# PROYECTO VISIÓN POR COMPUTACIÓN
# CIFAR 10

Nicolás Goldschwartz

# VERSIÓN 1

```
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.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))


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



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

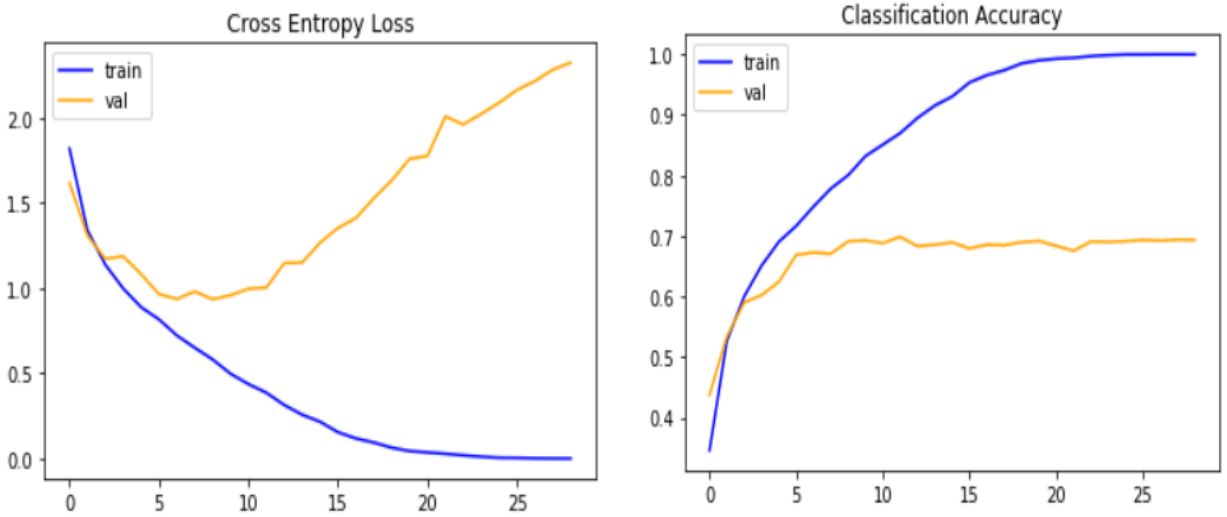
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 32, 32, 32)        896

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 16, 16, 128)       36992

 conv2d_5 (Conv2D)           (None, 16, 16, 256)       295168

 flatten_1 (Flatten)         (None, 65536)             0

 dense_2 (Dense)             (None, 32)                2097184

 dense_3 (Dense)             (None, 10)                330

=================================================================
Total params: 2,430,570
Trainable params: 2,430,570
Non-trainable params: 0
_________________________________________________________________
```

```
[34] history = model.fit(x_train_scaled, y_train, epochs=30,
                         callbacks=[model_checkpoint_callback, earlystopping_loss_callback, earlystopping_acc_callback],
                         batch_size= 512, validation_data=(x_val_scaled, y_val))
```

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

```
> 68.860
```




Partimos desde el proyecto 0 con un modelo con una accuracy de 56.730. Para incrementar la precisión vamos a empezar por añadir a la arquitectura dos capas de convoluciones, una de 128 neuronas y otra de 256 para aumentar el barrido de las imágenes. El resultado ha sido muy satisfactorio , la accuracy ha aumentado más de 10 puntos hasta situarse en 68.860.

# VERSIÓN 2

```
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.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))

model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))

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



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

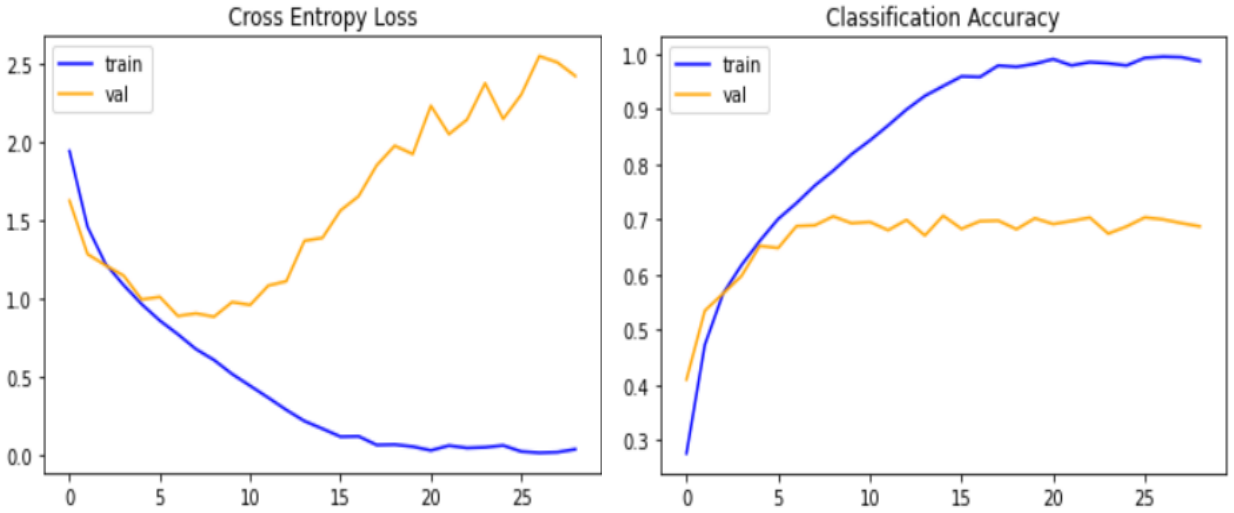
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 32, 32, 32)        896

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_7 (Conv2D)           (None, 16, 16, 128)       36992

 conv2d_8 (Conv2D)           (None, 16, 16, 256)       295168

 conv2d_9 (Conv2D)           (None, 16, 16, 128)       295040

 conv2d_10 (Conv2D)          (None, 16, 16, 256)       295168

 flatten_2 (Flatten)         (None, 65536)             0

 dense_4 (Dense)             (None, 32)                2097184

 dense_5 (Dense)             (None, 10)                330

=================================================================
Total params: 3,020,778
Trainable params: 3,020,778
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(x_train_scaled, y_train, epochs=30,
                    callbacks=[model_checkpoint_callback, earlystopping_loss_callback, earlystopping_acc_callback],
                    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))
```

```
> 68.430
```




Para esta segunda versión y siguiendo la línea de los resultados anteriores decidimos añadir otro paquete de dos convoluciones iguales que el anterior de 128 y 256 neuronas respectivamente. La accuracy ha disminuido un poco en vez de aumentar, resultando en un valor de 68.430. Es posible que el no incremento de accuracy se deba al hecho de no haber introducido ninguna capa maxpooling después de las capas de convolución. Las gráficas de función de error y de precisión no muestran bondad del modelo, sus líneas de entrenamiento y validación están muy distanciadas entre ellas en ambas gráficas.

# VERSIÓN 3

```
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.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
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'))
```

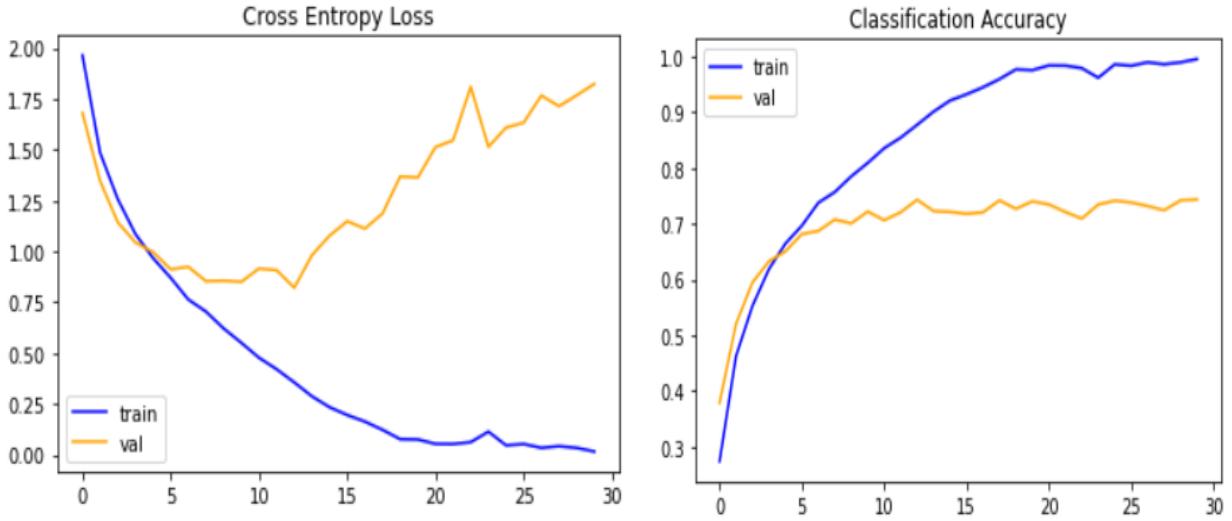
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_11 (Conv2D)          (None, 32, 32, 32)        896

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_12 (Conv2D)          (None, 16, 16, 128)       36992

 conv2d_13 (Conv2D)          (None, 16, 16, 256)       295168

 max_pooling2d_4 (MaxPooling  (None, 8, 8, 256)        0
 2D)

 conv2d_14 (Conv2D)          (None, 8, 8, 128)         295040

 conv2d_15 (Conv2D)          (None, 8, 8, 256)         295168

 max_pooling2d_5 (MaxPooling  (None, 4, 4, 256)        0
 2D)

 flatten_3 (Flatten)         (None, 4096)              0

 dense_6 (Dense)             (None, 32)                131104

 dense_7 (Dense)             (None, 10)                330

=================================================================
Total params: 1,054,698
Trainable params: 1,054,698
Non-trainable params: 0
```

```
history = model.fit(x_train_scaled, y_train, epochs=30,
                    callbacks=[model_checkpoint_callback, earlystopping_loss_callback, earlystopping_acc_callback],
                    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
```




Para corregir el poco incremento de accuracy de la anterior arquitectura vamos a introducir dos capas maxpooling, una después de cada paquete de convoluciones con la idea de reducir la dimensión espacial, es decir el ancho por alto de la imagen del volumen de entrada para la siguiente capa. El resultado ha dado lugar a un incremento de la accuracy de unos 5 puntos llegando a 73.960. Las gráficas de error y precisión del modelo siguen igual que en la versión anterior.

# VERSIÓN 4

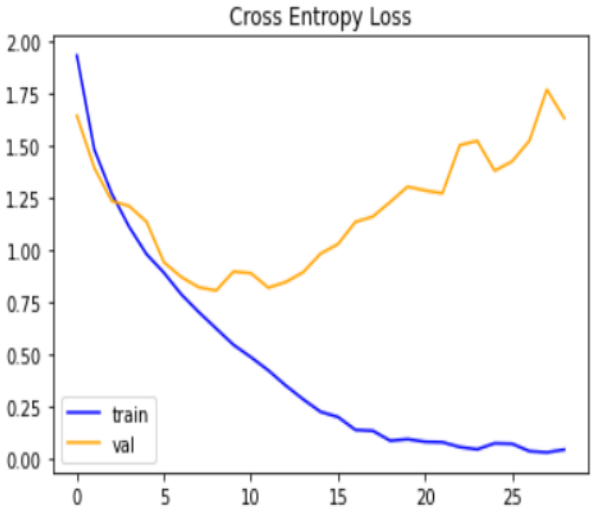
```
model = ks.Sequential()

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

model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
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(128, activation='relu'))
model.add(ks.layers.Dense(256, activation='relu'))



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

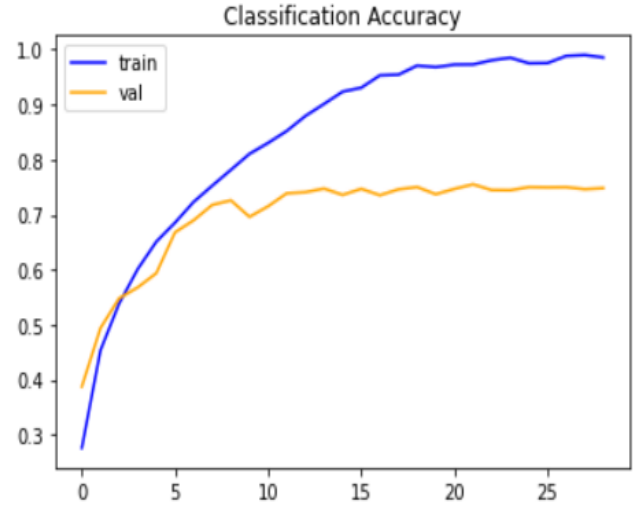
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (None, 32, 32, 32)        896

 max_pooling2d_6 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_17 (Conv2D)          (None, 16, 16, 128)       36992

 conv2d_18 (Conv2D)          (None, 16, 16, 256)       295168

 max_pooling2d_7 (MaxPooling  (None, 8, 8, 256)        0
 2D)

 conv2d_19 (Conv2D)          (None, 8, 8, 128)         295040

 conv2d_20 (Conv2D)          (None, 8, 8, 256)         295168

 max_pooling2d_8 (MaxPooling  (None, 4, 4, 256)        0
 2D)

 flatten_4 (Flatten)         (None, 4096)              0

 dense_8 (Dense)             (None, 32)                131104

 dense_9 (Dense)             (None, 128)               4224

 dense_10 (Dense)            (None, 256)               33024

 dense_11 (Dense)            (None, 10)                2570

=================================================================
Total params: 1,094,186
Trainable params: 1,094,186
Non-trainable params: 0
```

```
history = model.fit(x_train_scaled, y_train, epochs=30,
                    callbacks=[model_checkpoint_callback, earlystopping_loss_callback, earlystopping_acc_callback],
                    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))
```

```
> 73.540
```




Se introducen dos capas dense tras el flatten de 128 y 256 neuronas de entrada y salida respectivamente para aumentar las conexiones en la zona fully conected y obtener así una mejora en el área de clasificación. Finalmente la accuracy ha resultado un poco inferior a la versión anterior situándose en 73.540. Al parecer este incremento de capas tras el flatten no han sido de ayuda. Tampoco hay mejoras en las gráficas de error y precisión.

# VERSIÓN 5

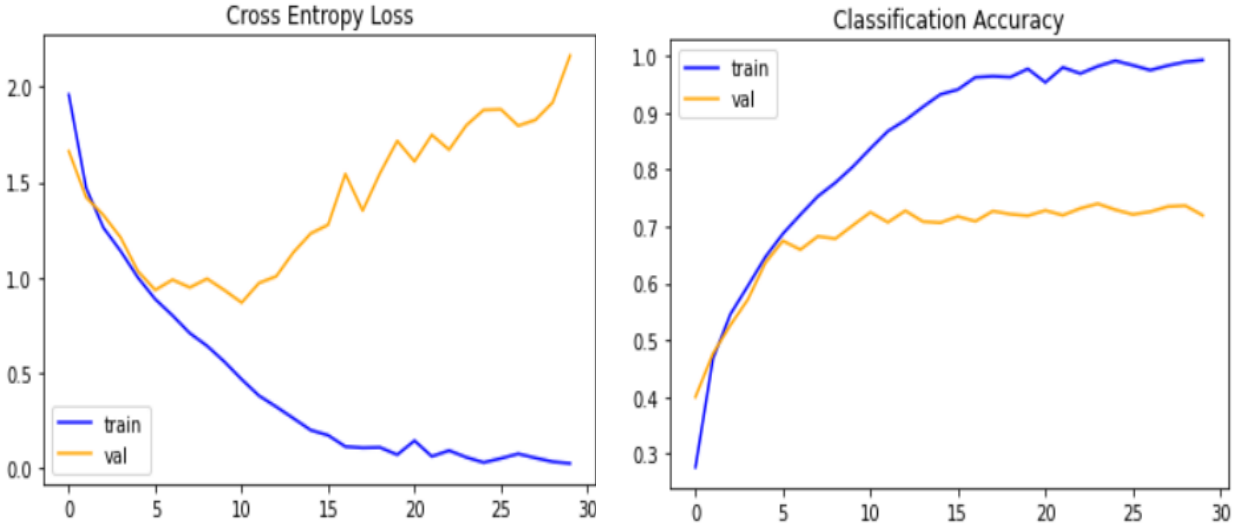
```
model = ks.Sequential()

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.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.Dense(128, activation='relu'))
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

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 16, 16, 128)       36992

 conv2d_2 (Conv2D)           (None, 16, 16, 256)       295168

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 256)        0
 2D)

 conv2d_3 (Conv2D)           (None, 8, 8, 128)         295040

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

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

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 256)               1048832

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 32)                4128

 dense_3 (Dense)             (None, 10)                330

=================================================================
Total params: 2,009,450
Trainable params: 2,009,450
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(x_train_scaled, y_train, epochs=30,
                    callbacks=[model_checkpoint_callback, earlystopping_loss_callback, earlystopping_acc_callback],
                    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))
```

```
> 71.580
```




Se realizan cambios en el orden de neuronas entre la capa flatten y la de softmax de modo que se pasa de orden creciente a decreciente, empezando con una capa de 256 otra de 128 y una final de 32 neuronas . Esta modificación ha arrojado una accuracy inferior a la de la versión anterior perdiendo dos puntos y quedando en 71.580. El overfitting y la validación de error siguen sin mejoras.

# VERSIÓN 6

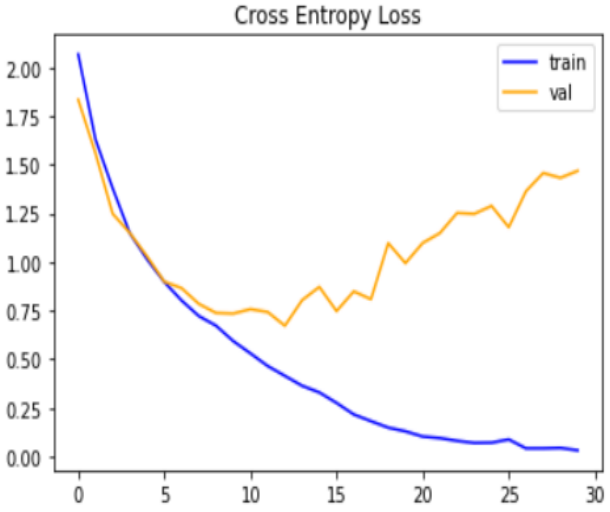
```
model = ks.Sequential()

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), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
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(64, activation='relu'))
model.add(ks.layers.Dense(128, activation='relu'))



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

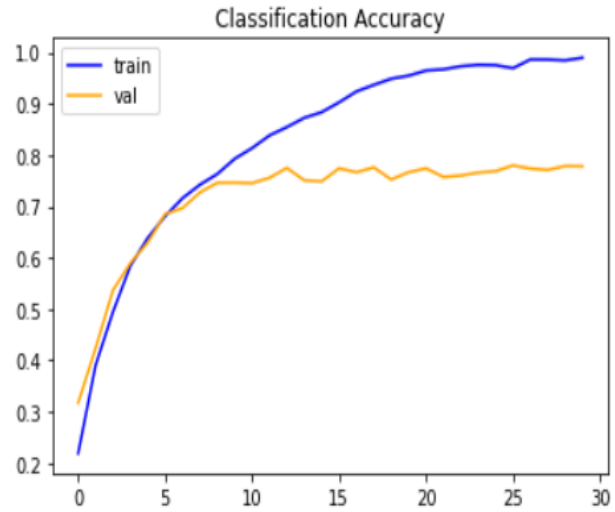
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)            (None, 32, 32, 32)        896

 conv2d_6 (Conv2D)            (None, 32, 32, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 64)        0
 2D)

 conv2d_7 (Conv2D)            (None, 16, 16, 128)       73856

 conv2d_8 (Conv2D)            (None, 16, 16, 256)       295168

 max_pooling2d_4 (MaxPooling  (None, 8, 8, 256)         0
 2D)

 conv2d_9 (Conv2D)            (None, 8, 8, 128)         295040

 conv2d_10 (Conv2D)           (None, 8, 8, 256)         295168

 max_pooling2d_5 (MaxPooling  (None, 4, 4, 256)         0
 2D)

 flatten_1 (Flatten)          (None, 4096)              0

 dense_4 (Dense)              (None, 32)                131104

 dense_5 (Dense)              (None, 64)                2112

 dense_6 (Dense)              (None, 128)               8320

 dense_7 (Dense)              (None, 10)                1290

=================================================================
Total params: 1,121,450
Trainable params: 1,121,450
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(x_train_scaled, y_train, epochs=30,
                    callbacks=[model_checkpoint_callback, earlystopping_loss_callback, earlystopping_acc_callback],
                    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))
```

```
> 77.310
```




Se introducen varias correcciones , en primer lugar se añade una capa de convolución de 64 neuronas al primer paquete, quedando finalmente tres capas de convolución seguidos de un maxpooling cada una en segundo lugar se modifica el número de neuronas y su orden entre el flatten y la última capa se establece un orden ascendente de 32, 64 y 128 neuronas, la idea es que mejore en la precisión en la clasificación de las imágenes .

El resultado ahora sí es notable , la accuracy ha mejorado sustancialmente hasta alcanzar los 77.310.

# VERSIÓN 7

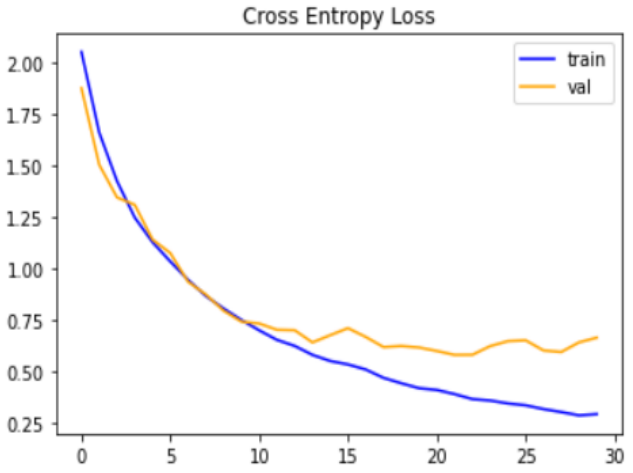
```
model = ks.Sequential()

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), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(32, activation='relu'))
model.add(ks.layers.Dense(64, activation='relu'))
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dense(512, activation='relu'))


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

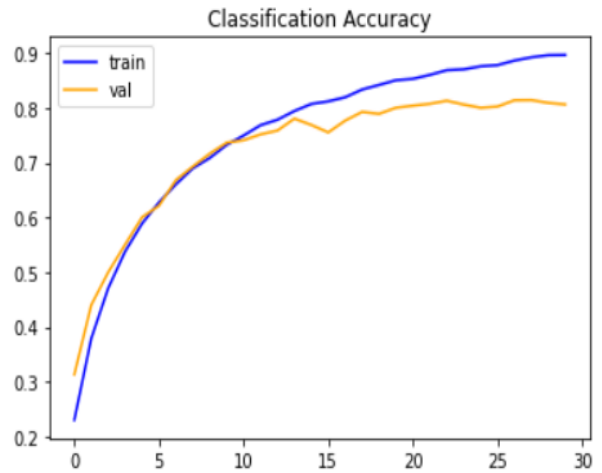
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_11 (Conv2D)          (None, 32, 32, 32)        896

 conv2d_12 (Conv2D)          (None, 32, 32, 64)        18496

 max_pooling2d_6 (MaxPooling  (None, 16, 16, 64)       0
 2D)

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

 conv2d_13 (Conv2D)          (None, 16, 16, 128)       73856

 conv2d_14 (Conv2D)          (None, 16, 16, 256)       295168

 max_pooling2d_7 (MaxPooling  (None, 8, 8, 256)        0
 2D)

 dropout_1 (Dropout)         (None, 8, 8, 256)         0

 conv2d_15 (Conv2D)          (None, 8, 8, 128)         295040

 conv2d_16 (Conv2D)          (None, 8, 8, 256)         295168

 max_pooling2d_8 (MaxPooling  (None, 4, 4, 256)        0
 2D)

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

 flatten_2 (Flatten)         (None, 4096)              0

 dense_8 (Dense)             (None, 32)                131104

 dense_9 (Dense)             (None, 64)                2112

 dense_10 (Dense)            (None, 128)               8320

 dense_11 (Dense)            (None, 512)               66048

 dense_12 (Dense)            (None, 10)                5130

=================================================================
Total params: 1,191,338
Trainable params: 1,191,338
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(x_train_scaled, y_train, epochs=30,
                    callbacks=[model_checkpoint_callback, earlystopping_loss_callback, earlystopping_acc_callback],
                    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))
```

```
> 80.100
```




En esta versión se introducen dos cambios principales uno para corregir el overfitting y otro para mejorar la precisión en la clasificación, para el primero se introducen tres capas de dropout con un rate de 0.4 tras el maxpooling de cada paquete de convoluciones. Para la segunda modificación se añade una capa dense de 514 neuronas en último paquete tras el flatten. Los cambios introducidos han aumentado la accuracy llegando a un 80.100 y los dropouts han mejorado notablemente el overfitting.

# VERSIÓN 8

```
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), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Conv2D(512, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(1024, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(32, activation='relu'))
model.add(ks.layers.Dense(64, activation='relu'))
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dense(512, activation='relu'))



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

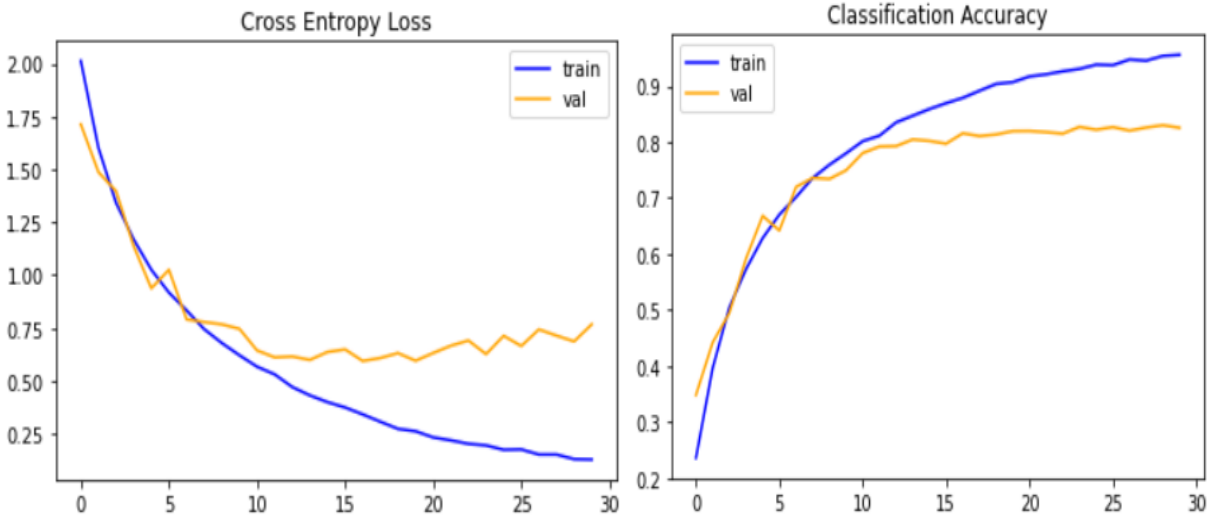
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_17 (Conv2D)          (None, 32, 32, 32)        896
 conv2d_18 (Conv2D)          (None, 32, 32, 64)        18496
 max_pooling2d_9 (MaxPooling  (None, 16, 16, 64)       0
 2D)
 dropout_3 (Dropout)         (None, 16, 16, 64)        0
 conv2d_19 (Conv2D)          (None, 16, 16, 128)       73856
 conv2d_20 (Conv2D)          (None, 16, 16, 256)       295168
 max_pooling2d_10 (MaxPoolin  (None, 8, 8, 256)        0
 g2D)
 dropout_4 (Dropout)         (None, 8, 8, 256)         0
 conv2d_21 (Conv2D)          (None, 8, 8, 512)         1180160
 conv2d_22 (Conv2D)          (None, 8, 8, 1024)        4719616
 max_pooling2d_11 (MaxPoolin  (None, 4, 4, 1024)       0
 g2D)
 dropout_5 (Dropout)         (None, 4, 4, 1024)        0
 flatten_3 (Flatten)         (None, 16384)             0
 dense_13 (Dense)            (None, 32)                524320
 dense_14 (Dense)            (None, 64)                2112
 dense_15 (Dense)            (None, 128)               8320
 dense_16 (Dense)            (None, 512)               66048
 dense_17 (Dense)            (None, 10)                5130
=================================================================
Total params: 6,894,122
Trainable params: 6,894,122
Non-trainable params: 0
```

```
history = model.fit(x_train_scaled, y_train, epochs=30,
                    callbacks=[model_checkpoint_callback, earlystopping_loss_callback, earlystopping_acc_callback],
                    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))
```

```
> 81.940
```




Se introducen más neuronas al último paquete de convoluciones incrementando en 512 y 1024 respectivamente, de manera que la arquitectura del modelo va incrementando el número de neuronas desde la primera a la última convolución, empezando en 32 y finalizando en 1024.
La accuracy ha mejorado casi dos puntos situándose en 81.940 pero el overfitting ha aumentado en relación a la versión anterior

# VERSIÓN 9

```
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), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Conv2D(512, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(1024, (3,3), padding='same', activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(32, activation='relu'))
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.Dense(512, activation='relu'))



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

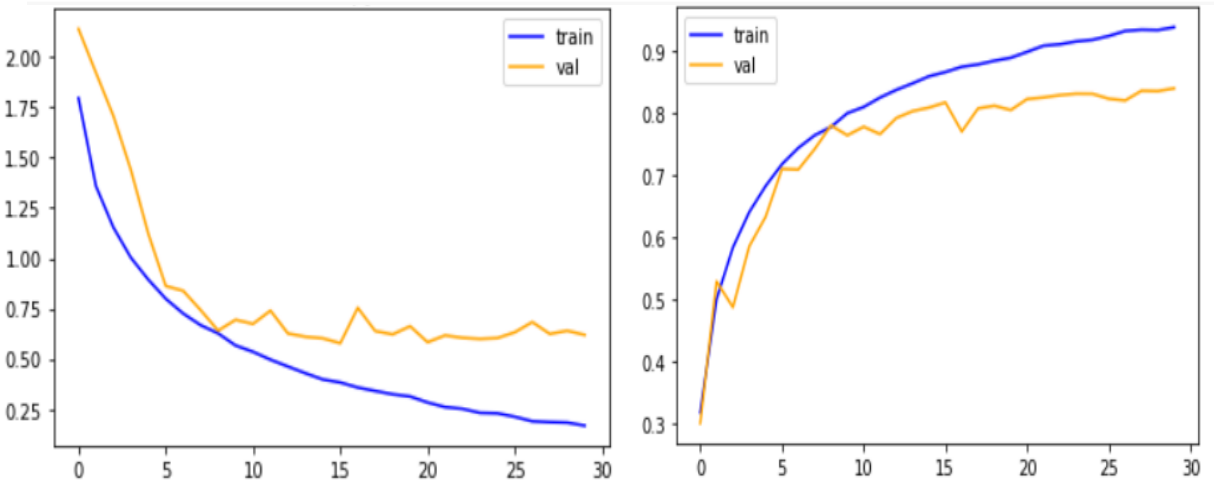
```
model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_72 (Conv2D)          (None, 32, 32, 32)        896
 conv2d_73 (Conv2D)          (None, 32, 32, 64)        18496
 max_pooling2d_36 (MaxPoolin  (None, 16, 16, 64)       0
 g2D)
 dropout_36 (Dropout)        (None, 16, 16, 64)        0
 conv2d_74 (Conv2D)          (None, 16, 16, 128)       73856
 conv2d_75 (Conv2D)          (None, 16, 16, 256)       295168
 max_pooling2d_37 (MaxPoolin  (None, 8, 8, 256)        0
 g2D)
 dropout_37 (Dropout)        (None, 8, 8, 256)         0
 conv2d_76 (Conv2D)          (None, 8, 8, 512)         1180160
 conv2d_77 (Conv2D)          (None, 8, 8, 1024)        4719616
 batch_normalization_16 (Bat  (None, 8, 8, 1024)       4096
 chNormalization)
 max_pooling2d_38 (MaxPoolin  (None, 4, 4, 1024)       0
 g2D)
 dropout_38 (Dropout)        (None, 4, 4, 1024)        0
 flatten_12 (Flatten)        (None, 16384)             0
 dense_60 (Dense)            (None, 32)                524320
 dense_61 (Dense)            (None, 128)               4224
 dense_62 (Dense)            (None, 256)               33024
 dense_63 (Dense)            (None, 512)               131584
 dense_64 (Dense)            (None, 10)                5130
=================================================================
Total params: 6,990,570
Trainable params: 6,988,522
Non-trainable params: 2,048
_________________________________________________________________
```

```
history = model.fit(x_train_scaled, y_train, epochs=30,
                    callbacks=[model_checkpoint_callback, earlystopping_loss_callback, earlystopping_acc_callback],
                    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))
```

```
> 82.320
```




Modifico el número de neuronas de dos capa tras el flatten pasando a 32, 128, 256 y 512. Además añado una capa de batch normalization tras el tercer grupo de convoluciones con la idea de que normalice los mini batches para evitar que tengan muchas diferencias entre ellos con el propósito de que sean mas similares los datos que llegan a los filtros de modos que la red entrene mejor y el modelo adquiera mayor robustez. El resultado ha sido muy satisfactorio aumentando la acuraccy hasta 82.320

# VERSIÓN 10

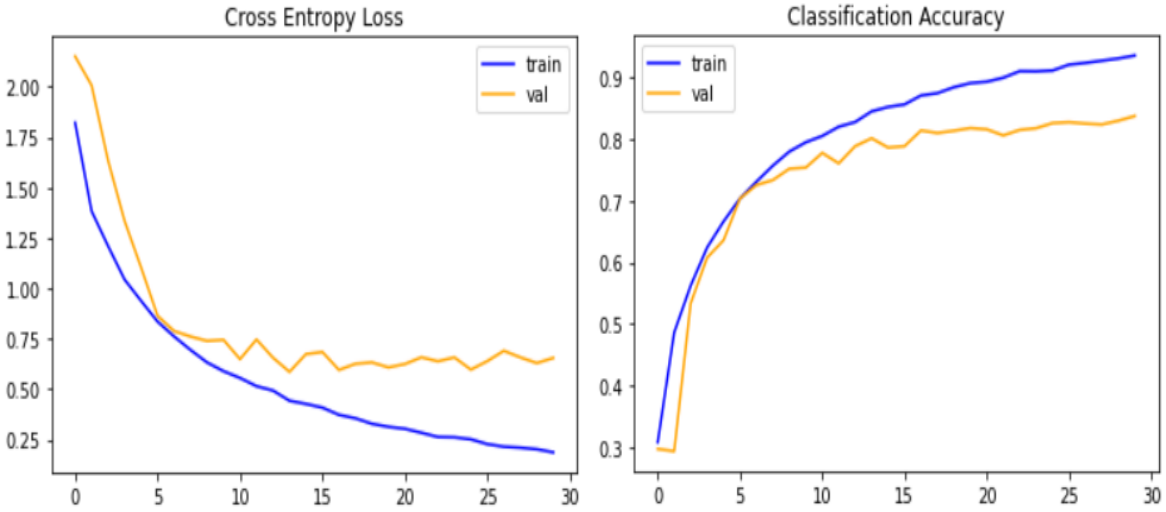
```
model = ks.Sequential()

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), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))


model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Conv2D(512, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(1024, (3,3), padding='same', activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(32, activation='relu'))
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.Dense(1024, activation='relu'))
```

```
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, 256)       295168
 max_pooling2d_1 (MaxPooling  (None, 8, 8, 256)        0
 2D)
 dropout_1 (Dropout)         (None, 8, 8, 256)         0
 conv2d_4 (Conv2D)           (None, 8, 8, 512)         1180160
 conv2d_5 (Conv2D)           (None, 8, 8, 1024)        4719616
 batch_normalization (BatchN  (None, 8, 8, 1024)       4096
 ormalization)
 max_pooling2d_2 (MaxPooling  (None, 4, 4, 1024)       0
 2D)
 dropout_2 (Dropout)         (None, 4, 4, 1024)        0
 flatten (Flatten)           (None, 16384)             0
 dense (Dense)               (None, 32)                524320
 dense_1 (Dense)             (None, 128)               4224
 dense_2 (Dense)             (None, 256)               33024
 dense_3 (Dense)             (None, 512)               131584
 dense_4 (Dense)             (None, 1024)              525312
 dense_5 (Dense)             (None, 10)                10250
=================================================================
Total params: 7,521,002
Trainable params: 7,518,954
Non-trainable params: 2,048
```

```
history = model.fit(x_train_scaled, y_train, epochs=30,
                    callbacks=[model_checkpoint_callback, earlystopping_loss_callback, earlystopping_acc_callback],
                    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))
```

```
> 83.310
```




Al Introducir una capa más de dense de 1024 neuronas tras el flatten y situada en última posición, la acuraccy ha mejorado casi un punto hasta situarse en 83.310 a su vez las gráficas muestran unos resultados muy similares a las versiones anteriores

# VERSIÓN 11

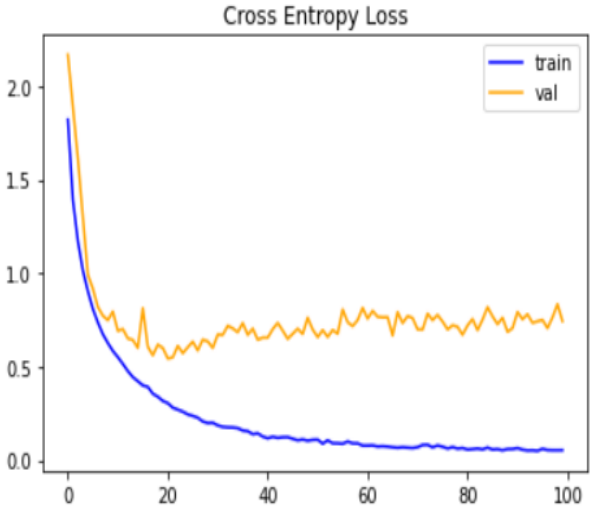
```
model = ks.Sequential()

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), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))


model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Conv2D(512, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(1024, (3,3), padding='same', activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(32, activation='relu'))
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.Dense(1024, activation='relu'))


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

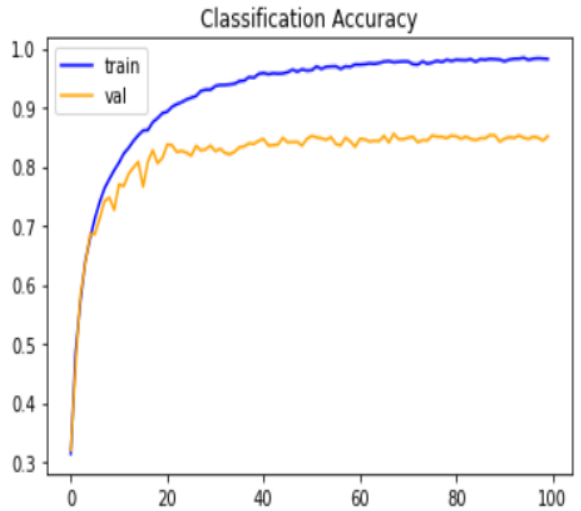
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896
 conv2d_1 (Conv2D)           (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, 256)       295168
 max_pooling2d_1 (MaxPooling  (None, 8, 8, 256)        0
 2D)
 dropout_1 (Dropout)         (None, 8, 8, 256)         0
 conv2d_4 (Conv2D)           (None, 8, 8, 512)         1180160
 conv2d_5 (Conv2D)           (None, 8, 8, 1024)        4719616
 batch_normalization (BatchN  (None, 8, 8, 1024)       4096
 ormalization)
 max_pooling2d_2 (MaxPooling  (None, 4, 4, 1024)       0
 2D)
 dropout_2 (Dropout)         (None, 4, 4, 1024)        0
 flatten (Flatten)           (None, 16384)             0
 dense (Dense)               (None, 32)                524320
 dense_1 (Dense)             (None, 128)               4224
 dense_2 (Dense)             (None, 256)               33024
 dense_3 (Dense)             (None, 512)               131584
 dense_4 (Dense)             (None, 1024)              525312
 dense_5 (Dense)             (None, 10)                10250
=================================================================
Total params: 7,521,002
Trainable params: 7,518,954
Non-trainable params: 2,048
_________________________________________________________________
```

```
history = model.fit(x_train_scaled, y_train, epochs=100,
                    callbacks=[model_checkpoint_callback, earlystopping_loss_callback, earlystopping_acc_callback],
                    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))
```

```
> 84.780
```




Mantengo la misma arquitectura que en la versión anterior y aumento las epochs a 100, la idea es que la red aprenda mas tiempo. El resultado ha sido satisfactorio y la accuracy ha aumentado hasta 84.780 .
Las gráficas parecen indicar que tanto validación error como validación de precisión del modelo se mantienen con cierto overfitting.

# VERSIÓN 12

```
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), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))



model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Conv2D(512, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(1024, (3,3), padding='same', activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(32, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(256, 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(1024, activation='relu'))



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

```
model.summary()
```

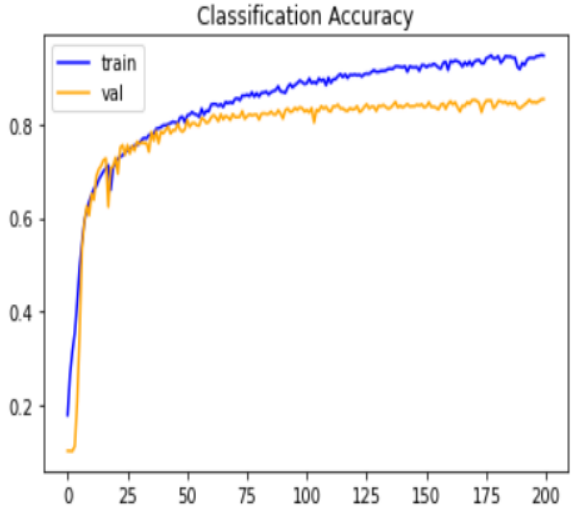
| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_43 (Conv2D) | (None, 32, 32, 32) | 896 |
| conv2d_44 (Conv2D) | (None, 32, 32, 64) | 18496 |
| max_pooling2d_21 (MaxPooling2D) | (None, 16, 16, 64) | 0 |
| dropout_26 (Dropout) | (None, 16, 16, 64) | 0 |
| conv2d_45 (Conv2D) | (None, 16, 16, 128) | 73856 |
| conv2d_46 (Conv2D) | (None, 16, 16, 256) | 295168 |
| max_pooling2d_22 (MaxPooling2D) | (None, 8, 8, 256) | 0 |
| dropout_27 (Dropout) | (None, 8, 8, 256) | 0 |
| conv2d_47 (Conv2D) | (None, 8, 8, 512) | 1180160 |
| conv2d_48 (Conv2D) | (None, 8, 8, 1024) | 4719616 |
| batch_normalization_7 (BatchNormalization) | (None, 8, 8, 1024) | 4096 |
| max_pooling2d_23 (MaxPooling2D) | (None, 4, 4, 1024) | 0 |
| dropout_28 (Dropout) | (None, 4, 4, 1024) | 0 |
| flatten_7 (Flatten) | (None, 16384) | 0 |
| dense_42 (Dense) | (None, 32) | 524320 |
| dropout_29 (Dropout) | (None, 32) | 0 |
| dense_43 (Dense) | (None, 128) | 4224 |
| dropout_30 (Dropout) | (None, 128) | 0 |
| dense_44 (Dense) | (None, 256) | 33024 |
| dropout_31 (Dropout) | (None, 256) | 0 |
| dense_45 (Dense) | (None, 512) | 131584 |
| dropout_32 (Dropout) | (None, 512) | 0 |
| dense_46 (Dense) | (None, 1024) | 525312 |
| dense_47 (Dense) | (None, 10) | 10250 |

```
Total params: 7,521,002
Trainable params: 7,518,954
Non-trainable params: 2,048
```

```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    callbacks=[model_checkpoint_callback, earlystopping_loss_callback, earlystopping_acc_callback],
                    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))
```

```
> 84.790
```


Se modifica la arquitectura y se añaden cuatro capas de dropout de 0.3 tras el flatten, una entre cada capa dense, la idea es que se reduzca el overfitting, se amplían las epochs hasta 200 para contrarrestar una posible pérdida de precisión al pasar las imágenes por menos neuronas. La accuracy se ha mantenido muy similar y el overfitting ha disminuido, tal y como se puede apreciar en los gráficos

# VERSIÓN 13

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))


model.add(ks.layers.Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Conv2D(512, (3,3), padding='same', activation='relu'))
model.add(ks.layers.Conv2D(1024, (3,3), padding='same', activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(32, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(256, 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(1024, activation='relu'))



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

```
model.summary()
```

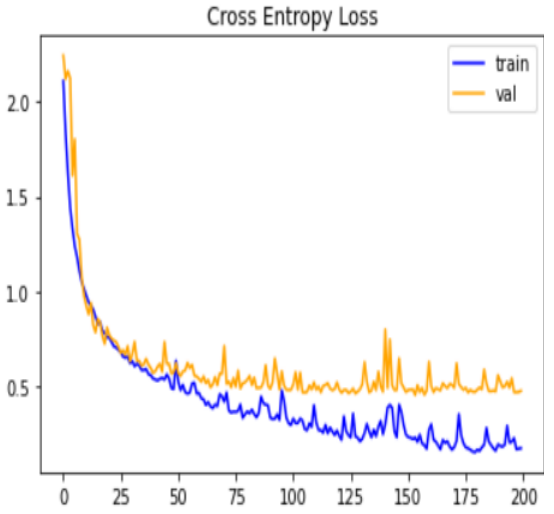
| | | |
|---|---|---|
| conv2d_6 (Conv2D) | (None, 32, 32, 64) | 1792 |
| conv2d_7 (Conv2D) | (None, 32, 32, 128) | 73856 |
| max_pooling2d_3 (MaxPooling2D) | (None, 16, 16, 128) | 0 |
| dropout_7 (Dropout) | (None, 16, 16, 128) | 0 |
| conv2d_8 (Conv2D) | (None, 16, 16, 128) | 147584 |
| conv2d_9 (Conv2D) | (None, 16, 16, 256) | 295168 |
| max_pooling2d_4 (MaxPooling2D) | (None, 8, 8, 256) | 0 |
| dropout_8 (Dropout) | (None, 8, 8, 256) | 0 |
| conv2d_10 (Conv2D) | (None, 8, 8, 512) | 1180160 |
| conv2d_11 (Conv2D) | (None, 8, 8, 1024) | 4719616 |
| batch_normalization_1 (BatchNormalization) | (None, 8, 8, 1024) | 4096 |
| max_pooling2d_5 (MaxPooling2D) | (None, 4, 4, 1024) | 0 |
| dropout_9 (Dropout) | (None, 4, 4, 1024) | 0 |
| flatten_1 (Flatten) | (None, 16384) | 0 |
| dense_6 (Dense) | (None, 32) | 524320 |
| dropout_10 (Dropout) | (None, 32) | 0 |
| dense_7 (Dense) | (None, 128) | 4224 |
| dropout_11 (Dropout) | (None, 128) | 0 |
| dense_8 (Dense) | (None, 256) | 33024 |
| dropout_12 (Dropout) | (None, 256) | 0 |
| dense_9 (Dense) | (None, 512) | 131584 |
| dropout_13 (Dropout) | (None, 512) | 0 |
| dense_10 (Dense) | (None, 1024) | 525312 |
| dense_11 (Dense) | (None, 10) | 10250 |

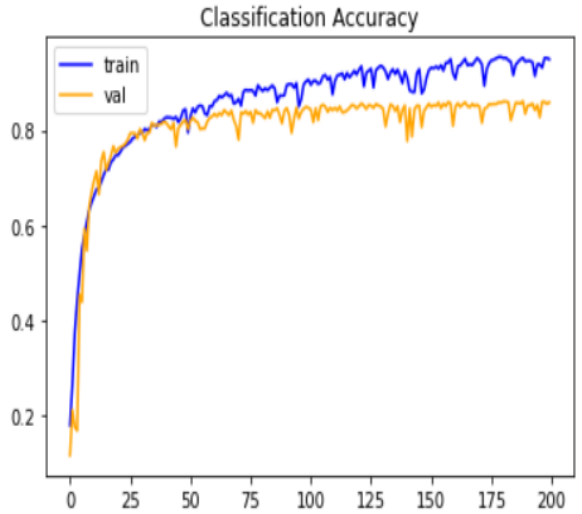
```
Total params: 7,650,986
Trainable params: 7,648,938
Non-trainable params: 2,048
```

```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    callbacks=[model_checkpoint_callback, earlystopping_loss_callback, earlystopping_acc_callback],
                    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))
```

```
> 85.480
```




Se aumenta el número de neuronas de las dos primeras capas de convoluciones con la idea de aumentar la accuracy del modelo, el resto se mantiene igual. La accuracy ha mejorado hasta situarse en 85.480 pero los gráficos presentan muchos picos.