CIFAR 10
Redes Neuronales

Antonio Navarro Barrero

Entrenamiento 1

Modelo básico, sin

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(3, (6, 6), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))

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_6"

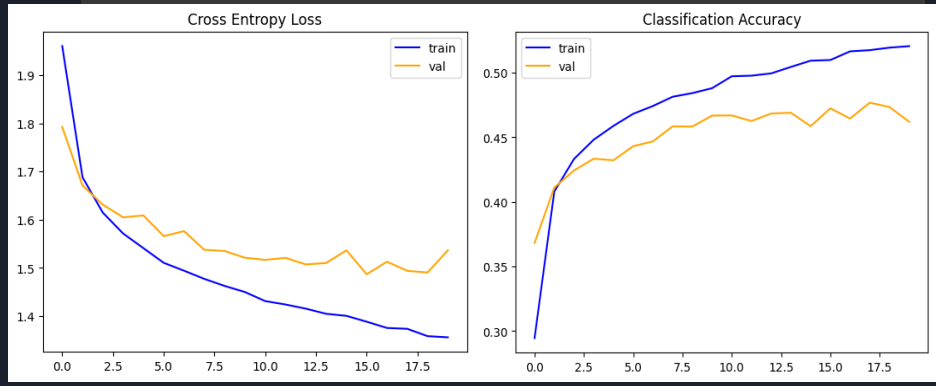
Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 32, 32, 3)	327
flatten_6 (Flatten)	(None, 3072)	0
dense_14 (Dense)	(None, 32)	98,336
dense_15 (Dense)	(None, 10)	330

Total params: 98,993 (386.69 KB)
Trainable params: 98,993 (386.69 KB)
Non-trainable params: 0 (0.00 B)

```
history = model.fit(x_train, y_train, epochs=20 ,batch_size= 512,
                    validation_data=(x_val, y_val), callbacks=[callback_loss, callback_accuracy])
```

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

```
> 48.170
```



Idea y conclusión:
Primeramente genero un modelo sencillo, sin modificar el esquema base, con el objetivo de visualizar los resultados que obtenemos con este primer modelo.
Vemos que con este modelo, que presenta casi 99.000 parámetros, llegamos a un accuracy en test de 48.17%, valor muy por debajo del umbral deseado.
Por ello vamos a modificar el modelo para mejorar los resultados.

Entrenamiento 2

```python
model = ks.Sequential()

model.add(ks.layers.Conv2D(16, (6, 6), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Conv2D(32, (6, 6), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))

model.add(ks.layers.Conv2D(64, (6, 6), 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'))
```

```python
model.summary()
```

Model: "sequential_10"

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 32, 32, 16)	1,744
max_pooling2d_4 (MaxPooling2D)	(None, 16, 16, 16)	0
conv2d_13 (Conv2D)	(None, 16, 16, 32)	18,464
max_pooling2d_5 (MaxPooling2D)	(None, 8, 8, 32)	0
conv2d_14 (Conv2D)	(None, 8, 8, 64)	73,792
max_pooling2d_6 (MaxPooling2D)	(None, 4, 4, 64)	0
flatten_10 (Flatten)	(None, 1024)	0
dense_22 (Dense)	(None, 32)	32,800
dense_23 (Dense)	(None, 10)	330

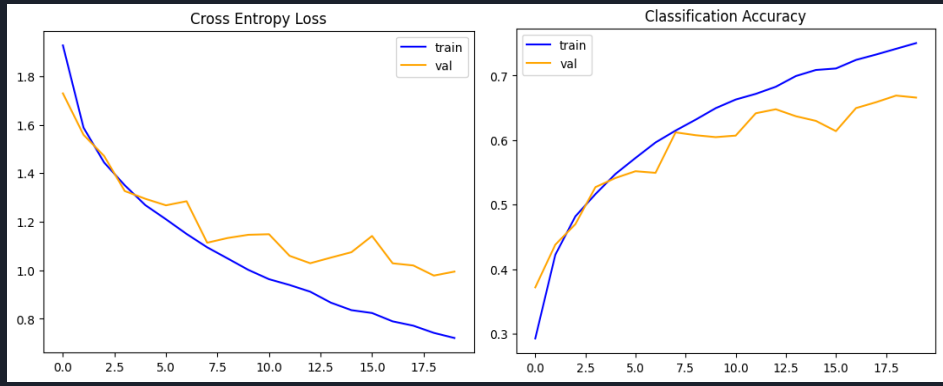
Total params: 127,130 (496.60 KB)
Trainable params: 127,130 (496.60 KB)
Non-trainable params: 0 (0.00 B)

```python
history = model.fit(x_train, y_train, epochs=20 ,batch_size= 512,
                    validation_data=(x_val, y_val), callbacks=[callback_loss, callback_accuracy])
```

Modelo

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

```
> 66.470
```



Idea y conclusión:
Ahora he añadido 3 capas convolucionales progresivas con su correspondiente capa de MaxPooling (para reducir la dimensionalidad de la imagen y así intentar que el modelo sea más eficiente), con el objetivo de generar un modelo más avanzado y robusto.
Como vemos en los resultados, hemos conseguido aumentar notablemente la precisión del modelo, llegando a un 66.47% de accuracy en test.

Entrenamiento 3

D

Modelo con

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (6, 6), 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.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))


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(0.25))


model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dense(10, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential_1"
```

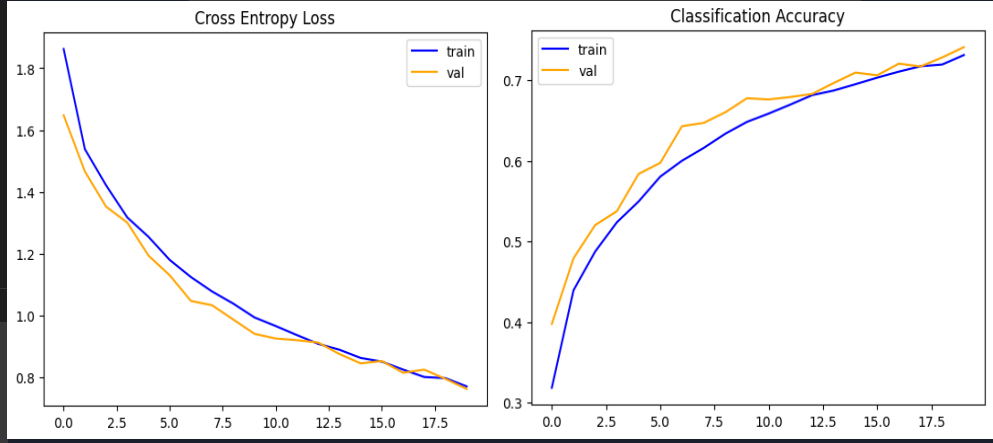
Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 32, 32, 32)	3,488
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_4 (Conv2D)	(None, 16, 16, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_2 (Dropout)	(None, 8, 8, 64)	0
conv2d_5 (Conv2D)	(None, 8, 8, 128)	73,856
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_3 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 128)	262,272
dense_1 (Dense)	(None, 10)	1,290

```
Total params: 359,402 (1.37 MB)
Trainable params: 359,402 (1.37 MB)
Non-trainable params: 0 (0.00 B)
```

```
history = model.fit(x_train, y_train, epochs=20 ,batch_size= 512,
                    validation_data=(x_val, y_val), callbacks=[callback_loss, callback_accuracy])
```

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

```
> 72.800
```



Idea y conclusión:

En este tercer modelo, he aumentado el número de filtros en las capas convolucionales y he introducido capas de Dropout después de algunas de las capas de Max Pooling. El objetivo de este cambio es reducir el riesgo de sobreajuste y mejorar la generalización del modelo. Con ello, logramos alcanzar un accuracy de casi un 73% en el conjunto de test, lo que representa una mejora significativa respecto a los modelos anteriores. A pesar de este incremento, aún estamos por debajo del 90% de accuracy, por lo que seguiremos ajustando el modelo para mejorar los resultados.

Entrenamiento 4

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (6, 6), strides=1, activation='relu',
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
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.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.25))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dense(10, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 32, 32, 32)	3,488
batch_normalization_7 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d_6 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_10 (Conv2D)	(None, 16, 16, 64)	18,496
batch_normalization_8 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_7 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_6 (Dropout)	(None, 8, 8, 64)	0
conv2d_11 (Conv2D)	(None, 8, 8, 128)	73,856
batch_normalization_9 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_8 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_7 (Dropout)	(None, 4, 4, 128)	0
flatten_2 (Flatten)	(None, 2048)	0
dense_4 (Dense)	(None, 128)	262,272
dense_5 (Dense)	(None, 10)	1,290

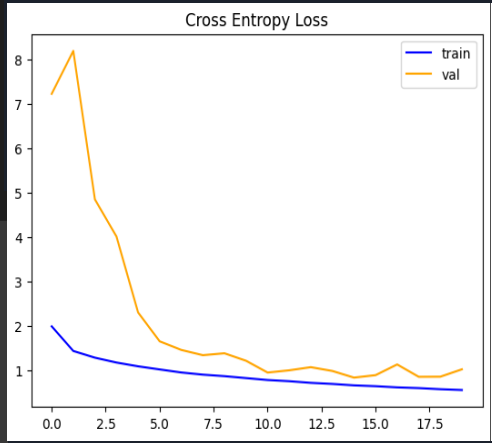
```
Total params: 360,298 (1.37 MB)
Trainable params: 359,850 (1.37 MB)
Non-trainable params: 448 (1.75 KB)
```

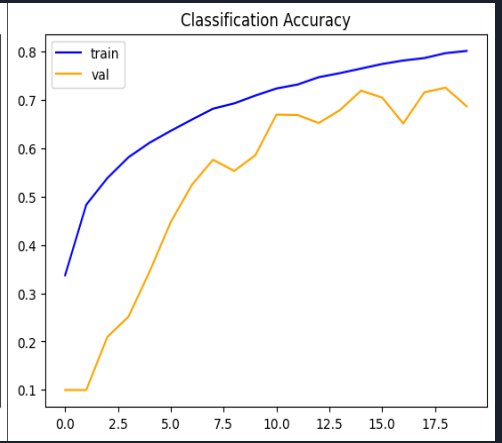
Modelo con Batch

```
history = model.fit(x_train, y_train, epochs=20 ,batch_size= 512,
                    validation_data=(x_val, y_val), callbacks=[callback_loss, callback_accuracy])
```

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

```
> 72.360
```





Idea y conclusión:

En este cuarto modelo, he implementado el uso de Batch Normalization en la red, pero, observando tanto las gráficas como la propia precisión en el fichero test, o hemos conseguido mejorar el accuracy del modelo, a pesar de haber añadido el Batch Normalization.

Para solucionar esto, vamos a aumentar la complejidad del modelo, añadiendo nuevas capas, y por tanto, aumentando también la profundidad de la red.

Entrenamiento 5

Profundidad

```
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.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
          padding='same', 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(128, (3, 3), strides=1, activation='relu',
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.25))

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.35))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(10, activation='softmax'))
```

Modelo con Mayor

```
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_22 (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization_20 (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_23 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_21 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_20 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_14 (Dropout)	(None, 16, 16, 64)	0
conv2d_24 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_22 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_25 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_23 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_21 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_15 (Dropout)	(None, 8, 8, 128)	0
conv2d_26 (Conv2D)	(None, 8, 8, 256)	295,168
batch_normalization_24 (BatchNormalization)	(None, 8, 8, 256)	1,024
conv2d_27 (Conv2D)	(None, 8, 8, 256)	590,080
batch_normalization_25 (BatchNormalization)	(None, 8, 8, 256)	1,024
max_pooling2d_22 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_16 (Dropout)	(None, 4, 4, 256)	0
flatten_7 (Flatten)	(None, 4096)	0
dense_15 (Dense)	(None, 512)	2,097,664
batch_normalization_26 (BatchNormalization)	(None, 512)	2,048
dropout_17 (Dropout)	(None, 512)	0
dense_16 (Dense)	(None, 10)	5,130

Total params: 3,253,834 (12.41 MB)
Trainable params: 3,251,018 (12.40 MB)

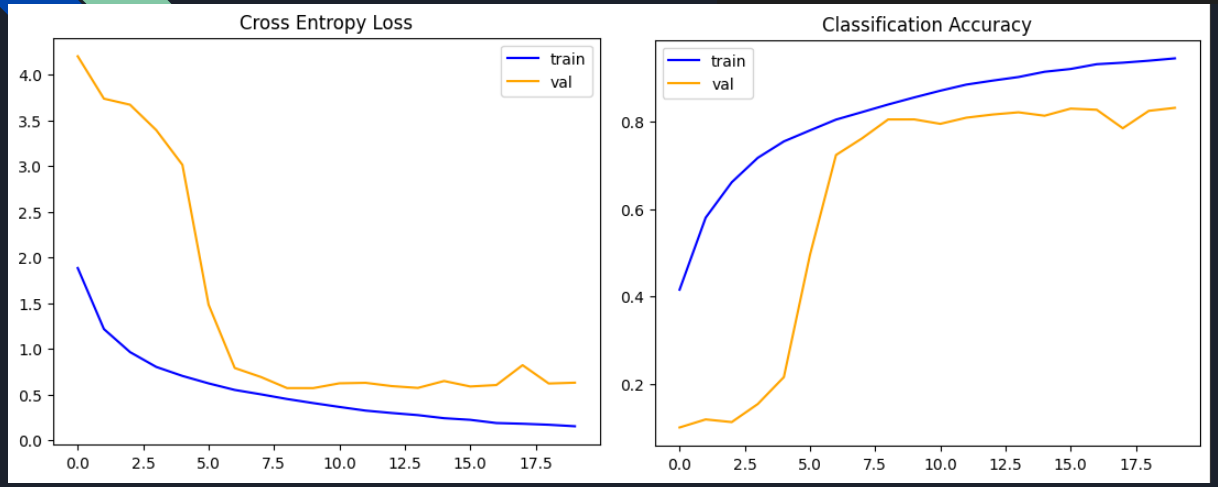
Entrenamiento 5
Profundidad

Modelo con Mayor

```
history = model.fit(x_train, y_train, epochs=20 ,batch_size= 512,
                    validation_data=(x_val, y_val), callbacks=[callback_loss, callback_accuracy])
```

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

```
> 82.550
```



Idea y conclusión:
En este 4º modelo, he incrementado la profundidad de la red añadiendo más capas convolucionales y filtros. Aplico Batch Normalization después de cada capa convolucional para acelerar el entrenamiento y mejorar la estabilidad del modelo. También he ajustado las tasas de Dropout para combatir el sobreajuste a medida que la red se vuelve más profunda. Esto nos ha permitido superar el 80% de accuracy, alcanzando un 83% en el conjunto de test. Aún teniendo ya un modelo bastante fiable, al tener una precisión del 83%, vamos a intentar aplicar nuevas técnicas al modelo, con el objetivo de aumentar la precisión hasta el 90%.

Para ello, voy a implementar el uso de Data Aumentation, que consiste en modificar las imágenes exsitentes en el fichero train, por ejemplo con rotaciones o cambios de color, con el objetivo de aumentar la información al tener mayor número de imágenes para una misma categoría.

Entrenamiento 6

```python
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.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
          padding='same', 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(128, (3, 3), strides=1, activation='relu',
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.25))

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.35))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(10, activation='softmax'))
```

```python
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# 5. Ajustar el generador a los datos de entrenamiento
datagen.fit(x_train)
```

Modelo con Data

```python
model.summary()
```

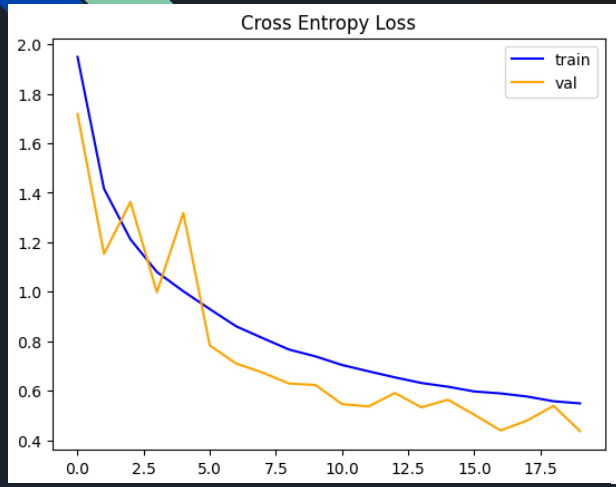
Layer (type)	Output Shape	Param #
conv2d_22 (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization_20 (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_23 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_21 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_20 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_14 (Dropout)	(None, 16, 16, 64)	0
conv2d_24 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_22 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_25 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_23 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_21 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_15 (Dropout)	(None, 8, 8, 128)	0
conv2d_26 (Conv2D)	(None, 8, 8, 256)	295,168
batch_normalization_24 (BatchNormalization)	(None, 8, 8, 256)	1,024
conv2d_27 (Conv2D)	(None, 8, 8, 256)	590,080
batch_normalization_25 (BatchNormalization)	(None, 8, 8, 256)	1,024
max_pooling2d_22 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_16 (Dropout)	(None, 4, 4, 256)	0
flatten_7 (Flatten)	(None, 4096)	0
dense_15 (Dense)	(None, 512)	2,097,664
batch_normalization_26 (BatchNormalization)	(None, 512)	2,048
dropout_17 (Dropout)	(None, 512)	0
dense_16 (Dense)	(None, 10)	5,130

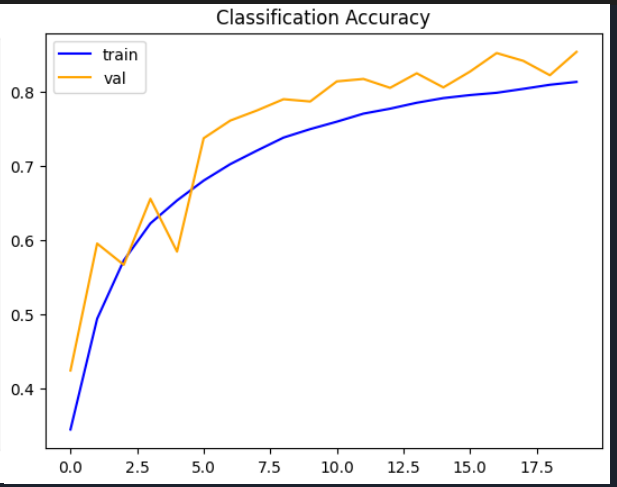
Total params: 3,253,834 (12.41 MB)
Trainable params: 3,251,018 (12.40 MB)

Entrenamiento 6
Aumentation

Modelo con Data

```
history = model.fit(datagen.flow(x_train, y_train, batch_size=32), epochs=20 ,
                    validation_data=(x_val, y_val), callbacks=[callback_loss, callback_accuracy])
```






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

```
> 84.730
```

Idea y conclusión:
Mediante el uso de Data Aumentation en el modelo, cuyo objetivo es tener más información en el entrenamiento de la red, no conseguimos una mejora significativa en el modelo, pasando de un 82,5% a un 84.7% en accuracy. Esto puede ser debido a que no le ha dado tiempo a captar ciertos patrones por culpa de tener un bajo número de Epochs. (Aumento el número de los Epoch porque como las imágenes del entrenamiento son más complejas debido al Data Aumentation, le permito que entrene durante más tiempo)

Incremento los Epochs del modelo a 100.

Entrenamiento 7

Modelo

```
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.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
          padding='same', 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(128, (3, 3), strides=1, activation='relu',
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.25))

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.35))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(10, activation='softmax'))
```

```
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# 5. Ajustar el generador a los datos de entrenamiento
datagen.fit(x_train)
```

```
model.summary()
```

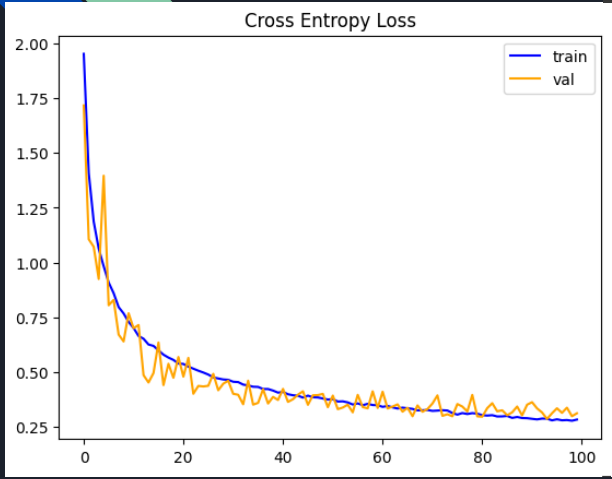
Layer (type)	Output Shape	Param #
conv2d_22 (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization_20 (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_23 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_21 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_20 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_14 (Dropout)	(None, 16, 16, 64)	0
conv2d_24 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_22 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_25 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_23 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_21 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_15 (Dropout)	(None, 8, 8, 128)	0
conv2d_26 (Conv2D)	(None, 8, 8, 256)	295,168
batch_normalization_24 (BatchNormalization)	(None, 8, 8, 256)	1,024
conv2d_27 (Conv2D)	(None, 8, 8, 256)	590,080
batch_normalization_25 (BatchNormalization)	(None, 8, 8, 256)	1,024
max_pooling2d_22 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_16 (Dropout)	(None, 4, 4, 256)	0
flatten_7 (Flatten)	(None, 4096)	0
dense_15 (Dense)	(None, 512)	2,097,664
batch_normalization_26 (BatchNormalization)	(None, 512)	2,048
dropout_17 (Dropout)	(None, 512)	0
dense_16 (Dense)	(None, 10)	5,130

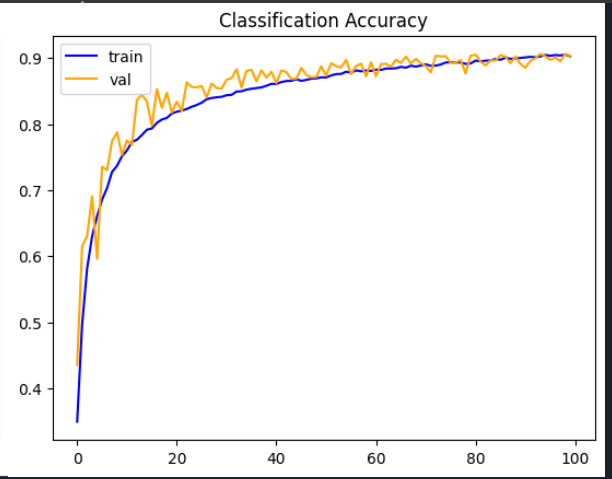
Total params: 3,253,834 (12.41 MB)
Trainable params: 3,251,018 (12.40 MB)

Entrenamiento 7 Final

Modelo

```
history = model.fit(datagen.flow(x_train, y_train, batch_size=32), epochs=100 ,
                    validation_data=(x_val, y_val), callbacks=[callback_loss, callback_accuracy])
```






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

```
> 90.120
```

Idea y conclusión:

Vemos que aplicando 100 epochs al modelo conseguimos llegar al 90% de precisión. Gracias a la gráfica 'Classification Accuracy' vemos como este modelo necesita un mayor número de Epochs para un correcto funcionamiento, ya que, hasta que no llega a los 60 Epoch, el modelo tiene una mayor precisión en el fichero test que en el train. Esto es debido al Data Aumentation, ya que complica el entrenamiento y necesita más tiempo para aprender. Ya en los últimos Epochs, vemos como ambas líneas se igualan y prácticamente no existe variación entre ambos ficheros.

Hemos conseguido finalmente un modelo que supera el 90% de precisión en test, lo que lo convierte en un modelo muy robusto y fiable.

Entrenamiento 7 Final

Modelo

Categorías:

0 avión	5 perro
1 automóvil	6 rana
2 pájaro	7 caballo
3 gato	8 barco
4 venado	9 camión

Este modelo de redes neuronales realiza la **clasificación de imágenes** para diversas categorías, como animales y vehículos, con una **precisión del 90%**. Para entender cómo funcionan las predicciones, vamos a utilizar un ejemplo. En la primera imagen de un gato, el modelo predice correctamente la clase "3" con un 100% de confianza.