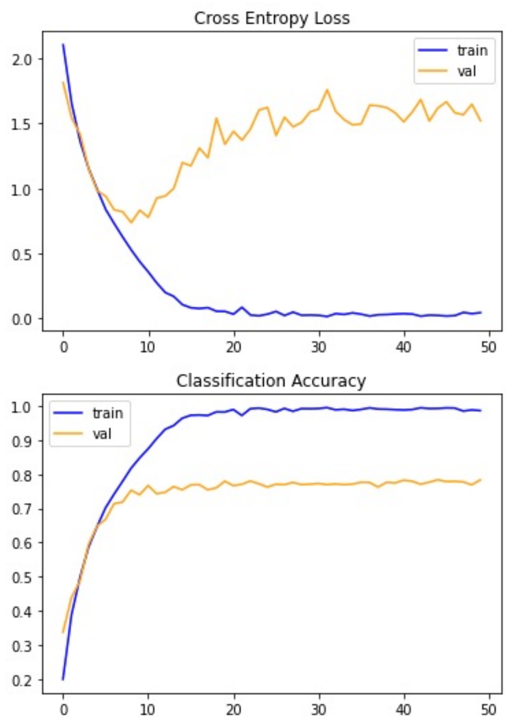
Entrenamiento:

```
# Fijamos epochs 50 y batch size 512
history = model.fit(x_train_scaled, y_train, epochs=50,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

Evaluación del resultado:

```
_, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
```

```
> 78.020
```



Conclusiones:

- Loss -> el modelo aprende rápidamente clasificar los datos de train, parece que memoriza bien, pero falla con los datos de validación.

- Accuracy -> sigue el overfitting, hay que añadir técnicas que impiden la memorización, p.ej. drop-out.

Proyecto 5

Descripción:

Se mantiene la arquitectura de capas, añadimos drop-outs en la parte de la extracción de características

Modelo:

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(32, activation='relu'))
model.add(ks.layers.Dense(10, activation='softmax'))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 conv2d_1 (Conv2D)           (None, 32, 32, 64)        18496

 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)       0
 )

 dropout (Dropout)           (None, 16, 16, 64)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 128)       73856

 conv2d_3 (Conv2D)           (None, 16, 16, 128)       147584

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 dropout_1 (Dropout)         (None, 8, 8, 128)         0

 conv2d_4 (Conv2D)           (None, 8, 8, 256)         295168

 conv2d_5 (Conv2D)           (None, 8, 8, 256)         590080

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 256)        0
 2D)

 dropout_2 (Dropout)         (None, 4, 4, 256)         0

 conv2d_6 (Conv2D)           (None, 4, 4, 512)         1180160
```

```
 conv2d_7 (Conv2D)           (None, 4, 4, 512)         2359808

 max_pooling2d_3 (MaxPooling  (None, 2, 2, 512)        0
 2D)

 dropout_3 (Dropout)         (None, 2, 2, 512)         0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 32)                65568

 dense_1 (Dense)             (None, 10)                330

=================================================================
Total params: 4,731,946
Trainable params: 4,731,946
Non-trainable params: 0
_________________________________________________________________
```

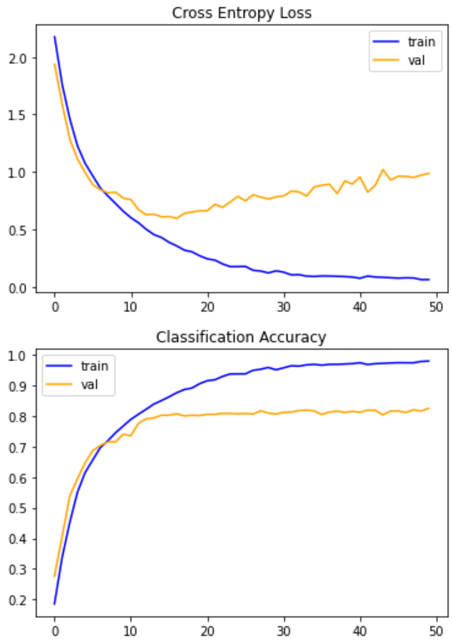
Entrenamiento:

```
# Fijamos epochs 50 y batch size 512
history = model.fit(x_train_scaled, y_train, epochs=50,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

Evaluación del resultado:

```
_, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
```

```
> 80.850
```



Conclusiones:

- Loss -> falta todavía reducir el error con los datos de validación.

- Accuracy -> el overfitting extremo se ha retrasado, hay que mejorar la arquitectura de la parte de clasificación.

Proyecto 6

Descripción:

Cambiamos la arquitectura en la parte de clasificación, añadimos convoluciones con una cantidad considerable de neuronas y con drop-outs

Modelo:

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(10, activation='softmax'))
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 128)       147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 256)         590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 256)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 4, 4, 512)         1180160
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 4, 4, 512)         2359808
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 512)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 2, 2, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1049088
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 5,982,922
Trainable params: 5,982,922
```

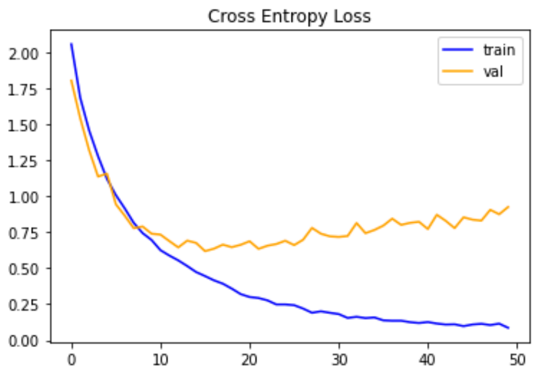
Entrenamiento:

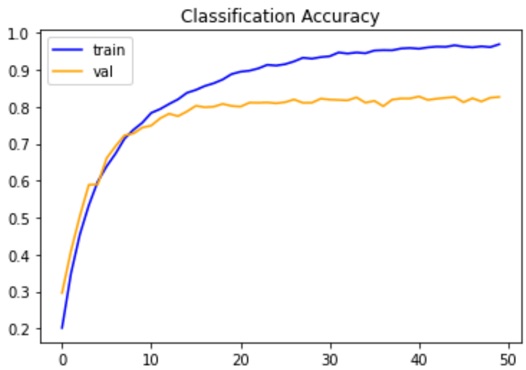
```
# Fijamos epochs 50 y batch size 512
history = model.fit(x_train_scaled, y_train, epochs=50,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

Evaluación del resultado:

```
_, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
```

```
> 82.410
```





Conclusiones:

- Loss -> la divergencia entre train y validation se ha reducido un poco, todavía falta reducir el error con los datos de validación.

- Accuracy -> todavía hay una distancia de 0.1 entre train y validation, al menos se mantiene constante.

Proyecto 7

Descripción:

Mantenemos la arquitectura, cambiamos el optimizer de Adam a SGD para comparar los resultados

Modelo: idéntico al modelo del proyecto 6

```python
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(10, activation='softmax'))
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
conv2d_1 (Conv2D)	(None, 32, 32, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 128)	0
dropout_1 (Dropout)	(None, 8, 8, 128)	0
conv2d_4 (Conv2D)	(None, 8, 8, 256)	295168
conv2d_5 (Conv2D)	(None, 8, 8, 256)	590080
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
conv2d_6 (Conv2D)	(None, 4, 4, 512)	1180160
conv2d_7 (Conv2D)	(None, 4, 4, 512)	2359808
max_pooling2d_3 (MaxPooling2	(None, 2, 2, 512)	0
dropout_3 (Dropout)	(None, 2, 2, 512)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 512)	1049088
dropout_4 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 512)	262656
dropout_5 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130

Total params: 5,982,922
Trainable params: 5,982,922

Optimizer:

```
opt = ks.optimizers.SGD(learning_rate=0.01, momentum=0.9)
```

```
model.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

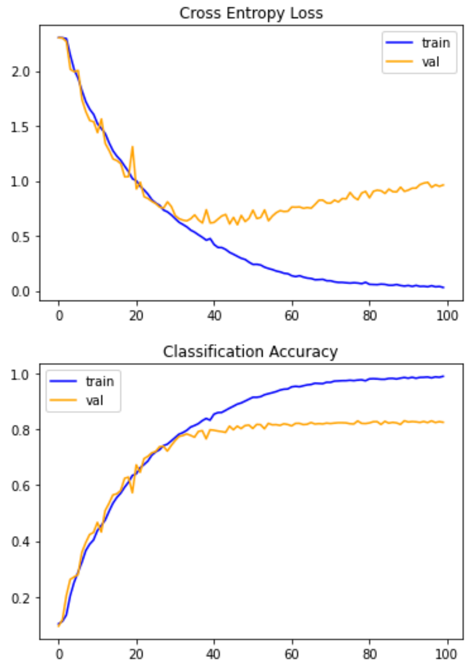
Entrenamiento:

```
# Fijamos epochs 100 y batch size 512
history = model.fit(x_train_scaled, y_train, epochs=100,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

Evaluación del resultado:

```
_, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
```

```
> 82.140
```



Conclusiones:

- Los resultados son muy parecidos al modelo 6, tanto en el ratio del accuracy como en las divergencias en las curvas loss y accuracy.

- SGD es más lento que Adam, y empieza a divergir 30 epochs más tarde.

Proyecto 8

Descripción:

Mantenemos la arquitectura, cambiamos de nuevo el optimizer a Adam, y probamos learning rates inferiores al ratio por defecto 0.001

Modelo:

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(10, activation='softmax'))
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
conv2d_1 (Conv2D)	(None, 32, 32, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 128)	0
dropout_1 (Dropout)	(None, 8, 8, 128)	0
conv2d_4 (Conv2D)	(None, 8, 8, 256)	295168
conv2d_5 (Conv2D)	(None, 8, 8, 256)	590080
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
conv2d_6 (Conv2D)	(None, 4, 4, 512)	1180160
conv2d_7 (Conv2D)	(None, 4, 4, 512)	2359808
max_pooling2d_3 (MaxPooling2	(None, 2, 2, 512)	0
dropout_3 (Dropout)	(None, 2, 2, 512)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 512)	1049088
dropout_4 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 512)	262656
dropout_5 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130

Total params: 5,982,922
Trainable params: 5.982.922

Optimizer:

```
Lazy_Adam = ks.optimizers.Adam(learning_rate=0.0003)
```

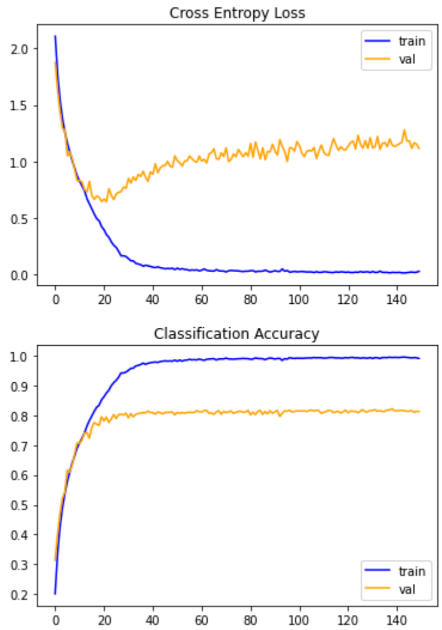
Entrenamiento:

```
# Fijamos epochs 150 y batch size 512
history = model.fit(x_train_scaled, y_train, epochs=150,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

Evaluación del resultado:

```
_, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
```

```
> 80.680
```



Conclusiones:

- Los resultados empeoran, la accuracy está por debajo de los resultados en los dos proyectos anteriores: proyecto 6 (Adam, learning rate 0.001) y proyecto 7 (SGD, learning rate 0.01 y momentum 0.9).

- Las curvas de loss siguen divergiendo.

Proyecto 9

Descripción:

Mantenemos la arquitectura, usamos el optimizer Adam con valores por defecto, y aplicamos data augmentation.

Modelo:

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(10, activation='softmax'))
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
conv2d_1 (Conv2D)	(None, 32, 32, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 128)	0
dropout_1 (Dropout)	(None, 8, 8, 128)	0
conv2d_4 (Conv2D)	(None, 8, 8, 256)	295168
conv2d_5 (Conv2D)	(None, 8, 8, 256)	590080
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
conv2d_6 (Conv2D)	(None, 4, 4, 512)	1180160
conv2d_7 (Conv2D)	(None, 4, 4, 512)	2359808
max_pooling2d_3 (MaxPooling2	(None, 2, 2, 512)	0
dropout_3 (Dropout)	(None, 2, 2, 512)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 512)	1049088
dropout_4 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 512)	262656
dropout_5 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130

Total params: 5,982,922
Trainable params: 5,982,922

Parámetros para data augmentation:

```
rotation_range = 15,
zoom_range = 0.2,
horizontal_flip = True,
brightness_range = (0.6, 1.0),
shear_range = 0.5
```

Optimizer:

```
model.compile(optimizer='Adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

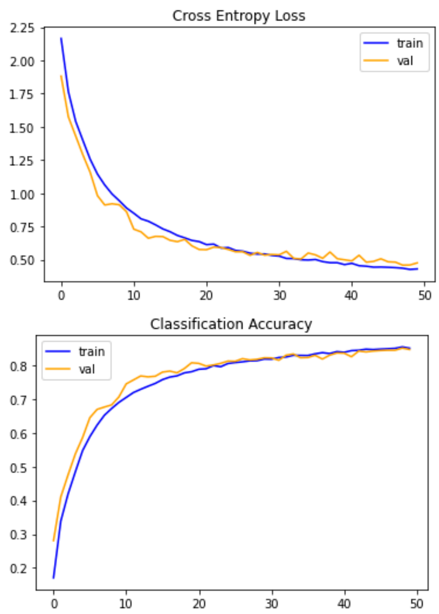
Entrenamiento:

```
history = model.fit(train_generator, epochs=50,
                    validation_data=validation_generator,
                    steps_per_epoch=100, validation_steps=50,
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

Evaluación del resultado:

```
_, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
```

```
> 83.920
```



Conclusiones:

- La accuracy ha subido dos puntos en relación al mejor resultado anterior, llega casi al 0.84.

- Tanto las curvas de loss como las de accuracy están muy alineadas, las divergencias han desaparecido.

Proyecto 10

Descripción:

Mantenemos la arquitectura, usamos el optimizer SGD con valores por defecto, y aplicamos data augmentation con más parámetros.

Modelo:

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (3, 3), activation='relu', kernel_regularizer=l2(0.001), padding= 'same',
                    kernel_initializer='he_uniform', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.001), padding= 'same',
                    kernel_initializer='he_uniform', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Conv2D(128, (3, 3), activation='relu', kernel_regularizer=l2(0.001), padding= 'same',
                    kernel_initializer='he_uniform', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), activation='relu', kernel_regularizer=l2(0.001), padding= 'same',
                    kernel_initializer='he_uniform', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(256, (3, 3), activation='relu', kernel_regularizer=l2(0.001), padding= 'same',
                    kernel_initializer='he_uniform', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(256, (3, 3), activation='relu', kernel_regularizer=l2(0.001), padding= 'same',
                    kernel_initializer='he_uniform', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(512, (3, 3), activation='relu', kernel_regularizer=l2(0.001), padding= 'same',
                    kernel_initializer='he_uniform', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(512, (3, 3), activation='relu', kernel_regularizer=l2(0.001), padding= 'same',
                    kernel_initializer='he_uniform', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu', kernel_regularizer=l2(0.001), kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(512, activation='relu', kernel_regularizer=l2(0.001), kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(10, activation='softmax'))
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 64)        18496

 batch_normalization_1 (Batc  (None, 32, 32, 64)       256
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)       0
 )

 dropout (Dropout)           (None, 16, 16, 64)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 128)       73856
```

```
batch_normalization_2 (Batc  (None, 16, 16, 128)     512
hNormalization)

conv2d_3 (Conv2D)           (None, 16, 16, 128)      147584

batch_normalization_3 (Batc  (None, 16, 16, 128)     512
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 8, 8, 128)       0
2D)

dropout_1 (Dropout)         (None, 8, 8, 128)        0

conv2d_4 (Conv2D)           (None, 8, 8, 256)        295168

batch_normalization_4 (Batc  (None, 8, 8, 256)       1024
hNormalization)

conv2d_5 (Conv2D)           (None, 8, 8, 256)        590080

batch_normalization_5 (Batc  (None, 8, 8, 256)       1024
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 4, 4, 256)       0
2D)

dropout_2 (Dropout)         (None, 4, 4, 256)        0

conv2d_6 (Conv2D)           (None, 4, 4, 512)        1180160

batch_normalization_6 (Batc  (None, 4, 4, 512)       2048
hNormalization)

conv2d_7 (Conv2D)           (None, 4, 4, 512)        2359808

batch_normalization_7 (Batc  (None, 4, 4, 512)       2048
hNormalization)

max_pooling2d_3 (MaxPooling  (None, 2, 2, 512)       0
2D)

dropout_3 (Dropout)         (None, 2, 2, 512)        0

flatten (Flatten)           (None, 2048)             0

dense (Dense)               (None, 512)              1049088

batch_normalization_8 (Batc  (None, 512)             2048
hNormalization)

dropout_4 (Dropout)         (None, 512)              0

dense_1 (Dense)             (None, 512)              262656

batch_normalization_9 (Batc  (None, 512)             2048
hNormalization)

dropout_5 (Dropout)         (None, 512)              0

dense_2 (Dense)             (None, 10)               5130

=================================================================
Total params: 5,994,570
Trainable params: 5,988,746
Non-trainable params: 5,824
_________________________________________________________________
```

Parámetros para data augmentation:

```
rotation_range = 15,
zoom_range = 0.2,
horizontal_flip = True,
brightness_range = (0.7, 1.0),
shear_range = 0.3
```

Optimizer:

```
opt = ks.optimizers.SGD(learning_rate=0.01, momentum=0.9)

model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

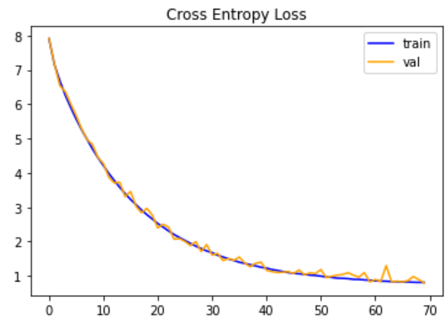
Entrenamiento:

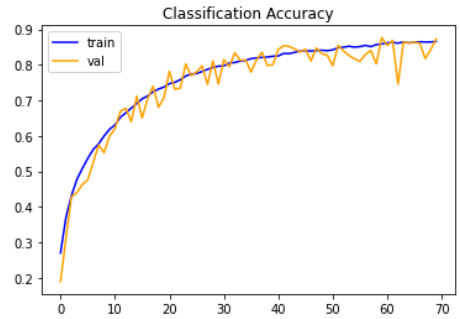
```
# Fijamos los steps en función de los tamaños del batch en los generadores de imagenes:
# train_generator y validation_generator
# train_generator tiene un batch_size = 256, si tengo 40000 imágenes, necesito 156 steps
# validation_generator tiene un batch_size de 64, si tengo 10000 imágenes, necesito 156 steps

history = model.fit(train_generator, epochs=100,
                    validation_data=validation_generator,
                    steps_per_epoch=156, validation_steps=156,
                    callbacks=[callback_checkpoint, callback_val_loss, callback_val_accuracy])
```

Evaluación del resultado:

```
_, acc = model.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
```

```
> 87.530
```





Conclusiones:

- La accuracy ha subido de nuevo y supera 0.87.
- Las curvas de loss están muy alineadas, y el valor va bajando continuamente, lo que indica que el modelo funciona bien.
- La curvas de accuracy están igualmente bastante alineadas, ambas superan al final claramente el 80%.

Proyecto 11

Descripción:

Probamos Transfer Learning, cambiamos la arquitectura a VGG16. Usamos el optimizer SGD con valores por defecto, y aplicamos data augmentation.

Modelo:

```
vgg = vgg16.VGG16(include_top=False, weights='imagenet', input_shape=(32,32,3))
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 32, 32, 3)]	0
block1_conv1 (Conv2D)	(None, 32, 32, 64)	1792
block1_conv2 (Conv2D)	(None, 32, 32, 64)	36928
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	0
block2_conv1 (Conv2D)	(None, 16, 16, 128)	73856
block2_conv2 (Conv2D)	(None, 16, 16, 128)	147584
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	0
block3_conv1 (Conv2D)	(None, 8, 8, 256)	295168
block3_conv2 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv3 (Conv2D)	(None, 8, 8, 256)	590080
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	0
block4_conv1 (Conv2D)	(None, 4, 4, 512)	1180160
block4_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block4_conv3 (Conv2D)	(None, 4, 4, 512)	2359808
block4_pool (MaxPooling2D)	(None, 2, 2, 512)	0
block5_conv1 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv2 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv3 (Conv2D)	(None, 2, 2, 512)	2359808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

```
model_with_vgg = ks.Sequential()

model_with_vgg.add(vgg_model)
model_with_vgg.add(ks.layers.Dense(512, activation='relu', input_shape=(input_shape,)))
model_with_vgg.add(ks.layers.Dropout(0.3))
model_with_vgg.add(ks.layers.Dense(512, activation='relu'))
model_with_vgg.add(ks.layers.Dropout(0.3))
model_with_vgg.add(ks.layers.Dense(10, activation='softmax'))

model_with_vgg.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
model (Functional)	(None, 512)	14714688
dense (Dense)	(None, 512)	262656
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 512)	262656
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130

Total params: 15,245,130
Trainable params: 13,509,642
Non-trainable params: 1,735,488

Parámetros para data augmentation:

```
rotation_range = 15,
zoom_range = 0.3,
horizontal_flip = True,
brightness_range = (0.7, 1.0),
shear_range = 0.2,
width_shift_range=0.2,
height_shift_range=0.2,
fill_mode='nearest'
```

Optimizer:

```
opt = ks.optimizers.SGD(learning_rate=0.01, momentum=0.9)

model_with_vgg.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Entrenamiento:

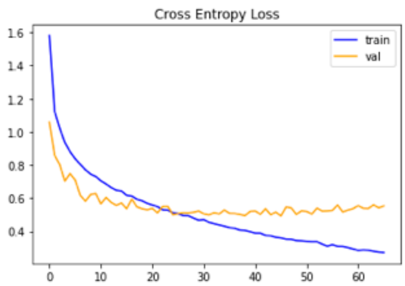
```
# Usamos Early Stopping con patience 20
callback_val_loss = EarlyStopping(monitor='val_loss', patience=20)
callback_val_accuracy = EarlyStopping(monitor='val_accuracy', patience=20)
```

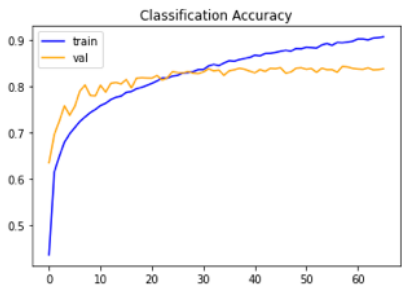
```
# Cambiamos el batch_size por los valores de los generadores de imagenes:
# train_generator y validation_generator

# steps_per_epochs = total imagenes / batch_size del train generator = 40000 / 256 = 156
# validation_steps = total imagenes validacion / batch_size del validation generator = 10000 / 64 = 156


# train_generator tiene un batch_size = 128, si tengo 40000 imagenes, hará 312 steps
# validation_generator tiene un batch_size de 64, si tengo 10000 imagenes (validation), genera 156 steps

history = model_with_vgg.fit(train_generator, epochs=150,
                    validation_data=validation_generator,
                    steps_per_epoch=156, validation_steps=156,
                    callbacks=[callback_checkpoint, callback_val_loss, callback_val_accuracy])
```

Evaluación del resultado:

```
_, acc = model_with_vgg.evaluate(x_test_scaled, y_test, verbose=0)
print('> %.3f' % (acc * 100.0))
```

```
> 82.960
```





Conclusiones:

- La accuracy ha bajado a 0.83.
- Tanto en loss como en accuracy las dos curvas empiezan a divergir a partir de 30 epochs.
- Parece que las curvas de train estén bien: debido al uso de VGG16, las trayectorias son más pronunciadas en el tramo inicial, en comparación con los dos proyectos anteriores.
- Las curvas de val se estancan, y se convierten prácticamente en líneas rectas.