Entregable 4

Silvia Julià

NUCLIO DIGITAL SCHOOL

Proyecto I

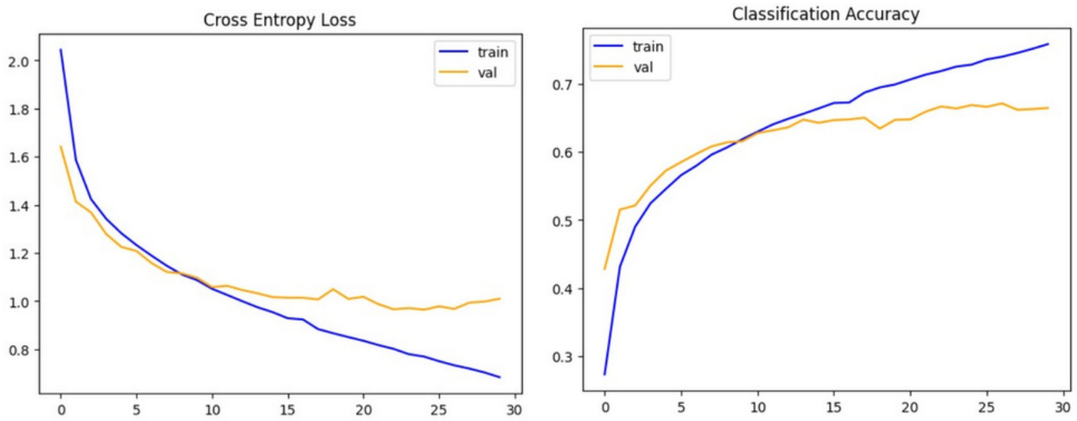
```
model = ks.Sequential()

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

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(500, activation='relu'))
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(10, activation='softmax'))
```

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

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 500)               4096500

 dropout (Dropout)           (None, 500)               0

 dense_1 (Dense)             (None, 10)                5010

=================================================================
Total params: 4102406 (15.65 MB)
Trainable params: 4102406 (15.65 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
history = model.fit(x_train_scaled, y_train,
                    callbacks=[modelchk, callback_loss, callback_acc],
                    epochs=300, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

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

```
> 65.360
```



Conclusiones y observaciones:

En este primer proyecto aumentamos neuronas en la capa dense, y añadimos un dropout para controlar un poco el overfitting que puede causar este aumento.

El resultado es un accuracy mucho mayor (el proyecto base tenía 60.260) aunque aun falta mejorar tanto el accuracy como el overffiting.

Proyecto II

```
model = Sequential()

# Primera capa de convolución
model.add(Conv2D(32, (3, 3), strides=1, activation='relu', padding='same', input_shape=(32,32,3)))
model.add(MaxPooling2D((2, 2)))

# Segunda capa de convolución
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

# Tercera capa de convolución
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(10, activation='softmax'))
```

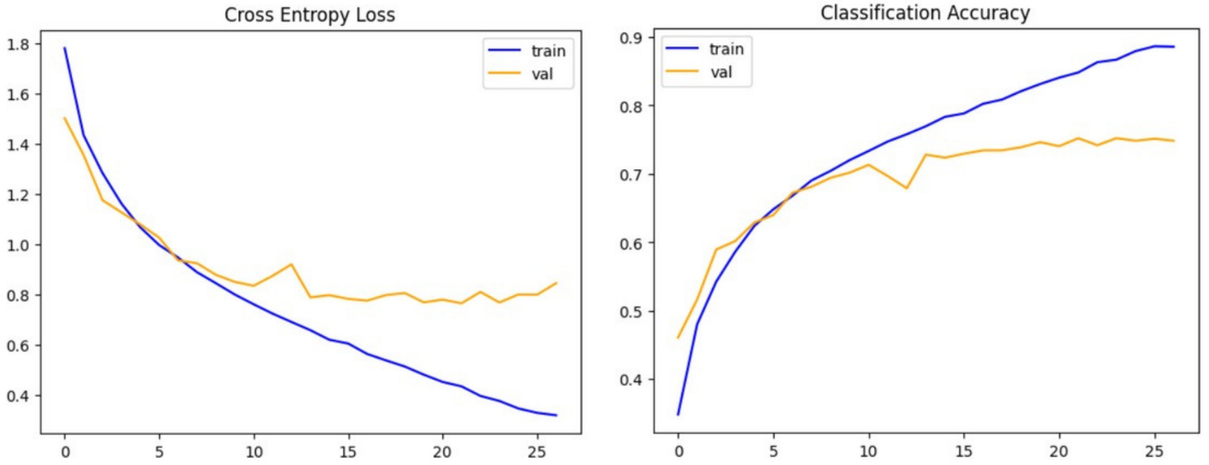
Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d_10 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_11 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_11 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_12 (Conv2D)	(None, 8, 8, 128)	73856
max_pooling2d_12 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten_8 (Flatten)	(None, 2048)	0
dense_30 (Dense)	(None, 500)	1024500
dropout_22 (Dropout)	(None, 500)	0
dense_31 (Dense)	(None, 10)	5010

```
Total params: 1122758 (4.28 MB)
Trainable params: 1122758 (4.28 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
history = model.fit(x_train_scaled, y_train,
                    callbacks=[modelchk, callback_loss, callback_acc],
                    epochs=300, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

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

```
> 73.960
```



Conclusiones y observaciones:

Con la intención de mejorar el modelo añadimos 2 capas de convulciones, lo que conlleva una mejora innegable del accuracy, pero un mayor problema de overffiting.

Proyecto III

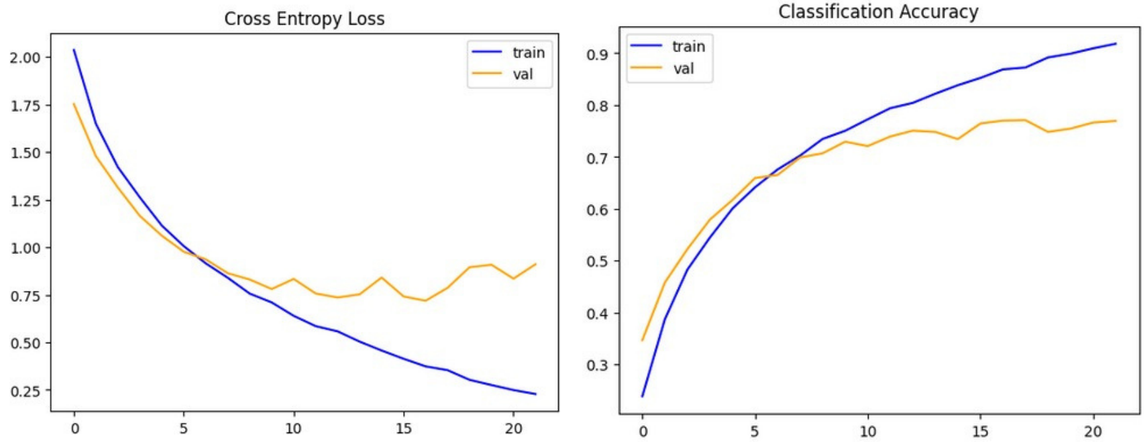
```
model = Sequential()
model.add(Conv2D(32, (3, 3), strides=1, activation='relu', padding='same', input_shape=(32,32,3)))
model.add(Conv2D(32, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(32, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

```
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896

conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248

conv2d_2 (Conv2D)            (None, 32, 32, 32)        9248

max_pooling2d (MaxPooling2   (None, 16, 16, 32)        0
D)

conv2d_3 (Conv2D)            (None, 16, 16, 64)        18496

conv2d_4 (Conv2D)            (None, 16, 16, 64)        36928

conv2d_5 (Conv2D)            (None, 16, 16, 64)        36928

max_pooling2d_1 (MaxPoolin   (None, 8, 8, 64)          0
g2D)

conv2d_6 (Conv2D)            (None, 8, 8, 128)         73856

conv2d_7 (Conv2D)            (None, 8, 8, 128)         147584

conv2d_8 (Conv2D)            (None, 8, 8, 128)         147584

max_pooling2d_2 (MaxPoolin   (None, 4, 4, 128)         0
g2D)

flatten (Flatten)            (None, 2048)              0

dense (Dense)                (None, 500)               1024500

dropout (Dropout)            (None, 500)               0

dense_1 (Dense)              (None, 10)                5010

=================================================================
Total params: 1510278 (5.76 MB)
Trainable params: 1510278 (5.76 MB)
```

```
history = model.fit(x_train_scaled, y_train,
                    callbacks=[modelchk, callback_loss, callback_acc],
                    epochs=300, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

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

```
> 76.470
```



Conclusiones y observaciones:

En este caso decido probar aumentar las convulciones d para ver como afecta al modelo.

Aunque como podemos ver seguimos teniendo problemas de overfitting que intentaremos mejorar en el siguiente proyecto.

Proyecto IV

```
model = Sequential()
model.add(Conv2D(32, (3, 3), strides=1, activation='relu', padding='same', input_shape=(32,32,3)))
model.add(Dropout(0.4))
model.add(Conv2D(32, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Dropout(0.4))
model.add(Conv2D(32, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Dropout(0.4))
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Dropout(0.4))
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Dropout(0.4))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Dropout(0.4))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

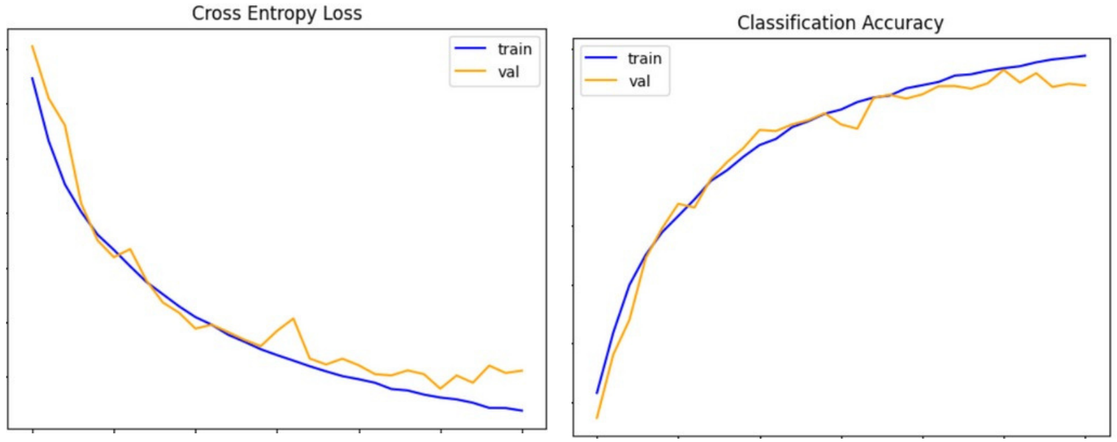
conv2d_9 (Conv2D)	(None, 32, 32, 32)	896
dropout_10 (Dropout)	(None, 32, 32, 32)	0
conv2d_10 (Conv2D)	(None, 32, 32, 32)	9248
dropout_11 (Dropout)	(None, 32, 32, 32)	0
conv2d_11 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_12 (Conv2D)	(None, 16, 16, 64)	18496
dropout_12 (Dropout)	(None, 16, 16, 64)	0
conv2d_13 (Conv2D)	(None, 16, 16, 64)	36928
dropout_13 (Dropout)	(None, 16, 16, 64)	0
conv2d_14 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_15 (Conv2D)	(None, 8, 8, 128)	73856
dropout_14 (Dropout)	(None, 8, 8, 128)	0
conv2d_16 (Conv2D)	(None, 8, 8, 128)	147584
dropout_15 (Dropout)	(None, 8, 8, 128)	0
conv2d_17 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_2 (Dense)	(None, 500)	1024500
dropout_16 (Dropout)	(None, 500)	0
dense_3 (Dense)	(None, 10)	5010

Total params: 1510278 (5.76 MB)
Trainable params: 1510278 (5.76 MB)

```
history = model.fit(x_train_scaled, y_train,
                    callbacks=[modelchk, callback_loss, callback_acc],
                    epochs=300, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

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

```
> 73.480
```



Conclusiones y observaciones:

Al añadir dropouts de 0.4 parece que ha mejorado el overfitting, pero he dificultado demasiado el aprendizaje del modelo, pasando de un 76 a un 73. Veamos si se puede mejorar.

Proyecto V

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

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

model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

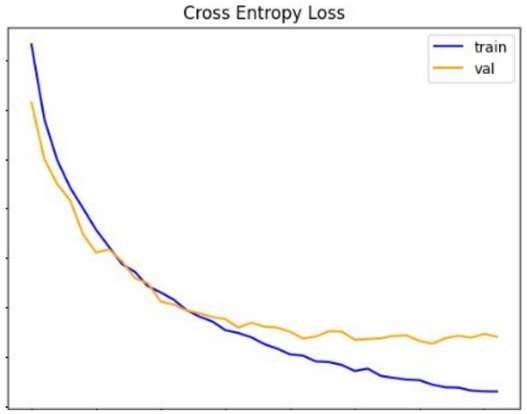
conv2d (Conv2D)	(None, 32, 32, 32)	896
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
conv2d_2 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_3 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_4 (Conv2D)	(None, 16, 16, 64)	36928
conv2d_5 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_6 (Conv2D)	(None, 8, 8, 128)	73856
conv2d_7 (Conv2D)	(None, 8, 8, 128)	147584
conv2d_8 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 500)	1024500
dropout_3 (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 10)	5010

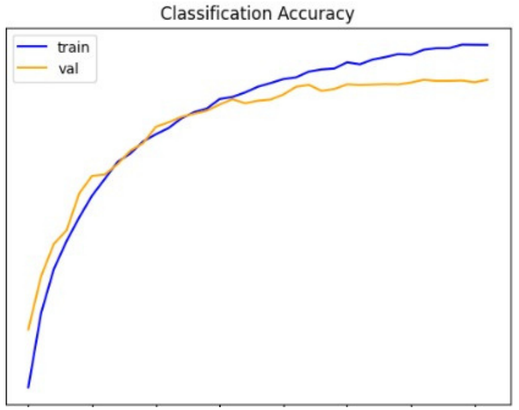
Total params: 1510278 (5.76 MB)
Trainable params: 1510278 (5.76 MB)

```
history = model.fit(x_train_scaled, y_train,
                    callbacks=[modelchk, callback_loss, callback_acc],
                    epochs=300, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

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

```
> 80.380
```





Conclusiones y observaciones:

En este caso he decidio dejar a 0.3 los dropouts y al mismo tiempo disminuir la cantidad de estos, ha resultado mejorar tanto el overfitting como el accuracy. Hemos llegado por primera vez a 80.

Proyecto VI

```
model = Sequential()
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same', input_shape=(32, 32, 3)))
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

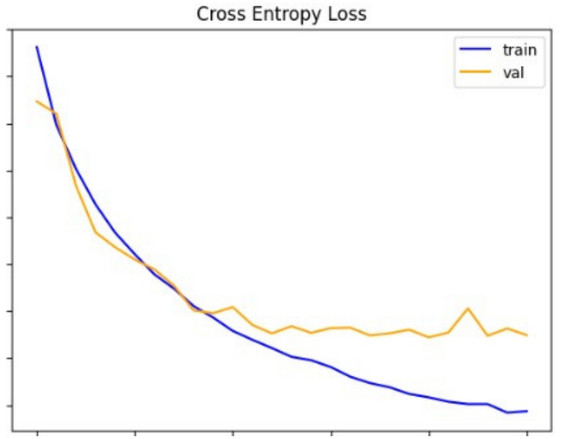
conv2d (Conv2D)	(None, 32, 32, 64)	1792
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36928
conv2d_2 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_3 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_4 (Conv2D)	(None, 16, 16, 128)	147584
conv2d_5 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_1 (Dropout)	(None, 8, 8, 128)	0
conv2d_6 (Conv2D)	(None, 8, 8, 256)	295168
conv2d_7 (Conv2D)	(None, 8, 8, 256)	590080
conv2d_8 (Conv2D)	(None, 8, 8, 256)	590080
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 1024)	4195328
dropout_3 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 10)	10250

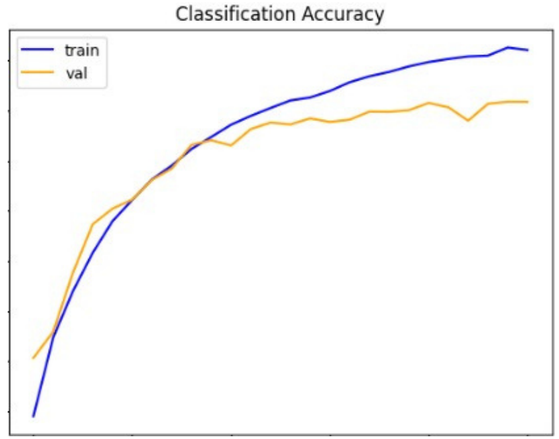
Total params: 6125578 (23.37 MB)
Trainable params: 6125578 (23.37 MB)

```
history = model.fit(x_train_scaled, y_train,
                    callbacks=[modelchk, callback_loss, callback_acc],
                    epochs=300, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

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

```
> 80.760
```





Conclusiones y observaciones:

Aumentamos los filtros para ver como afectan al modelo. El accuracy apenas ha mejorado, pero hay más overfitting y el tiempo de entrenamiento a aumentado.

Aunque es cierto que no mucho(de 0:04:24 a 0:06:49), no considero que una mejora tan pequeña merezca la pena.

Proyecto VII

```
model = Sequential()
model.add(Conv2D(32, (3, 3), strides=1, padding='same', input_shape=(32, 32, 3)))
model.add(Conv2D(32, (3, 3), strides=1, padding='same'))
model.add(Conv2D(32, (3, 3), strides=1, padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3, 3), strides=1, padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), strides=1, padding='same'))
model.add(Conv2D(64, (3, 3), strides=1, padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

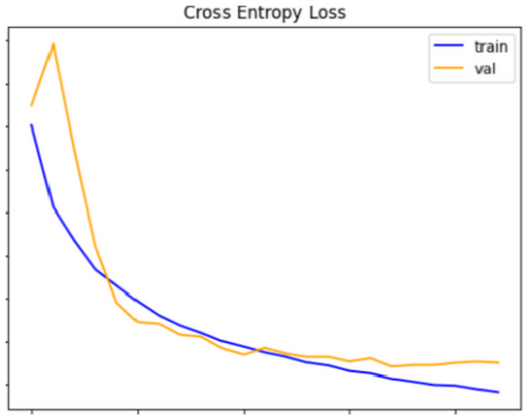
conv2d_27 (Conv2D)	(None, 32, 32, 32)	896
conv2d_28 (Conv2D)	(None, 32, 32, 32)	9248
conv2d_29 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_9 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_12 (Dropout)	(None, 16, 16, 32)	0
conv2d_30 (Conv2D)	(None, 16, 16, 64)	18496
batch_normalization_9 (BatchNormalization)	(None, 16, 16, 64)	256
conv2d_31 (Conv2D)	(None, 16, 16, 64)	36928
conv2d_32 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_10 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_13 (Dropout)	(None, 8, 8, 64)	0
conv2d_33 (Conv2D)	(None, 8, 8, 128)	73856
batch_normalization_10 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_34 (Conv2D)	(None, 8, 8, 128)	147584
conv2d_35 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_11 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_14 (Dropout)	(None, 4, 4, 128)	0
flatten_3 (Flatten)	(None, 2048)	0
dense_6 (Dense)	(None, 500)	1024500
dropout_15 (Dropout)	(None, 500)	0
dense_7 (Dense)	(None, 10)	5010

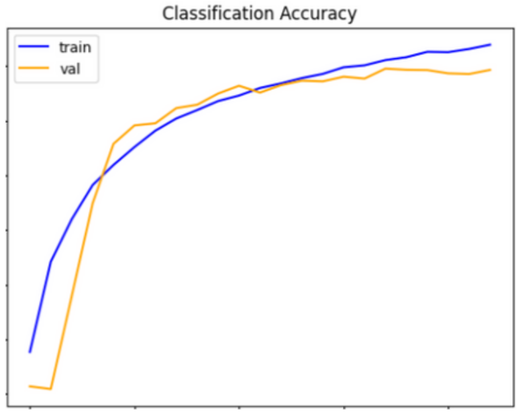
Total params: 1511046 (5.76 MB)
Trainable params: 1510662 (5.76 MB)

```
history = model.fit(x_train_scaled, y_train,
                    callbacks=[modelchk, callback_loss, callback_acc],
                    epochs=300, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

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

```
> 78.440
```





Conclusiones y observaciones:

Le añado 2 batchnormalization ya que no tenemos mucho overfitting y no queremos dificultar demasiado el aprendizaje del modelo.

Ha empeorado el modelo, probaremos con quitando uno ya que parece que el modelo aprende peor con 2.

Proyecto VIII

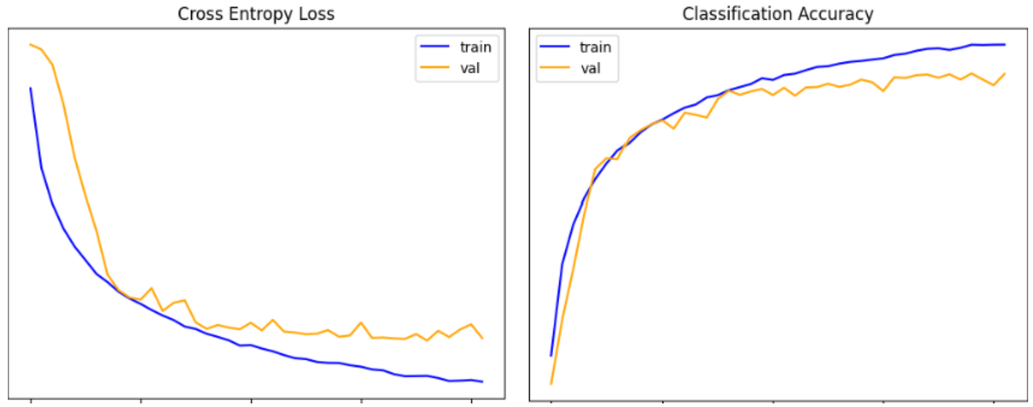
```
model = Sequential()
model.add(Conv2D(32, (3, 3), strides=1, activation='relu', padding='same', input_shape=(32,32,3)))
model.add(Conv2D(32, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(32, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2,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

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

model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

```
=================================================================
conv2d_18 (Conv2D)          (None, 32, 32, 32)        896
conv2d_19 (Conv2D)          (None, 32, 32, 32)        9248
conv2d_20 (Conv2D)          (None, 32, 32, 32)        9248
max_pooling2d_6 (MaxPoolin  (None, 16, 16, 32)        0
g2D)
dropout_8 (Dropout)         (None, 16, 16, 32)        0
conv2d_21 (Conv2D)          (None, 16, 16, 64)        18496
conv2d_22 (Conv2D)          (None, 16, 16, 64)        36928
conv2d_23 (Conv2D)          (None, 16, 16, 64)        36928
max_pooling2d_7 (MaxPoolin  (None, 8, 8, 64)          0
g2D)
dropout_9 (Dropout)         (None, 8, 8, 64)          0
conv2d_24 (Conv2D)          (None, 8, 8, 128)         73856
batch_normalization_3 (Bat  (None, 8, 8, 128)         512
chNormalization)
conv2d_25 (Conv2D)          (None, 8, 8, 128)         147584
conv2d_26 (Conv2D)          (None, 8, 8, 128)         147584
max_pooling2d_8 (MaxPoolin  (None, 4, 4, 128)         0
g2D)
dropout_10 (Dropout)        (None, 4, 4, 128)         0
flatten_2 (Flatten)         (None, 2048)              0
dense_4 (Dense)             (None, 500)               1024500
dropout_11 (Dropout)        (None, 500)               0
dense_5 (Dense)             (None, 10)                5010
=================================================================
Total params: 1510790 (5.76 MB)
Trainable params: 1510534 (5.76 MB)
```

```
history = model.fit(x_train_scaled, y_train,
                    callbacks=[modelchk, callback_loss, callback_acc],
                    epochs=300, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

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

```
> 81.600
```



Conclusiones y observaciones:

Le añado solamnte un batchnormalization para no dificultar demasiado el aprendizaje.

Hasta ahora es el mejor resultado, su aprendizaje no se ha visto afectado en este caso.

Proyecto IX

```
model = Sequential()
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same', input_shape=(32, 32, 3)))
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

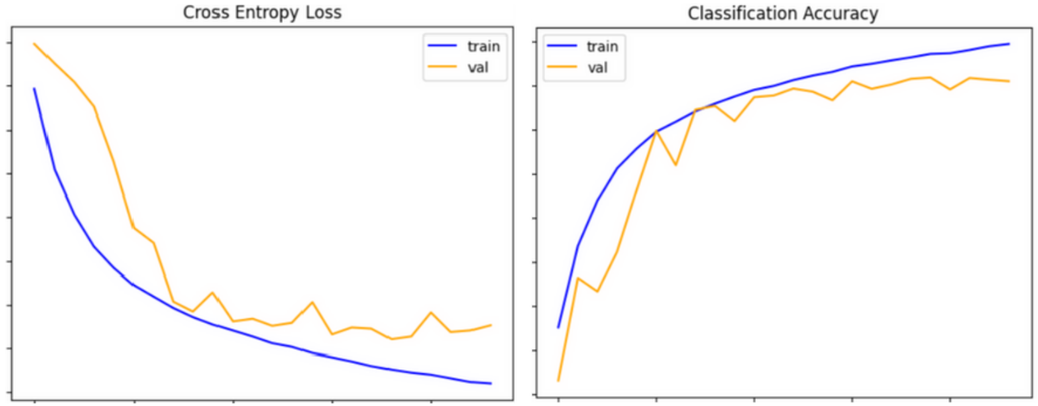
conv2d_18 (Conv2D)	(None, 32, 32, 64)	1792
conv2d_19 (Conv2D)	(None, 32, 32, 64)	36928
conv2d_20 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d_6 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_8 (Dropout)	(None, 16, 16, 64)	0
conv2d_21 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_22 (Conv2D)	(None, 16, 16, 128)	147584
conv2d_23 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_7 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_9 (Dropout)	(None, 8, 8, 128)	0
conv2d_24 (Conv2D)	(None, 8, 8, 256)	295168
batch_normalization_3 (BatchNormalization)	(None, 8, 8, 256)	1024
conv2d_25 (Conv2D)	(None, 8, 8, 256)	590080
conv2d_26 (Conv2D)	(None, 8, 8, 256)	590080
max_pooling2d_8 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_10 (Dropout)	(None, 4, 4, 256)	0
flatten_2 (Flatten)	(None, 4096)	0
dense_4 (Dense)	(None, 1024)	4195328
dropout_11 (Dropout)	(None, 1024)	0
dense_5 (Dense)	(None, 10)	10250

Total params: 6126602 (23.37 MB)
Trainable params: 6126090 (23.37 MB)

```
history = model.fit(x_train_scaled, y_train,
                    callbacks=[modelchk, callback_loss, callback_acc],
                    epochs=300, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

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

```
> 80.700
```



Conclusiones y observaciones:

Por último, antes de pasar a técnicas mas avanzadas, probaremos como afecta el batchnormalization al notebook con aumento de filtros(proyecto VIII), ya que en ese caso necesitabamos mejorar el overfitting.

Ni el accuracy ni el overfitting mejoran, de momento el proyecto ganador es el X.

Proyecto X - DG

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

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

model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

conv2d_7 (Conv2D)	(None, 32, 32, 32)	896
conv2d_8 (Conv2D)	(None, 32, 32, 32)	9248
conv2d_9 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_2 (Dropout)	(None, 16, 16, 32)	0
conv2d_10 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_11 (Conv2D)	(None, 16, 16, 64)	36928
conv2d_12 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_3 (Dropout)	(None, 8, 8, 64)	0
conv2d_13 (Conv2D)	(None, 8, 8, 128)	73856
batch_normalization (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_14 (Conv2D)	(None, 8, 8, 128)	147584
conv2d_15 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_4 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_4 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 500)	1024500
dropout_5 (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 10)	5010

Total params: 1510790 (5.76 MB)
Trainable params: 1510534 (5.76 MB)

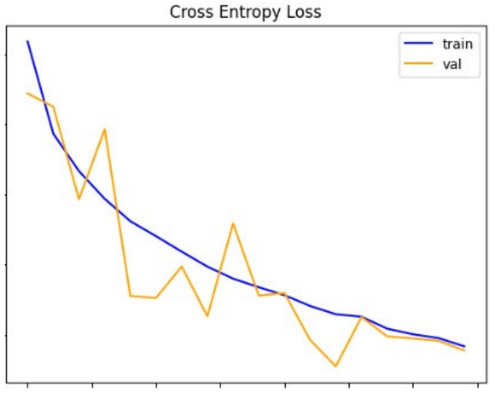
```
train_datagen = ImageDataGenerator(
    rescale=1./255, rotation_range = 15, horizontal_flip = True,
    width_shift_range=0.12, height_shift_range=0.12)

train_generator = train_datagen.flow(
    x_train, y_train, batch_size=30)
```

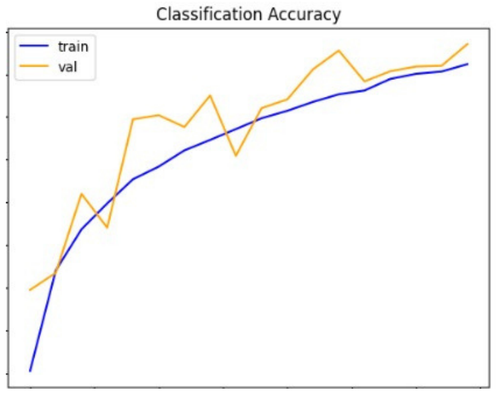
```
validation_datagen = ImageDataGenerator(
    rescale=1./255)
validation_generator = validation_datagen.flow(
    x_val, y_val, batch_size=20)
```

```
history = model.fit(train_generator, epochs=300, validation_data = validation_generator,
                    batch_size = 256, callbacks=[modelchk, callback_loss, callback_acc])
```

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

```
> 68.750
```





Conclusiones y observaciones:

Empezamos utilizando la técnica de data aumentation. En este primer intento el accuracy empeora drasticamente.

Vamos a intentar controlar el overffing, parece que este modelo no funciona muy bien con los nuevos datos

Proyecto XI - DG

```
model = Sequential()
# Primera capa de convolución
model.add(Conv2D(32, (3, 3), strides=1, activation='relu', padding='same', input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(32, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))
# Segunda capa de convolución
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))
# Tercera capa de convolución
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
conv2d_2 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_3 (Conv2D)	(None, 16, 16, 64)	18496
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 64)	256
conv2d_4 (Conv2D)	(None, 16, 16, 64)	36928
conv2d_5 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_6 (Conv2D)	(None, 8, 8, 128)	73856
batch_normalization_2 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_7 (Conv2D)	(None, 8, 8, 128)	147584
conv2d_8 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 500)	1024500
dropout_3 (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 10)	5010

```
Total params: 1511174 (5.76 MB)
Trainable params: 1510726 (5.76 MB)
Non-trainable params: 448 (1.75 KB)
```

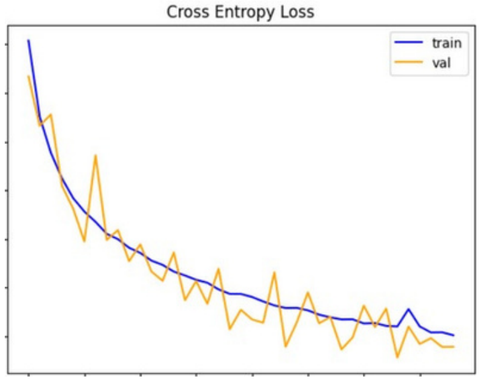
```
train_datagen = ImageDataGenerator(
    rescale=1./255, rotation_range = 15, horizontal_flip = True,
    width_shift_range=0.12, height_shift_range=0.12)

train_generator = train_datagen.flow(
    x_train, y_train, batch_size=30)
```

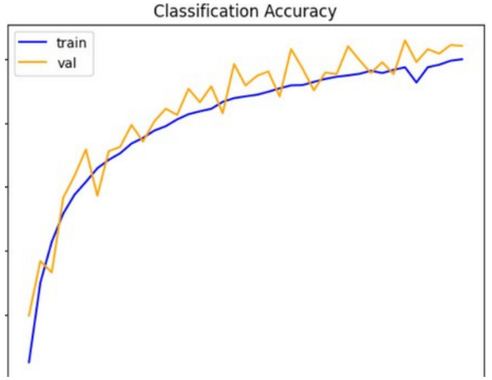
```
validation_datagen = ImageDataGenerator(
    rescale=1./255)
validation_generator = validation_datagen.flow(
    x_val, y_val, batch_size=20)
```

```
history = model.fit(train_generator, epochs=300, validation_data = validation_generator,
                    batch_size = 256, callbacks=[modelchk, callback_loss, callback_acc])
```

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

```
> 82.250
```





Conclusiones y observaciones:

Añadimos batchnormalization. El resultado es un accuracy de 82, el mejor modelo hasta ahora.

Aunque el segundo mejor modelo era de 81, y tardaba 4min aproximadamente en cargar, este en cambio ha tardado 24min por lo que no merece la pena una mejora tan pequeña.

Proyecto XII - TL

```
model_vgg16 = vgg16.VGG16(include_top = False, weights = 'imagenet',
                          input_shape = (32,32,3))
```

```
model_post_vgg = ks.Sequential()

model_post_vgg.add(final_vgg16)
model_post_vgg.add(ks.layers.Dense(500, activation='relu'))
model_post_vgg.add(ks.layers.Dropout(0.5))
model_post_vgg.add(ks.layers.Dense(10, activation='softmax'))

model_post_vgg.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 model (Functional)          (None, 512)               14714688

 dense (Dense)               (None, 500)               256500

 dropout (Dropout)           (None, 500)               0

 dense_1 (Dense)             (None, 10)                5010

=================================================================
Total params: 14976198 (57.13 MB)
Trainable params: 14716038 (56.14 MB)
Non-trainable params: 260160 (1016.25 KB)
_________________________________________________________________
```

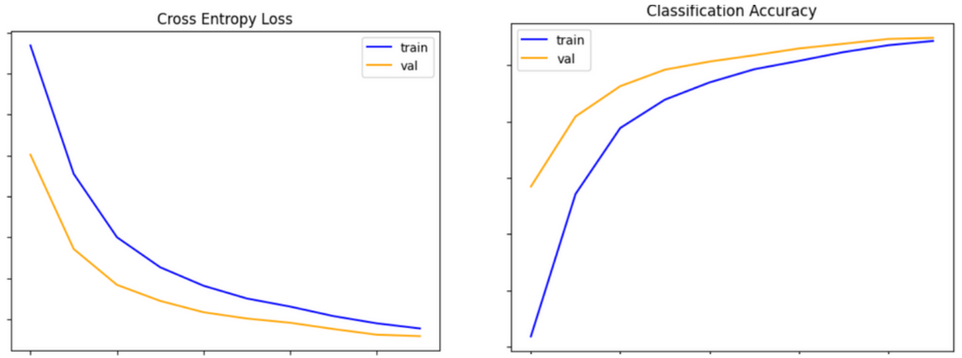
```
new_adam = Adam(learning_rate=0.000005)
model_post_vgg.compile(loss='sparse_categorical_crossentropy',
                       optimizer=new_adam, metrics=['accuracy'])
```

```
history = model_post_vgg.fit(x=x_train_scaled, y=y_train, batch_size=512,
                             epochs=10, callbacks=[callback],
                             validation_data=(x_val_scaled, y_val))
```

```
Tiempo de entrenamiento: 0:00:00.007997
```

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

```
> 74.190
```



Conclusiones y observaciones:

He implementado tranfer learning con el modelo vgg16, además he cambiado el learning rate.

El accuracy ha empeorado, pero el tiempo de entrenamiento ha disminuido muchisimo, por lo que voy a intentar mejorar el modelo a partir de aquí.

Proyecto XIII - TL + WD

```
model_vgg16 = vgg16.VGG16(include_top = False, weights = 'imagenet',
                          input_shape = (32,32,3))
```

```
model_post_vgg = ks.Sequential()

model_post_vgg.add(final_vgg16)
model_post_vgg.add(Dense(500, activation='relu', kernel_regularizer=l2(0.00001)))
model_post_vgg.add(Dropout(0.5))
model_post_vgg.add(Dense(10, activation='softmax'))
```

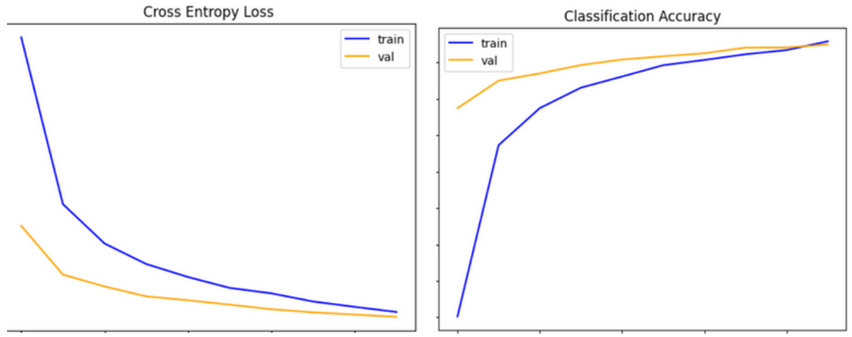
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 model (Functional)          (None, 512)               14714688

 dense_2 (Dense)             (None, 500)               256500

 dropout_1 (Dropout)         (None, 500)               0

 dense_3 (Dense)             (None, 10)                5010

=================================================================
Total params: 14976198 (57.13 MB)
Trainable params: 14716038 (56.14 MB)
Non-trainable params: 260160 (1016.25 KB)
```

```
new_adam = Adam(learning_rate=0.000005)
model_post_vgg.compile(loss='sparse_categorical_crossentropy',
                       optimizer=new_adam, metrics=['accuracy'])
```

```
history = model_post_vgg.fit(x=x_train_scaled, y=y_train, batch_size=512,
                             epochs=10, callbacks=[callback],
                             validation_data=(x_val_scaled, y_val))
```

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

```
> 77.500
```



Conclusiones y observaciones:

He añadido el l2 con la intención de controlar el overfitting

Ahora toca cambiar el optimizador ya que Adam no funciona bien con esta técnica.

Proyecto XIV - TL + WD

```
model_vgg16 = vgg16.VGG16(include_top = False, weights = 'imagenet',
                          input_shape = (32,32,3))
```

```
model_post_vgg = ks.Sequential()

model_post_vgg.add(final_vgg16)
model_post_vgg.add(Dense(500, activation='relu', kernel_regularizer=l2(0.00001)))
model_post_vgg.add(Dropout(0.5))
model_post_vgg.add(Dense(10, activation='softmax'))
```

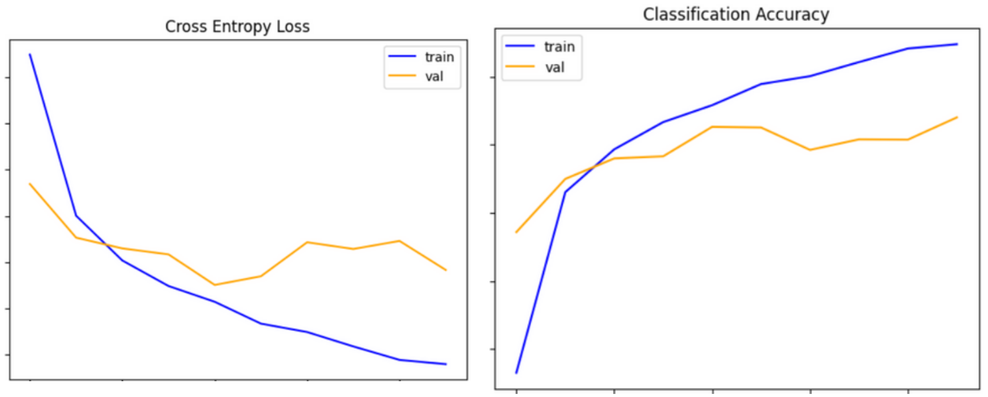
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 model (Functional)          (None, 512)               14714688

 dense_2 (Dense)             (None, 500)               256500

 dropout_1 (Dropout)         (None, 500)               0

 dense_3 (Dense)             (None, 10)                5010

=================================================================
Total params: 14976198 (57.13 MB)
Trainable params: 14716038 (56.14 MB)
Non-trainable params: 260160 (1016.25 KB)
```

```
new_sgd = SGD(learning_rate=0.01, momentum=0.9)
model_post_vgg.compile(loss='sparse_categorical_crossentropy',
                       optimizer=new_sgd, metrics=['accuracy'])
```

```
history = model_post_vgg.fit(x=x_train_scaled, y=y_train, batch_size=512,
                             epochs=10, callbacks=[callback],
                             validation_data=(x_val_scaled, y_val))
```

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

```
> 83.380
```



Conclusiones y observaciones:

Cambiamos el optimizador a SGD lo cual mejora el modelo.

Ahora queremos controlar el overfitting y para ello probaré con data generation, ya que parece que el modelo es muy complejo y tenemos pocos datos.

Proyecto XV - TL + DG

```
model_vgg16 = vgg16.VGG16(include_top = False, weights = 'imagenet',
                          input_shape = (32,32,3))
```

```
model_post_vgg = ks.Sequential()

model_post_vgg.add(final_vgg16)
model_post_vgg.add(Dense(500, activation='relu', kernel_regularizer=l2(0.00001)))
model_post_vgg.add(Dropout(0.5))
model_post_vgg.add(Dense(10, activation='softmax'))
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 model (Functional)          (None, 512)               14714688

 dense_2 (Dense)             (None, 500)               256500

 dropout_1 (Dropout)         (None, 500)               0

 dense_3 (Dense)             (None, 10)                5010

=================================================================
Total params: 14976198 (57.13 MB)
Trainable params: 14716038 (56.14 MB)
Non-trainable params: 260160 (1016.25 KB)
```

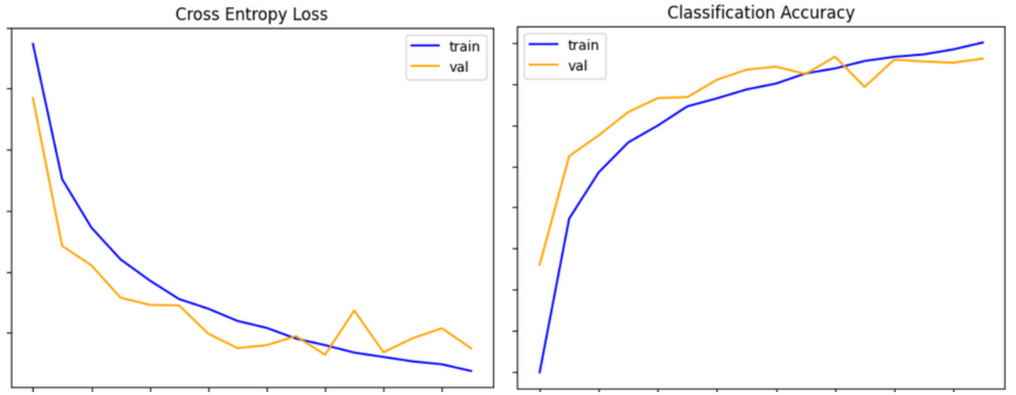
```
new_sgd = SGD(learning_rate=0.01, momentum=0.9)
model_post_vgg.compile(loss='sparse_categorical_crossentropy',
                       optimizer=new_sgd, metrics=['accuracy'])
```

```
train_datagen = ImageDataGenerator(rescale=1./255,
    rotation_range = 15, horizontal_flip = True,
    width_shift_range=0.12,height_shift_range=0.12)

train_generator = train_datagen.flow(
    x_train, y_train,batch_size=30)
```

```
history = model_post_vgg.fit(train_generator, epochs=300,
                             validation_data = validation_generator,
                             batch_size = 256, callbacks=[callback])
```

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

```
> 82.720
```



Conclusiones y observaciones:

He aumentado también los epochs para scar el máximo partido al modelos.

Parece que la teoria se ha confirmado, aumentar los datos ha controlado el overfitting.

Conclusión

Proyecto	Procedimiento	Accuracy	Train time
P.1	Aumento de neuronas	65,86	0:00:43
P.2	Aumento de capas	73,96	0:00:48
P.3	Aumento de convulciones	76,47	0:02:19
P.4	Dropouts 0.4	73,48	0:04:03
P.5	Dropouts 0.3	80,38	0:04:24
P.6	Cambiar los filtros	80,76	0:06:49
P.7	BatchNormalization v.1	78,44	0:02:59
P.8	BatchNormalization v.2	81,60	0:04:57
P.9	Filtros + BN	80,70	0:05:59
P.10	Datageneration	68,75	0:11:16
P.11	Datagen. + control overfitting	82,25	0:24:29
P.12	Transfer learning	74,19	0:00:00
P.13	T. learning + Weight Decay	77,50	0:00:00
P.14	Cambio optimizador	83,38	0:00:00
P.15	T. learning + D. Generation	82,72	0:00:00

Conclusión

Aunque hemos tenido varios modelos con un buen accuracy, nos hemos encontrado con 2 problemas principales, el overfitting y el tiempo de entrenamiento.

Al llegar a tranfer learnign he podido apreciar lo eficaz que es para disminuir el segundo problema (el tiempo) por lo que he decido partir de ahí mejorando el modelo, y al final el overfitting.

Sin duda podemos concluir que el poryecto 15 con un acc de 82,72 (el segundo más alto) 0 seg de tiempo y un overfitting controlado es el ganador (por encima del 14 que aunque tenía mejor accuracy también tenía problemas de sobreajuste).