



# DEEP LEARNING

## VISION POR COMPUTACIÓN

### ENTREGABLE IV

Paula García Vico

## PROYECTO Y OBJETIVOS

El dataset a entrenar es el de CIFAR10, disponible en Keras, incluye 50.000 imágenes para entrenamiento y 10.000 para pruebas. Las imágenes son RGB, a todo color, de 32x32 píxeles y están clasificadas en 10 categorías diferentes que son aviones, automóviles, pájaros, gatos, ciervos, perros, ranas, caballos, barcos y camiones.

El objetivo del proyecto es evaluar 10 combinaciones diferentes de elementos, recolectar y analizar los resultados para proporcionar una comparación integral entre las combinaciones probadas.

Proyectos realizados y resultados obtenidos:

- Proyecto 0 → accuracy = 60,04 % (Early Stopping)
- Proyecto 1 → accuracy = 67,19 % (añadir más capas de convolución y densidad)
- Proyecto 2 → accuracy = 72,61 % (añadir dropout)
- Proyecto 3 → accuracy = 76,76 % (duplicidad de las capas)
- Proyecto 4 → accuracy = 80,21 % (incrementar la capacidad de las neuronas)
- Proyecto 5 → accuracy = 82,07 % (modificar dropout)
- Proyecto 6 → accuracy = 81,22 % (añadir batch normalization)
- Proyecto 7 → accuracy = 83,08 % (añadir kernel\_initializer y el kernel\_regularizer)
- Proyecto 8 → accuracy = 83,62 % (data augmentation)
- Proyecto 9 → accuracy = 62,76 % (VGG16 con ImageNet pero sin pesos congelados)
- Proyecto 10 → accuracy = 77,61 % (VGG16 con ImageNet y pesos congelados)

## 0. PROYECTO 0: base\_cnn\_cifar10.ipynb → Accuracy = 60,04%

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

Para revisar un modelo, nos basta con llamar al método `.summary()` del modelo

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 32)	262176
dense_1 (Dense)	(None, 10)	330

=====  
Total params: 263402 (1.00 MB)  
Trainable params: 263402 (1.00 MB)  
Non-trainable params: 0 (0.00 Byte)

El optimizador, de momento será Adam

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

Se añaden los callbacks de Early Stopping que permite detener el entrenamiento del modelo si una cierta métrica no mejora después de un cierto número de epochs (patience). En este caso, se detendrá el entrenamiento después de 5 epochs consecutivos sin mejora.

```
[6] # definir los callbacks de EarlyStopping
callback_loss=EarlyStopping (monitor='val_loss', patience=5)
callback_acc=EarlyStopping (monitor='val_accuracy', patience=5)
```

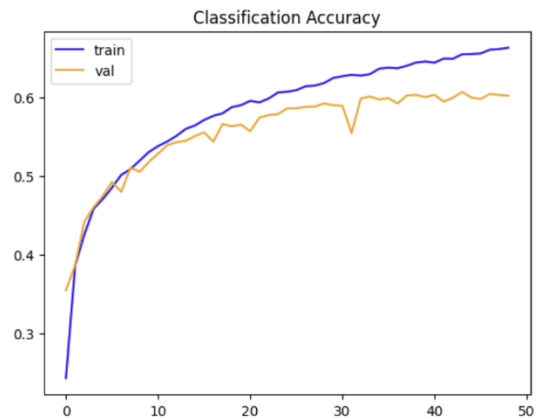
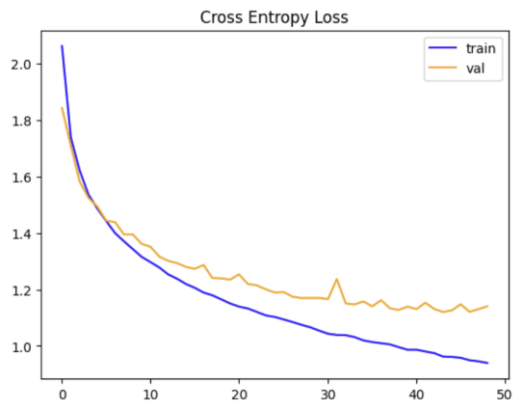
Se cambia el número de epochs a 300, se realiza esta cantidad de epochs ya que está delimitada por el Early Stopping.

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

El accuracy es de un 60%, aún no es un buen modelo

```
[16] _, acc = model.evaluate(x_test_scaled, y_test, verbose=0)  
      print('> %.3f' % (acc * 100.0))  
  
> 60.040
```

Representación gráfica de accuracy y entropy loss



## 1. PROYECTO 1: 1\_cnn\_cifar10.ipynb → Accuracy = 67,19 %

Se inicia añadiendo 2 capas más de convolución bidimensional y una capa más densidad a lo que ya tenía el proyecto base. A la vez, se aumenta la cantidad de neuronas en esta capa densa, pasando de una de 32 a una de 128 y otra de 64 sucesivamente.

Se decide iniciar de esta manera porque agregar más capas convolucionales y de densidad a un modelo puede ayudar a aumentar su capacidad de aprendizaje, permitiendo la extracción de patrones más complejos y facilitando la generalización de imágenes.

```
model = ks.Sequential()

# Primera capa
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)))

# Segunda capa
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)))

# Tercera capa
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(128, activation='relu'))
model.add(ks.layers.Dense(64, activation='relu'))

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

```
model.summary()
```

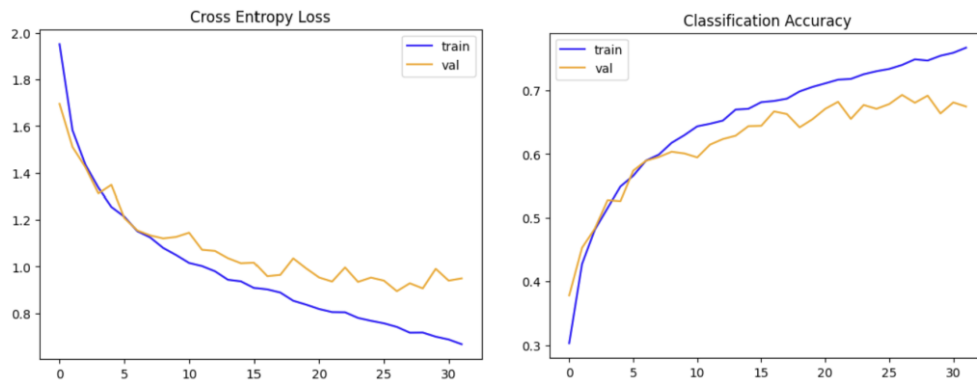
Model: "sequential"

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, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 32)	0
conv2d_2 (Conv2D)	(None, 8, 8, 32)	9248
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 32)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 128)	65664
dense_1 (Dense)	(None, 64)	8256
dense_2 (Dense)	(None, 10)	650

```
=====  
Total params: 93962 (367.04 KB)  
Trainable params: 93962 (367.04 KB)  
Non-trainable params: 0 (0.00 Byte)
```

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

```
> 67.190
```



El gráfico de "Cross Entropy Loss" muestra que el train es positivo, pero la pérdida de validación indica que puede haber un sobreajuste ya que se desvía del train. La "Classification Accuracy" revela un aumento constante en el entrenamiento, sin embargo, la validación, que no mejora de la misma forma, también indica sobreajuste. En ambos casos, el modelo mejora respecto del anterior ya que es más complejo, pero se debe ir con cuidado con el overfitting y la correcta generalización de los nuevos datos.

## 2. PROYECTO 2: 2\_cnn\_cifar10.ipynb → Accuracy = 72,61 %

En el modelo anterior se concluyó, a partir de las gráficas, que se podría tener un pequeño sobreajuste, se añade a cada capa un Dropout del 25%. Esta técnica se utiliza para reducir y/o prevenir dicho overfitting, ya que, su función consiste en desactivar un porcentaje de neuronas de la red de forma aleatoria durante el entrenamiento para ayudar a la red a aprender patrones más generalizables en lugar de memorizar datos. Los dropout también permitirán hacer más compleja y profunda la red neuronal en caso de que sea necesaria para mejorar el accuracy.

```
model = ks.Sequential()

# Primera capa
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.Dropout(0.25))


# Segunda capa
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.Dropout(0.25))


# Tercera capa
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.Dropout(0.25))

model.add(ks.layers.Flatten())

model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dropout(0.25))
model.add(ks.layers.Dense(64, activation='relu'))
model.add(ks.layers.Dropout(0.25))

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

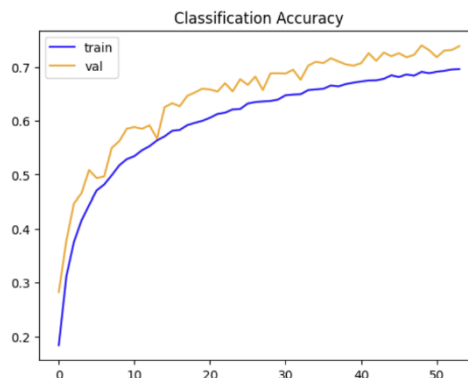
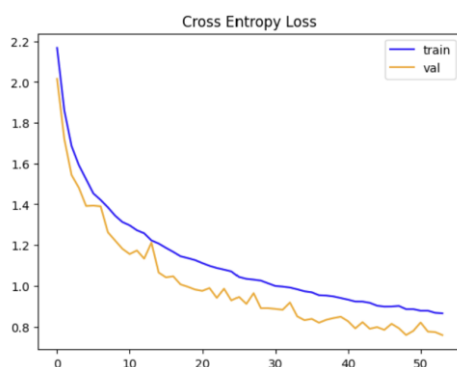
 `model.summary()`

 Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_1 (Conv2D)	(None, 16, 16, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 32)	0
dropout_1 (Dropout)	(None, 8, 8, 32)	0
conv2d_2 (Conv2D)	(None, 8, 8, 32)	9248
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 32)	0
dropout_2 (Dropout)	(None, 4, 4, 32)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 128)	65664
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8256
dropout_4 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 10)	650
=====		
Total params: 93962 (367.04 KB)		
Trainable params: 93962 (367.04 KB)		
Non-trainable params: 0 (0.00 Byte)		

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

> 72.610
```



Este modelo incluye dropout, por lo tanto, en comparación con el proyecto anterior se observa que, la brecha entre la pérdida de entrenamiento y la de validación es menor, lo cual indica que el dropout está ayudando a generalizar mejor. Además, ha habido un aumento de la precisión a 72,61%.



### 3. PROYECTO 3: 3\_cnn\_cifar10.ipynb → Accuracy = 76,76 %

En este caso, se opta por duplicar las capas con el objetivo de aumentar la profundidad para que, teóricamente, la red pueda aprender jerarquías más complejas. Además, ayudará a la regulación del sobre ajuste ya que el dropout anterior hacía que validación estuviera por debajo de train.

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

# Primera capa
model.add(ks.layers.Conv2D(16, (3, 3), strides=1, activation='relu',
padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(16, (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))

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

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

model.add(ks.layers.Flatten())

model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.Dropout(0.25))
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dropout(0.25))

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

```
[ ] model.summary()
```

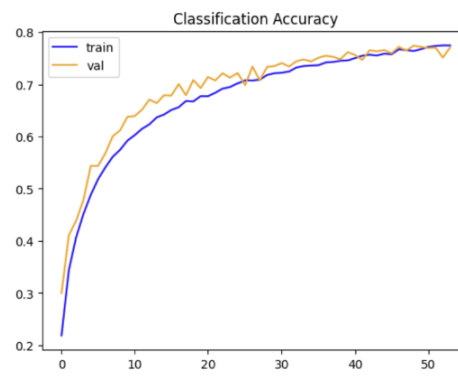
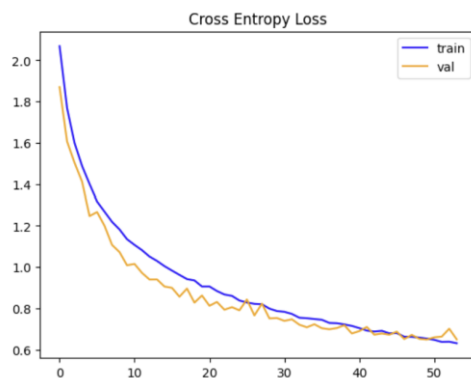
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 16)	448
conv2d_1 (Conv2D)	(None, 32, 32, 16)	2320
max_pooling2d (MaxPooling2D)	(None, 16, 16, 16)	0
dropout (Dropout)	(None, 16, 16, 16)	0
conv2d_2 (Conv2D)	(None, 16, 16, 32)	4640
conv2d_3 (Conv2D)	(None, 16, 16, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 32)	0
dropout_1 (Dropout)	(None, 8, 8, 32)	0
conv2d_4 (Conv2D)	(None, 8, 8, 32)	9248
conv2d_5 (Conv2D)	(None, 8, 8, 32)	9248
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 32)	0
dropout_2 (Dropout)	(None, 4, 4, 32)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 256)	131328
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896
dropout_4 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

=====  
Total params: 200666 (783.85 KB)  
Trainable params: 200666 (783.85 KB)  
Non-trainable params: 0 (0.00 Byte)

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

> 76.760



En este caso, añadir una duplicidad de las capas ha tenido un efecto de regularización implícito, como se observa en las gráficas, se ha reducido el overfitting sustancialmente.

#### 4. PROYECTO 4: 4\_cnn\_cifar10.ipynb → Accuracy = 80,21 %

En este proyecto, se ha decantado por incrementar la capacidad de las neuronas en las diferentes capas convolucionales. A diferencia del modelo anterior, que contaba con filtros de 16 y 32, en esta nueva estructura amplían las neuronas a 64, 128 y 256 en cada capa de manera sucesiva. El objetivo de este cambio es incrementar la capacidad del modelo para aprender relaciones más complejas en los datos, ya que venía siendo un modelo simple por el número de neuronas en cada capa que era reducido.

```
model = ks.Sequential()

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

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

# Tercera capa
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.25))

model.add(ks.layers.Flatten())

model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.Dropout(0.25))
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dropout(0.25))

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

```
model.summary()
```

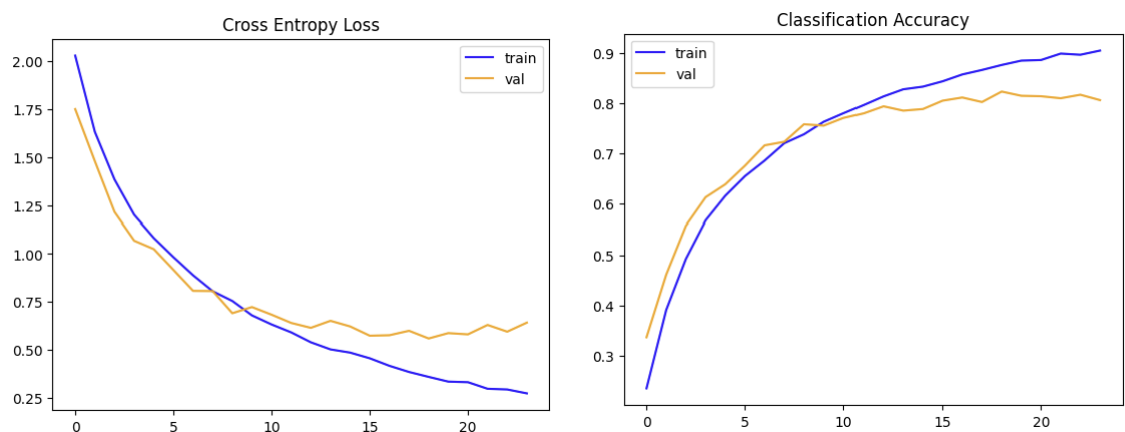
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1792
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_1 (Dropout)	(None, 8, 8, 128)	0
conv2d_4 (Conv2D)	(None, 8, 8, 256)	295168
conv2d_5 (Conv2D)	(None, 8, 8, 256)	590080
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 256)	1048832
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896
dropout_4 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

=====  
Total params: 2228426 (8.50 MB)  
Trainable params: 2228426 (8.50 MB)  
Non-trainable params: 0 (0.00 Byte)

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

> 80.210
```



En comparación con el proyecto anterior, es notable que este modelo tiene un mejor desempeño, ya que, al cambiar los filtros de 16 y 32 a 64, 128 y 2256 respectivamente, el modelo puede haber aprendido características más complejas que mejoran el rendimiento. Aunque, existe una mayor separación entre train y validación que podría dar lugar a un overfitting.

## 5. PROYECTO 5: 5\_cnn\_cifar10.ipynb → Accuracy = 82,07 %

Se decide aumentar el porcentaje de dropout en las capas de convolución de 25% a 30% y en las capas de densidad de 25% a 40%, para evitar que la red neuronal la separación al final del entrenamiento entre el entrenamiento y la validación, ya que esto está indicando un sobreajuste de la red.

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

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

# Segunda capa
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

# Tercera capa
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Flatten())

model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.Dropout(0.4))
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.Dropout(0.4))

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

```
model.summary()
```

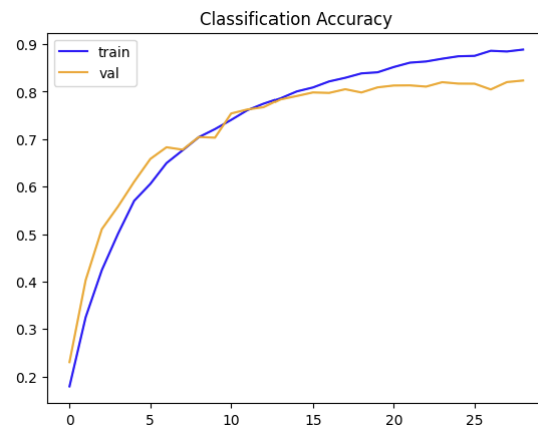
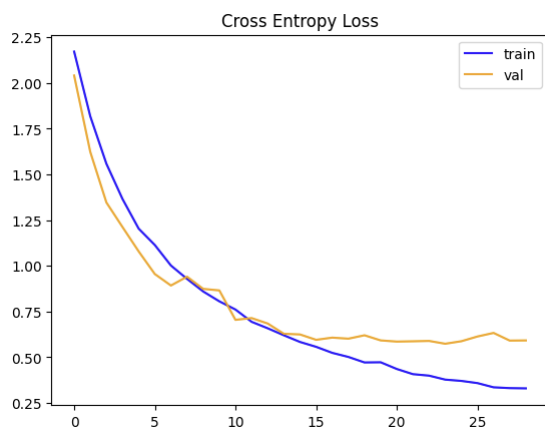
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1792
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_1 (Dropout)	(None, 8, 8, 128)	0
conv2d_4 (Conv2D)	(None, 8, 8, 256)	295168
conv2d_5 (Conv2D)	(None, 8, 8, 256)	590080
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 256)	1048832
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896
dropout_4 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

=====  
Total params: 2228426 (8.50 MB)  
Trainable params: 2228426 (8.50 MB)  
Non-trainable params: 0 (0.00 Byte)

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

> 82.070



En comparación con la gráfica anterior, sigue habiendo, al final de ambas, una separación entre las curvas de validación y entrenamiento. Es cierto que se ha conseguido reducir de forma notable la distancia entre las curvas gracias al incremento del dropout, sin embargo, no ha sido suficiente. Los pasos siguientes deberían incluir la incorporación de Batch Normalization y otras técnicas de regularización como L1 y/o L2, que podrían resultar más efectivas.

## 6. PROYECTO 6: 6\_cnn\_cifar10.ipynb → Accuracy = 81,22 %

Como en el proyecto anterior hay ya 9 capas, lo ideal es añadir Batch Normalization, debido a que, la arquitectura de la red neuronal es más compleja. Añadir Batch Normalization nos permite que el entrenamiento de la red se vuelva menos sensible en la elección de pesos, facilita el entrenamiento, ayuda a que aprenda más rápidamente y mejore su rendimiento.

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

# Primera capa
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(0.3))

# Segunda capa
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(0.3))

# Tercera capa
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(0.3))

model.add(ks.layers.Flatten())

model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.4))
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.4))

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

```
[ ] model.summary()
```

Model: "sequential"

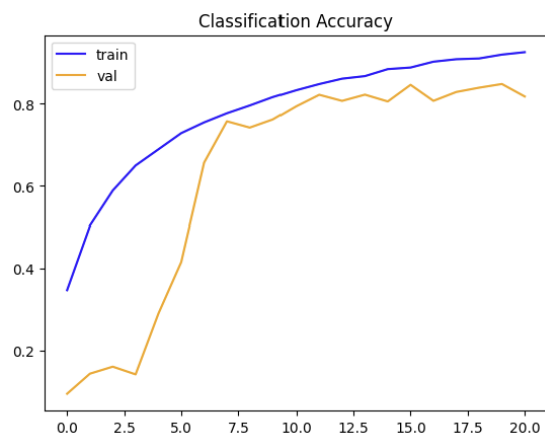
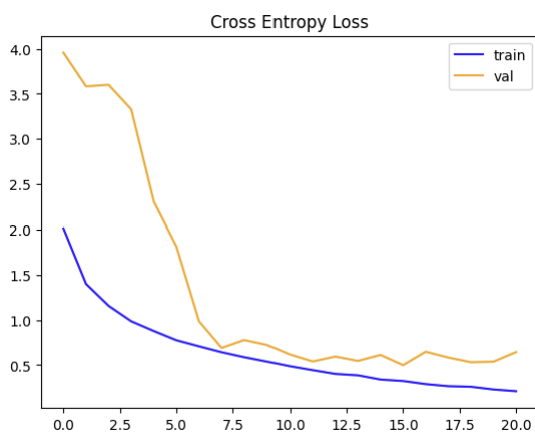
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1792
batch_normalization (Batch Normalization)	(None, 32, 32, 64)	256
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36928
batch_normalization_1 (Batch Normalization)	(None, 32, 32, 64)	256
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73856
batch_normalization_2 (Batch Normalization)	(None, 16, 16, 128)	512
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147584
batch_normalization_3 (Batch Normalization)	(None, 16, 16, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_1 (Dropout)	(None, 8, 8, 128)	0
conv2d_4 (Conv2D)	(None, 8, 8, 256)	295168
batch_normalization_4 (Batch Normalization)	(None, 8, 8, 256)	1024
conv2d_5 (Conv2D)	(None, 8, 8, 256)	590080
batch_normalization_5 (Batch Normalization)	(None, 8, 8, 256)	1024
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, 256)	1048832
batch_normalization_6 (Batch Normalization)	(None, 256)	1024
dropout_3 (Dropout)	(None, 256)	0

dense_1 (Dense)	(None, 128)	32896
batch_normalization_7 (Batch Normalization)	(None, 128)	512
dropout_4 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

=====  
 Total params: 2233546 (8.52 MB)  
 Trainable params: 2230986 (8.51 MB)  
 Non-trainable params: 2560 (10.00 KB)

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

> 81.220
```





## 7. PROYECTO 7: 7\_cnn\_cifar10.ipynb → Accuracy = 83,08 %

En este proyecto, se ha añadido dentro de cada capa el `kernel_initializer` y el `kernel_regularizer`, esto puede ayudar al modelo a que aprenda de imágenes nuevas de manera más efectiva.

Por un lado, `kernel_initializer='he_uniform'`: ayuda a evitar problemas iniciales de entrenamiento y puede hacer que el aprendizaje sea más eficiente. Por otro lado, `kernel_regularizer=l2(0.001)`: añade una penalización a los pesos más grandes, incentivando un modelo más simple y previniendo el sobreajuste.

```
model = ks.Sequential()

# Primera capa
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3), kernel_initializer = 'he_uniform', kernel_regularizer = l2(0.001)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same', kernel_initializer = 'he_uniform', kernel_regularizer = l2(0.001)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

# Segunda capa
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', kernel_initializer = 'he_uniform', kernel_regularizer = l2(0.001)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', kernel_initializer = 'he_uniform', kernel_regularizer = l2(0.001)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

# Tercera capa
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', kernel_initializer = 'he_uniform', kernel_regularizer = l2(0.001)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', kernel_initializer = 'he_uniform', kernel_regularizer = l2(0.001)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

model.add(ks.layers.Flatten())

model.add(ks.layers.Dense(256, activation='relu', kernel_initializer = 'he_uniform', kernel_regularizer = l2(0.001)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.4))
model.add(ks.layers.Dense(128, activation='relu', kernel_initializer = 'he_uniform', kernel_regularizer = l2(0.001)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.4))

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



## 8. PROYECTO 8: 8\_cnn\_cifar10.ipynb → Accuracy = 83,62 %

Mantenemos las mismas capas de la red neuronal que en el proyecto 6, en este caso se aplicará data augmentation.

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

# Primera capa
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(0.3))

# Segunda capa
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(0.3))

# Tercera capa
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(0.3))

model.add(ks.layers.Flatten())

model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.4))
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.4))

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

model.summary()

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 32, 32, 64)	1792
batch_normalization (Batch Normalization)	(None, 32, 32, 64)	256
conv2d_7 (Conv2D)	(None, 32, 32, 64)	36928
batch_normalization_1 (Batch Normalization)	(None, 32, 32, 64)	256
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_5 (Dropout)	(None, 16, 16, 64)	0
conv2d_8 (Conv2D)	(None, 16, 16, 128)	73856
batch_normalization_2 (Batch Normalization)	(None, 16, 16, 128)	512
conv2d_9 (Conv2D)	(None, 16, 16, 128)	147584
batch_normalization_3 (Batch Normalization)	(None, 16, 16, 128)	512
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_6 (Dropout)	(None, 8, 8, 128)	0
conv2d_10 (Conv2D)	(None, 8, 8, 256)	295168
batch_normalization_4 (Batch Normalization)	(None, 8, 8, 256)	1024
conv2d_11 (Conv2D)	(None, 8, 8, 256)	590080
batch_normalization_5 (Batch Normalization)	(None, 8, 8, 256)	1024
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_7 (Dropout)	(None, 4, 4, 256)	0
flatten_1 (Flatten)	(None, 4096)	0
dense_3 (Dense)	(None, 256)	1048832
batch_normalization_6 (Batch Normalization)	(None, 256)	1024
dropout_8 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 128)	32896
batch_normalization_15 (Batch Normalization)	(None, 128)	512
dropout_9 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 10)	1290

=====  
Total params: 2233546 (8.52 MB)  
Trainable params: 2230986 (8.51 MB)  
Non-trainable params: 2560 (10.00 KB)

Los datos de validación y train para este entrenamiento no se deben escalar

```
x_val = x_train[-10000:]
# x_val_scaled = x_train_scaled[-10000:]
y_val = y_train[-10000:]

x_train = x_train[:-10000]
# x_train_scaled = x_train_scaled[:-10000]
y_train = y_train[:-10000]
```

Como se ha mencionado anteriormente, se añade la técnica de Data Augmentation para intentar mejorar el accuracy del modelo

```
[ ] train_datagen = ImageDataGenerator(
    rescale=1./255,

    # Definamos las transformaciones
    rotation_range=45,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)

train_generator = train_datagen.flow(
    x_train, # Aquí hay que usar datos NO re-escalados... de ahí que no usemos x_train_rescaled!
    y_train,
    batch_size=64 # ritmo razonable de imágenes
)

[ ] validation_datagen = ImageDataGenerator(
    rescale=1./255
)
validation_generator = validation_datagen.flow(
    x_val,
    y_val,
    batch_size= 64
)
```

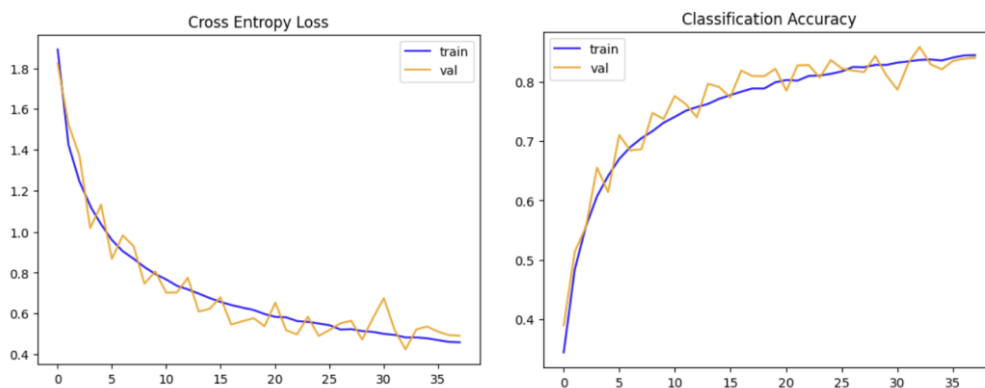
Se disminuye el tamaño del batch de 516 a 100. Al entrenar con un tamaño más pequeño se añade más variabilidad en los datos. Permite ayudar al modelo a generalizar mejor y evitar el sobreajuste.

```
▶ epochs = 100

history = model.fit(train_generator, epochs=epochs,
                    validation_data=validation_generator,
                    callbacks=[callback_loss, callback_acc],
                    steps_per_epoch = 625, # numero img train / batch size = 40000 / 64
                    validation_steps = 157 # numero img val / batch size val = 10000 / 64
                    )
```

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

> 83.620



El modelo tiene un buen rendimiento en entrenamiento y validación aunque con oscilaciones que indican un indicio de sobreajuste. La precisión del modelo es alta, ya que es de 83,62%, por lo tanto, el modelo está bien ajustado. Sin embargo, aún hay un margen de mejora que se podría explorar con otras técnicas, por ejemplo VGG16.

## 9. PROYECTO 9: 9\_cnn\_cifar10.ipynb → Accuracy = 62,76%

En el presente proyecto se implementará la arquitectura VGG16 por vez primera. Como paso inicial, se procederá a la carga de los pesos del modelo previamente entrenado con el conjunto de datos “ImageNet”, empleando para ello el parámetro “weights=imagenet”.

```
[ ] # Creemos una red que será extracción de features basada en VGG16
model_vgg16 = vgg16.VGG16(include_top = False,      # no quiero la salida
                           weights = 'imagenet',    #
                           input_shape = (32,32,3))
```

```
[ ] model_vgg16.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_11 (InputLayer)	[(None, 32, 32, 3)]	0
block1_conv1 (Conv2D)	(None, 32, 32, 64)	1792
block1_conv2 (Conv2D)	(None, 32, 32, 64)	36928
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	0
block2_conv1 (Conv2D)	(None, 16, 16, 128)	73856
block2_conv2 (Conv2D)	(None, 16, 16, 128)	147584
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	0
block3_conv1 (Conv2D)	(None, 8, 8, 256)	295168
block3_conv2 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv3 (Conv2D)	(None, 8, 8, 256)	590080
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	0
block4_conv1 (Conv2D)	(None, 4, 4, 512)	1180160
block4_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block4_conv3 (Conv2D)	(None, 4, 4, 512)	2359808
block4_pool (MaxPooling2D)	(None, 2, 2, 512)	0
block5_conv1 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv2 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv3 (Conv2D)	(None, 2, 2, 512)	2359808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0

-----  
Total params: 14714688 (56.13 MB)  
Trainable params: 14714688 (56.13 MB)  
Non-trainable params: 0 (0.00 Byte)

```
▶ # Añadir un Flatten en la última capa
output = model_vgg16.layers[-1].output
output_layer=ks.layers.Flatten()(output)
final_vgg16 = Model(model_vgg16.input,output_layer)
final_vgg16.summary()
```

Model: "model\_5"

Layer (type)	Output Shape	Param #
input_11 (InputLayer)	[(None, 32, 32, 3)]	0
block1_conv1 (Conv2D)	(None, 32, 32, 64)	1792
block1_conv2 (Conv2D)	(None, 32, 32, 64)	36928
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	0
block2_conv1 (Conv2D)	(None, 16, 16, 128)	73856
block2_conv2 (Conv2D)	(None, 16, 16, 128)	147584
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	0
block3_conv1 (Conv2D)	(None, 8, 8, 256)	295168
block3_conv2 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv3 (Conv2D)	(None, 8, 8, 256)	590080
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	0
block4_conv1 (Conv2D)	(None, 4, 4, 512)	1180160
block4_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block4_conv3 (Conv2D)	(None, 4, 4, 512)	2359808
block4_pool (MaxPooling2D)	(None, 2, 2, 512)	0
block5_conv1 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv2 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv3 (Conv2D)	(None, 2, 2, 512)	2359808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0
flatten_5 (Flatten)	(None, 512)	0

-----  
Total params: 14714688 (56.13 MB)  
Trainable params: 14714688 (56.13 MB)  
Non-trainable params: 0 (0.00 Byte)

```
[ ] # Veamos como va lo de "congelar" capas de entrenamiento
final_vgg16.trainable
```

True

En este primer intento, no se van a congelar los pesos de la arquitectura VGG16. Así que, durante el entrenamiento, los pesos de todas las capas del modelo se actualizarán y se ajustarán de acuerdo con los datos específicos del objetivo que deseamos alcanzar. Posteriormente, procesaremos todas las imágenes hasta el final de la arquitectura VGG16.

```
[ ] # llevamos las imnganes desde su origen al cuello de botella de VGG16, justo después del Flatten()
def llevarACuelloBotella (model, imagenes):
    return model.predict(imagenes)
```

```
[ ] x_train_vgg16 = llevarACuelloBotella (final_vgg16, x_train)
x_val_vgg16 = llevarACuelloBotella (final_vgg16, x_val)
x_test_vgg16 = llevarACuelloBotella (final_vgg16, x_test)
```

```
1250/1250 [=====] - 8s 7ms/step
313/313 [=====] - 2s 7ms/step
313/313 [=====] - 2s 7ms/step
```

```
[15] output_from_vgg16 = 512
```

```
[16] model_post_vgg = ks.Sequential()
```

```
model_post_vgg.add(ks.layers.InputLayer(input_shape=(output_from_vgg16)))
model_post_vgg.add(Dense(512, activation='relu', input_shape=(output_from_vgg16,)))
model_post_vgg.add(BatchNormalization())
model_post_vgg.add(Dense(512, activation='relu'))
model_post_vgg.add(BatchNormalization())
model_post_vgg.add(Dropout(0.3))
```

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

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
dense (Dense)	(None, 512)	262656
batch_normalization (Batch Normalization)	(None, 512)	2048
dense_1 (Dense)	(None, 512)	262656
batch_normalization_1 (Batch Normalization)	(None, 512)	2048
dropout (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130
=====		
Total params: 534538 (2.04 MB)		
Trainable params: 532490 (2.03 MB)		
Non-trainable params: 2048 (8.00 KB)		

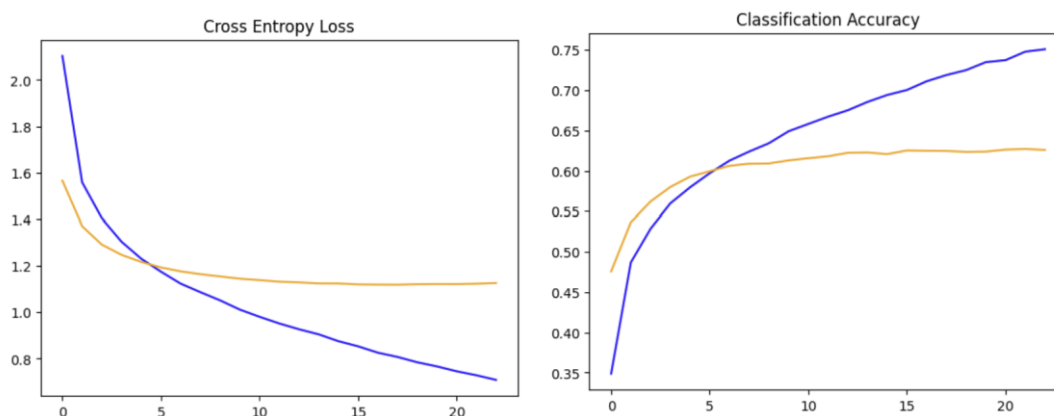
```
[17] new_adam = Adam(learning_rate=0.0001)
```

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

Se ha decidido definir un learning rate más bajo para el optimizador de Adam, de 0.0001, para hacer el modelo más complejo, de esta manera se converge de manera más estable aunque es más lenta.

```
[23] _, acc = model_post_vgg.evaluate(x_test_vgg16, y_test, verbose=0)
print('Modelo con Basic Transfer Learning > %.3f' % (acc * 100.0))
```

Modelo con Basic Transfer Learning > 62.760



Como se puede ver, aunque la arquitectura de VGG16 es más compleja que las anteriormente aplicadas, se observa que para el modelo de clasificación de CIFAR-10 los resultados no son buenos, ya que arquitecturas más simples muestran mejores resultados. En resumen, existe una gran pérdida durante el entrenamiento.

## 10. PROYECTO 10: 10\_cnn\_cifar10.ipynb → Accuracy = 77,61%

Se ha mantenido la misma arquitectura de red neuronal que en el caso anterior para VGG16. E contraposición, en este proyecto si se van a congelar los pesos, lo que significa que no se actualizarán durante el entrenamiento. En este caso se ha decidido hacer entrenables solamente las capas superiores, a partir de la capa “block3\_conv1”, por lo tanto, se están ajustando las capas más complejas de la red a nuevas características y se mantienen las características generales aprendidas previamente.

```
[ ] # Creemos una red que será extracción de features basada en VGG16
model_vgg16 = vgg16.VGG16(include_top = False,
                           weights = 'imagenet',
                           input_shape = (32,32,3))
```

```
model_vgg16.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 32, 32, 3)]	0
block1_conv1 (Conv2D)	(None, 32, 32, 64)	1792
block1_conv2 (Conv2D)	(None, 32, 32, 64)	36928
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	0
block2_conv1 (Conv2D)	(None, 16, 16, 128)	73856
block2_conv2 (Conv2D)	(None, 16, 16, 128)	147584
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	0
block3_conv1 (Conv2D)	(None, 8, 8, 256)	295168
block3_conv2 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv3 (Conv2D)	(None, 8, 8, 256)	590080
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	0
block4_conv1 (Conv2D)	(None, 4, 4, 512)	1180160
block4_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block4_conv3 (Conv2D)	(None, 4, 4, 512)	2359808
block4_pool (MaxPooling2D)	(None, 2, 2, 512)	0
block5_conv1 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv2 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv3 (Conv2D)	(None, 2, 2, 512)	2359808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0
Total params: 14714688 (56.13 MB)		
Trainable params: 14714688 (56.13 MB)		
Non-trainable params: 0 (0.00 Byte)		

```
# Añadir un Flatten en la última capa
output = model_vgg16.layers[-1].output
output_layer=ks.layers.Flatten()(output)
final_vgg16 = Model(model_vgg16.input,output_layer)
final_vgg16.summary()
```

Model: "model\_3"

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 32, 32, 3)]	0
block1_conv1 (Conv2D)	(None, 32, 32, 64)	1792
block1_conv2 (Conv2D)	(None, 32, 32, 64)	36928
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	0
block2_conv1 (Conv2D)	(None, 16, 16, 128)	73856
block2_conv2 (Conv2D)	(None, 16, 16, 128)	147584
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	0
block3_conv1 (Conv2D)	(None, 8, 8, 256)	295168
block3_conv2 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv3 (Conv2D)	(None, 8, 8, 256)	590080
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	0
block4_conv1 (Conv2D)	(None, 4, 4, 512)	1180160
block4_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block4_conv3 (Conv2D)	(None, 4, 4, 512)	2359808
block4_pool (MaxPooling2D)	(None, 2, 2, 512)	0
block5_conv1 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv2 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv3 (Conv2D)	(None, 2, 2, 512)	2359808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0
flatten_3 (Flatten)	(None, 512)	0
Total params: 14714688 (56.13 MB)		
Trainable params: 14714688 (56.13 MB)		
Non-trainable params: 0 (0.00 Byte)		

```
[ ] # Veamos como va lo de "congelar" capas de entrenamiento
final_vgg16.trainable
```

True

```
[ ] pd.set_option("max_colwidth", True)
layers = [(layer, layer.name, layer.trainable) for layer in final_vgg16.layers]
pd.DataFrame(layers, columns=("Layer", "Layer Name", "Is Trainable?"))
```

	Layer	Layer Name	Is Trainable?
0	<keras.src.engine.input_layer.InputLayer object at 0x7cf0ca86fd30>	input_4	True
1	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf0ca86d330>	block1_conv1	True
2	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf0b01c7cd0>	block1_conv2	True
3	<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7cf0523d0520>	block1_pool	True
4	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf0523d01f0>	block2_conv1	True
5	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf052346470>	block2_conv2	True
6	<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7cf11f2709a0>	block2_pool	True
7	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf0b01c5360>	block3_conv1	True
8	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f272050>	block3_conv2	True
9	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f272860>	block3_conv3	True
10	<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7cf11f273790>	block3_pool	True
11	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f273310>	block4_conv1	True
12	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f2728f0>	block4_conv2	True
13	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f2991e0>	block4_conv3	True
14	<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7cf11f298af0>	block4_pool	True
15	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f29ad10>	block5_conv1	True
16	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f29b430>	block5_conv2	True
17	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f29b160>	block5_conv3	True
18	<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7cf11f2a11b0>	block5_pool	True
19	<keras.src.layers.resizing.flatten.Flatten object at 0x7cf11f2a3580>	flatten_3	True

```
[ ] entrenable = False
for layer in final_vgg16.layers:
    if layer.name == "block3_conv1":
        entrenable = True
layer.trainable = entrenable
```

```
[ ] layers = [(layer, layer.name, layer.trainable) for layer in final_vgg16.layers]
pd.DataFrame(layers, columns=("Layer", "Layer Name", "Is Trainable?"))
```

	Layer	Layer Name	Is Trainable?
0	<keras.src.engine.input_layer.InputLayer object at 0x7cf0ca86fd30>	input_4	False
1	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf0ca86d330>	block1_conv1	False
2	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf0b01c7cd0>	block1_conv2	False
3	<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7cf0523d0520>	block1_pool	False
4	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf0523d01f0>	block2_conv1	False
5	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf052346470>	block2_conv2	False
6	<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7cf11f2709a0>	block2_pool	False
7	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf0b01c5360>	block3_conv1	True
8	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f272050>	block3_conv2	True
9	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f272860>	block3_conv3	True
10	<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7cf11f273790>	block3_pool	True
11	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f273310>	block4_conv1	True
12	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f2728f0>	block4_conv2	True
13	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f2991e0>	block4_conv3	True
14	<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7cf11f298af0>	block4_pool	True
15	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f29ad10>	block5_conv1	True
16	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f29b430>	block5_conv2	True
17	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7cf11f29b160>	block5_conv3	True
18	<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7cf11f2a11b0>	block5_pool	True
19	<keras.src.layers.resizing.flatten.Flatten object at 0x7cf11f2a3580>	flatten_3	True



```
[ ] model_post_vgg = ks.Sequential()

model_post_vgg.add(final_vgg16)
model_post_vgg.add(Dense(512, activation='relu'))
model_post_vgg.add(BatchNormalization())
model_post_vgg.add(Dense(512, activation='relu'))
model_post_vgg.add(BatchNormalization())
model_post_vgg.add(Dropout(0.3))

model_post_vgg.add(Dense(10, activation='softmax'))

model_post_vgg.summary()
```

Model: "sequential\_3"

Layer (type)	Output Shape	Param #
model_3 (Functional)	(None, 512)	14714688
dense_9 (Dense)	(None, 512)	262656
batch_normalization_6 (Batch Normalization)	(None, 512)	2048
dense_10 (Dense)	(None, 512)	262656
batch_normalization_7 (Batch Normalization)	(None, 512)	2048
dropout_3 (Dropout)	(None, 512)	0
dense_11 (Dense)	(None, 10)	5130

=====  
Total params: 15249226 (58.17 MB)  
Trainable params: 14987018 (57.17 MB)  
Non-trainable params: 262208 (1.00 MB)

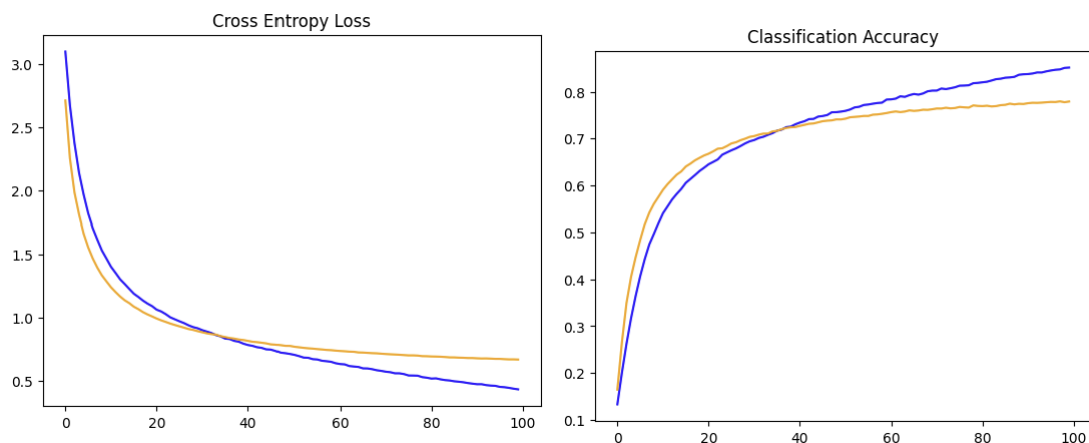
```
[ ] new_adam = Adam(learning_rate=0.000001)
```

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

Se ha tenido que ajustar la tasa de aprendizaje (learnign rate) a 0.000001, ya que durante el entrenamiento el val\_accuracy no se obtenía mejoras significativas. De esta manera permite que el modelo alcance un mayor nivel de precisión.

```
[ ] _, acc = model_post_vgg.evaluate(x_test, y_test, verbose=0)
print('Modelo con Basic Transfer Learning > %.3f' % (acc * 100.0))
```

Modelo con Basic Transfer Learning > 77.610



Se constata que en comparación con el proyecto anterior que comparten la misma arquitectura, la congelación de pesos mejora el modelo, sin embargo, se han obtenido mejores resultados con unas arquitecturas más simples y sin necesidad de estar preentrenadas.