Proyectos CIFAR10

DS-0921

Marc Campmany

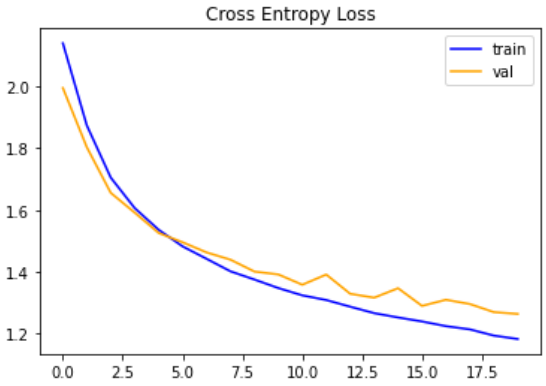
NUCLIO
DIGITAL SCHOOL

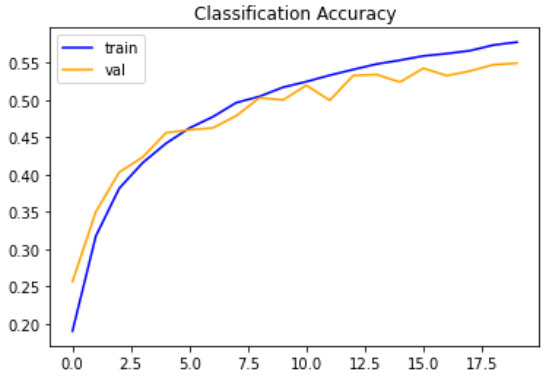
Proyecto 0 (base)

```
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'))
```

En set de Test:

> 54.700






```
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: 263,402
Trainable params: 263,402
Non-trainable params: 0
```

El coste podemos ver que es estable y es en el epoch 15 que se fija en un valor. Además la accuracy baila alrededor de 0.10.

No se puede decir que sea un modelo muy bueno.

Con más epochs podría subir un moco más la accuracy ya que el entreno se detiene demasiado pronto. Hay poco overfitting.

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

```
history = model.fit(x_train_scaled, y_train, epochs=20,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

```
Epoch 20/20
79/79 [==============================] - 1s 9ms/step - loss: 1.1815 - accuracy: 0.5777 - val_loss: 1.2629 - val_accuracy: 0.5494
```

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

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

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 32, 32, 32)        896

 max_pooling2d_5 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_6 (Conv2D)           (None, 16, 16, 32)        9248

 max_pooling2d_6 (MaxPooling  (None, 8, 8, 32)         0
 2D)

 flatten_3 (Flatten)         (None, 2048)              0

 dense_6 (Dense)             (None, 32)                65568

 dense_7 (Dense)             (None, 10)                330

=================================================================
Total params: 76,042
Trainable params: 76,042
Non-trainable params: 0
```

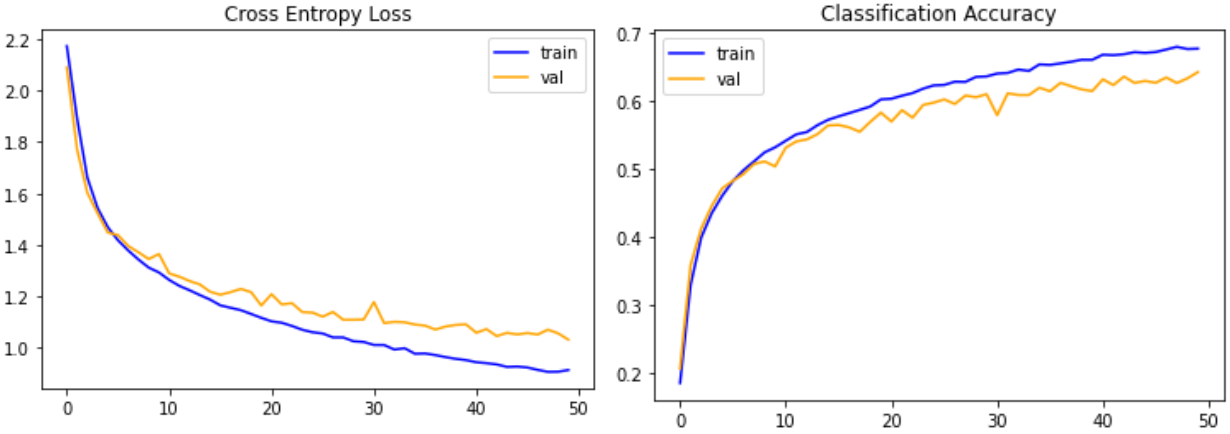
```
model.compile(optimizer='Adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

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

```
Epoch 49/50
79/79 [==============================] - 1s 10ms/step - loss: 0.9065 - accuracy: 0.6759 - val_loss: 1.0557 - val_accuracy: 0.6328
Epoch 50/50
79/79 [==============================] - 1s 10ms/step - loss: 0.9129 - accuracy: 0.6765 - val_loss: 1.0313 - val_accuracy: 0.6421
```

En set de test:

> 64.330



Se han aumentado los epochs hasta 50 para permitir un entreno más largo y se ha añadido una Conv2d+Maxpool para mejorar la obtención de Features en las capas de aprendizaje. Vemos que la accuracy en Test ha mejorado notablemente, en un 10%. Se observa que los epochs podrían aumentarse aún más ya que parece que sigue en tendencia creciente la accuracy.

El overfitting se ha mantenido bastante bien, aún sin añadir ninguna capa específica para controlarlo como sería un dropout.

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


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


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

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, 32)	16416
dense_1 (Dense)	(None, 10)	330

```
Total params: 36,138
Trainable params: 36,138
Non-trainable params: 0
```

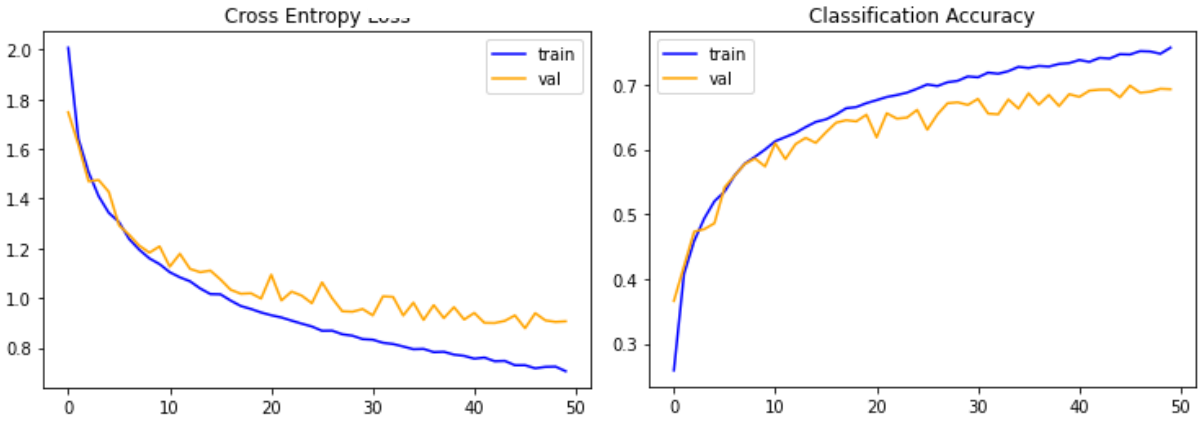
```
history = model.fit(x_train_scaled, y_train, epochs=50,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

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

```
Epoch 50/50
79/79 [==============================] - 1s 14ms/step - loss: 0.7052 - accuracy: 0.7570 - val_loss: 0.9062 - val_accuracy: 0.6928
```

En set de test:

> 68.870



El modelo sigue mejorando pero el overfitting va en aumento también. Parece que la parte de aprendizaje de características va mejorando con cada capa nueva de Conv2d+Maxpool.

Con más capas de convolución, se extraen features más específicos de texturas, colores, bordes, etc. Por tanto, el modelo debería poder clasificar mejor al tener más información extraída de las imágenes.

La idea será probar alguna capa más y después modificar el número de kernels en cada capa para ver cómo afecta.

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


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


model.add(ks.layers.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'))
```

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d_3 (MaxPooling 2D)	(None, 16, 16, 32)	0
conv2d_4 (Conv2D)	(None, 16, 16, 32)	9248
max_pooling2d_4 (MaxPooling 2D)	(None, 8, 8, 32)	0
conv2d_5 (Conv2D)	(None, 8, 8, 32)	9248
max_pooling2d_5 (MaxPooling 2D)	(None, 4, 4, 32)	0
conv2d_6 (Conv2D)	(None, 4, 4, 32)	9248
max_pooling2d_6 (MaxPooling 2D)	(None, 2, 2, 32)	0
flatten_1 (Flatten)	(None, 128)	0
dense_2 (Dense)	(None, 32)	4128
dense_3 (Dense)	(None, 10)	330

Total params: 33,098
Trainable params: 33,098
Non-trainable params: 0

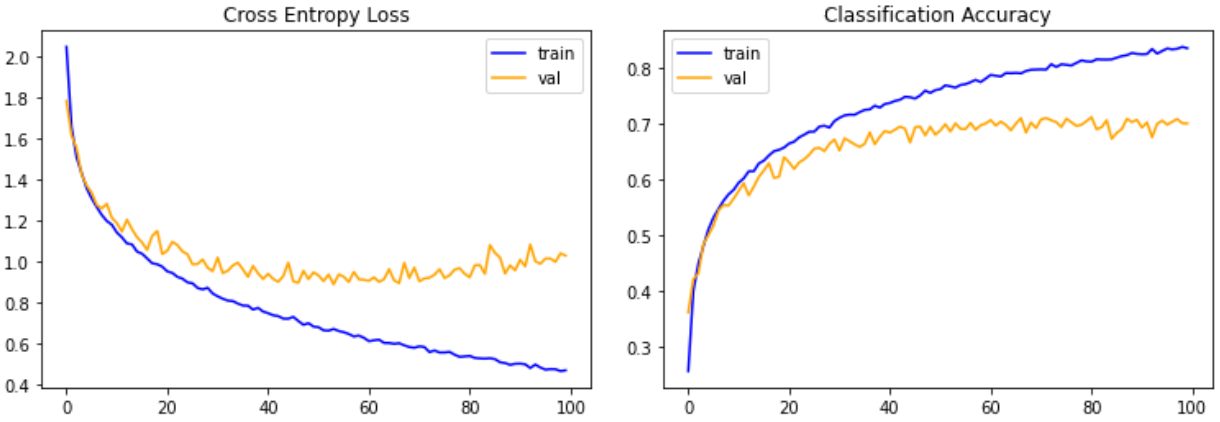
```
history = model.fit(x_train_scaled, y_train, epochs=100,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val))
```

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

```
Epoch 100/100
79/79 [==============================] - 1s 15ms/step - loss: 0.4689 - accuracy: 0.8343 - val_loss: 1.0295 - val_accuracy: 0.6997
```

En set de test:

> 69.250



El modelo ha aumentado un poco la accuracy pero vemos que al aumentar los epochs de entrenamiento, sin especificar un callback de Early Stopping, el modelo overfittea mucho. Puede que al no estar modificando los kernels en cada convolución, se están estrayendo los mismo features pero cada vez con menos información relevante ya que es sobre imágenes más reducidas.

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

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

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

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

Layer (type)	Output Shape	Param #
conv2d_36 (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d_36 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_37 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_37 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_38 (Conv2D)	(None, 8, 8, 128)	73856
max_pooling2d_38 (MaxPooling2D)	(None, 4, 4, 128)	0
conv2d_39 (Conv2D)	(None, 4, 4, 256)	295168
max_pooling2d_39 (MaxPooling2D)	(None, 2, 2, 256)	0
flatten_9 (Flatten)	(None, 1024)	0
dense_18 (Dense)	(None, 32)	32800
dense_19 (Dense)	(None, 10)	330

Total params: 421,546
Trainable params: 421,546
Non-trainable params: 0

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10)
```

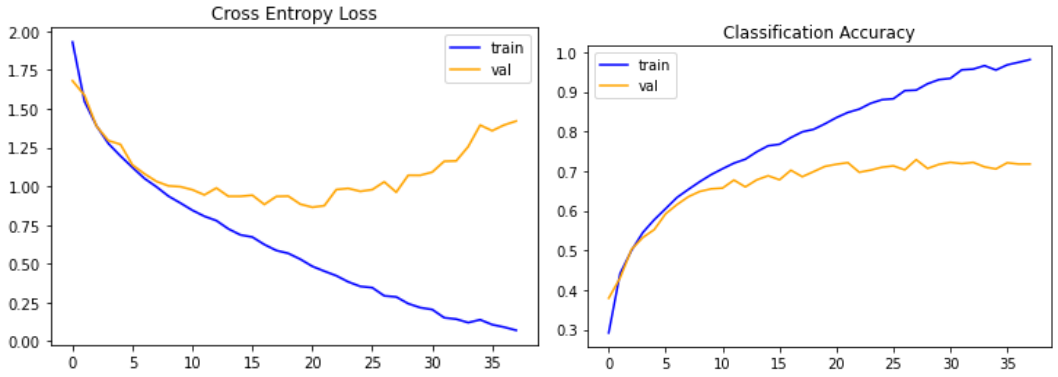
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_accuracy])
```

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

```
Epoch 38/200
79/79 [==============================] - 2s 22ms/step - loss: 0.0706 - accuracy: 0.9815 - val_loss: 1.4202 - val_accuracy: 0.7179
```

En set de test:

> 71.650



Se han modificado los filtros de cada capa de convolución de manera ascendente. Se puede apreciar que el overfitting se nos ha ido de las manos en este proyecto.

Al hacerlo con nº de kernels creciente, hacemos que en cada nueva capa de convolución se apliquen más filtros y por tanto se extraiga más detalle de características específicas del output anterior como colores, texturas, etc. en cada cada. Para ayudar a clasificar mejor.

Proyecto 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.Dropout(0.2))

model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
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'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

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

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

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

Layer (type)	Output Shape	Param #
conv2d_80 (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d_80 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_5 (Dropout)	(None, 16, 16, 32)	0
conv2d_81 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_81 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_6 (Dropout)	(None, 8, 8, 64)	0
conv2d_82 (Conv2D)	(None, 8, 8, 128)	73856
max_pooling2d_82 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_7 (Dropout)	(None, 4, 4, 128)	0
conv2d_83 (Conv2D)	(None, 4, 4, 256)	295168
max_pooling2d_83 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_8 (Dropout)	(None, 2, 2, 256)	0
flatten_20 (Flatten)	(None, 1024)	0
dense_47 (Dense)	(None, 32)	32800
dense_48 (Dense)	(None, 10)	330

Total params: 421,546
Trainable params: 421,546
Non-trainable params: 0

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=10)
```

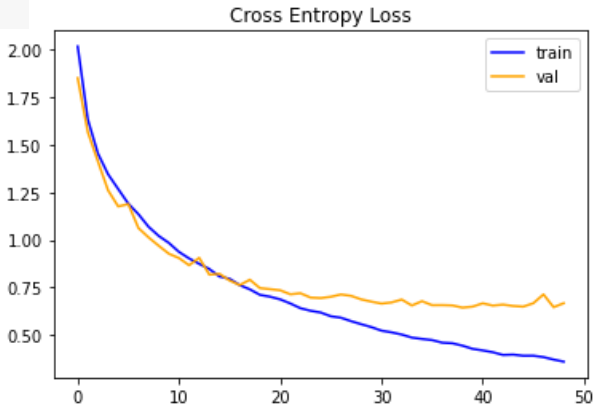
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

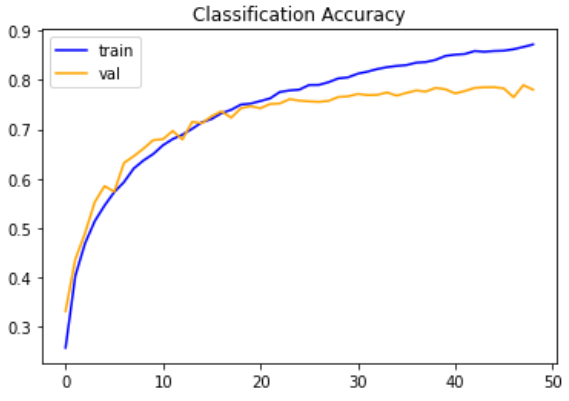
```
model.compile(optimizer='Adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
Epoch 49/200
79/79 [==============================] - 2s 25ms/step - loss: 0.3603 - accuracy: 0.8722 - val_loss: 0.6676 - val_accuracy: 0.7805
```

En set de test:

> 77.870





En este proyecto se ha tratado el overfitting. Principalmente se han aplicado capas de dropout después de cada MaxPooling de un 20% y además se ha agregado un early stopping asociado al validation_loss ya que en el proyecto anterior se disparaba.

Las capas de dropout han ayudado muchísimo a forzar el modelo a generalizar las predicciones y no sobreajustar el modelo al set de training. Con esto el accuracy de train obviamente ha bajado pero el de validación y test ha crecido muchísimo.

Proyecto 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.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
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'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

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

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

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

Layer (type)	Output Shape	Param #
conv2d_84 (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d_84 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_9 (Dropout)	(None, 16, 16, 32)	0
conv2d_85 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_85 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_10 (Dropout)	(None, 8, 8, 64)	0
conv2d_86 (Conv2D)	(None, 8, 8, 128)	73856
max_pooling2d_86 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_11 (Dropout)	(None, 4, 4, 128)	0
conv2d_87 (Conv2D)	(None, 4, 4, 256)	295168
max_pooling2d_87 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_12 (Dropout)	(None, 2, 2, 256)	0
flatten_21 (Flatten)	(None, 1024)	0
dense_49 (Dense)	(None, 64)	65600
dense_50 (Dense)	(None, 10)	650

Total params: 454,666
Trainable params: 454,666
Non-trainable params: 0

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=10)
```

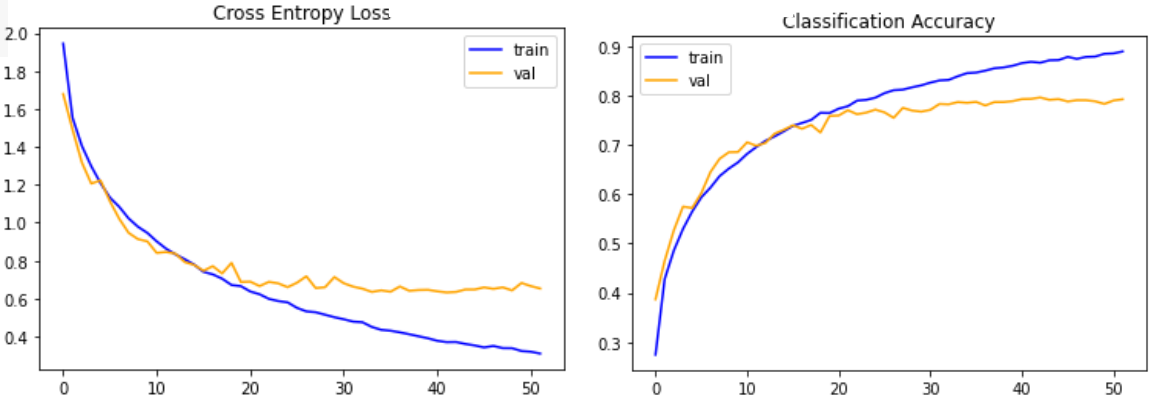
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

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

```
Epoch 52/200
79/79 [==============================] - 2s 25ms/step - loss: 0.3097 - accuracy: 0.8892 - val_loss: 0.6530 - val_accuracy: 0.7923
```

En set de test:

> 78.680



Hemos aumentado los nodos de la capa Fully Connected para mejorar la clasificación de las imágenes. La detección de características se mantiene igual ya que no hemos tocado nada de la parte de Aprendizaje de Features, pero aumentando las neuronas de la capa dense aumentamos el nivel de complejidad y relaciones que puede formular el modelo para asignar las categorías y predecir mejor cada imagen. Se ha probado con 128 y 512 neuronas y el resultado empeoraba así que hay un límite de complejidad que sea beneficioso.

Parece que para el train podría seguir más el entrenamiento así que para la siguiente aumentaremos el dropout para reducir overfitting y forzar generalización.

Proyecto 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(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(0.2))

model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
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'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

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

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

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

Layer (type)	Output Shape	Param #
conv2d_116 (Conv2D)	(None, 32, 32, 32)	896
conv2d_117 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_116 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_41 (Dropout)	(None, 16, 16, 32)	0
conv2d_118 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_117 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_42 (Dropout)	(None, 8, 8, 64)	0
conv2d_119 (Conv2D)	(None, 8, 8, 128)	73856
max_pooling2d_118 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_43 (Dropout)	(None, 4, 4, 128)	0
conv2d_120 (Conv2D)	(None, 4, 4, 256)	295168
max_pooling2d_119 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_44 (Dropout)	(None, 2, 2, 256)	0
flatten_29 (Flatten)	(None, 1024)	0
dense_66 (Dense)	(None, 64)	65600
dense_67 (Dense)	(None, 10)	650

Total params: 463,914
Trainable params: 463,914
Non-trainable params: 0

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=10)
```

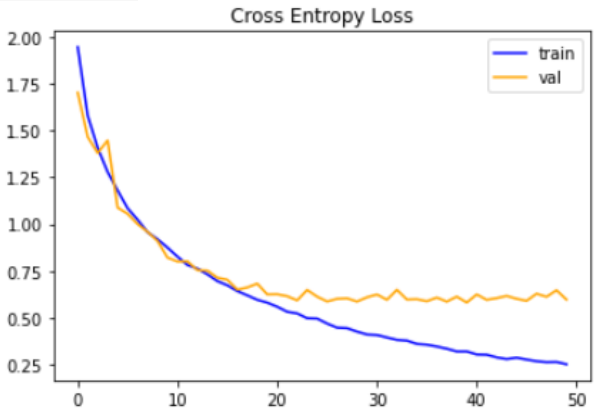
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

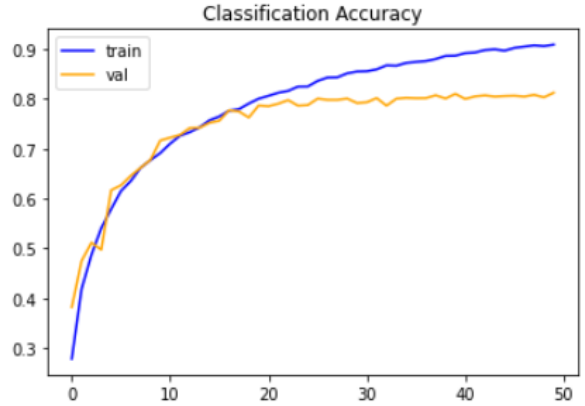
```
model.compile(optimizer='Adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
Epoch 50/200
79/79 [==============================] - 3s 38ms/step - loss: 0.2495 - accuracy: 0.9085 - val_loss: 0.5957 - val_accuracy: 0.8121
```

En set de test:

> 80.460





Como se comentaba en el proyecto anterior, se ha probado de aumentar la densidad de neuronas de la capa de clasificación pero no mejoraba el accuracy así que se ha optado por aumentar la capacidad de extracción de Features.

Se ha incluido una nueva capa de convolución en la primera etapa, replicando los modelos vgg donde las capas de convolución se apilan directamente entre sí con el mismo número de features. Al apilar las capas de convolución entre sí sin la reducció por maxpooling, se mantiene mejor la relación espacial de cada característica extraída por los filtros.

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

model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu', padding='same'))
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'))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu', padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

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

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

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

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_77 (Conv2D)          (None, 32, 32, 32)        896

 conv2d_78 (Conv2D)          (None, 32, 32, 32)        9248

 max_pooling2d_48 (MaxPoolin  (None, 16, 16, 32)       0
 g2D)

 dropout_48 (Dropout)        (None, 16, 16, 32)        0

 conv2d_79 (Conv2D)          (None, 16, 16, 64)        18496

 conv2d_80 (Conv2D)          (None, 16, 16, 64)        36928

 max_pooling2d_49 (MaxPoolin  (None, 8, 8, 64)         0
 g2D)

 dropout_49 (Dropout)        (None, 8, 8, 64)          0

 conv2d_81 (Conv2D)          (None, 8, 8, 128)         73856

 conv2d_82 (Conv2D)          (None, 8, 8, 128)         147584

 max_pooling2d_50 (MaxPoolin  (None, 4, 4, 128)        0
 g2D)

 dropout_50 (Dropout)        (None, 4, 4, 128)         0

 conv2d_83 (Conv2D)          (None, 4, 4, 256)         295168

 conv2d_84 (Conv2D)          (None, 4, 4, 256)         590080

 max_pooling2d_51 (MaxPoolin  (None, 2, 2, 256)        0
 g2D)

 dropout_51 (Dropout)        (None, 2, 2, 256)         0

 flatten_12 (Flatten)        (None, 1024)              0

 dense_24 (Dense)            (None, 64)                65600

 dense_25 (Dense)            (None, 10)                650

=================================================================
Total params: 1,238,506
Trainable params: 1,238,506
Non-trainable params: 0
```

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=10)
```

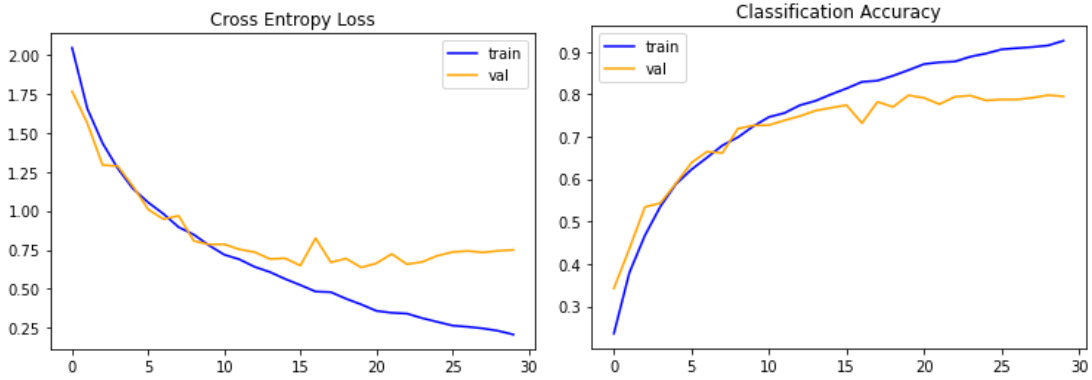
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

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

```
Epoch 30/200
79/79 [==============================] - 3s 32ms/step - loss: 0.2078 - accuracy: 0.9261 - val_loss: 0.7499 - val_accuracy: 0.7949
```

En set de test:

> 78.600



Al poner 2 conv2D seguidas en cada capa de convolución de la parte de extracción de features, la accuracy del set de test empeora. Viendo las gráficas parece principalmente porque vuelve a aparecer Overfitting así que sería necesario tunear los valores de los dropouts y aumentarlos.

Como se comentaba en el modelo anterior, la capacidad de captar features ha aumentado y mejora con este cambio. El problema es que en el modelo anterior aún no se podía observar overfitting ya que la dupliación de convoluciones se había hecho con la capa de solo 16 filtros y por tanto no se apreciaba el overfitting tanto como en este proyecto.

Proyecto 9a

```
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(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(0.3))

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

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

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

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

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

Layer (type)	Output Shape	Param #
conv2d_148 (Conv2D)	(None, 32, 32, 32)	896
conv2d_149 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_136 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_61 (Dropout)	(None, 16, 16, 32)	0
conv2d_150 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_151 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_137 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_62 (Dropout)	(None, 8, 8, 64)	0
conv2d_152 (Conv2D)	(None, 8, 8, 128)	73856
conv2d_153 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_138 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_63 (Dropout)	(None, 4, 4, 128)	0
conv2d_154 (Conv2D)	(None, 4, 4, 256)	295168
conv2d_155 (Conv2D)	(None, 4, 4, 256)	590080
max_pooling2d_139 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_64 (Dropout)	(None, 2, 2, 256)	0
flatten_34 (Flatten)	(None, 1024)	0
dense_76 (Dense)	(None, 64)	65600
dense_77 (Dense)	(None, 10)	650

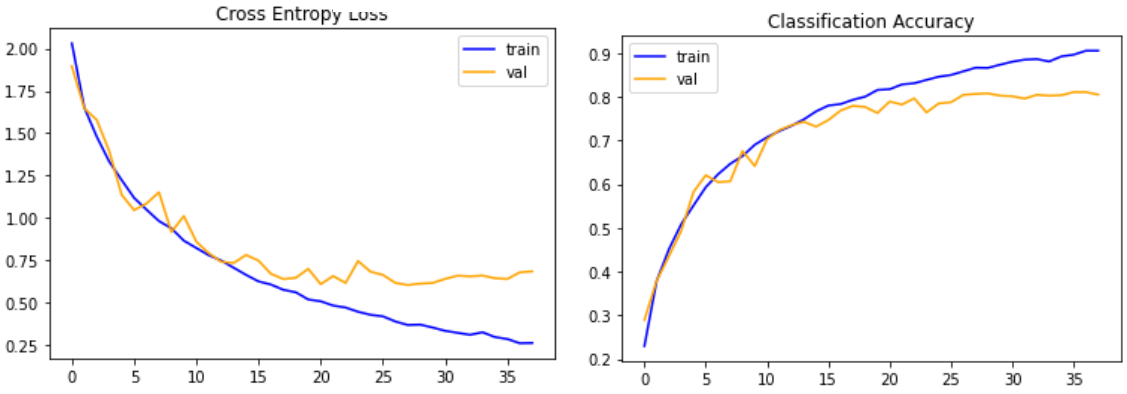
Total params: 1,238,506
Trainable params: 1,238,506
Non-trainable params: 0

```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

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

En set de test:

> 80.070



Se ha aumentado el porcentaje de dropout para tratar de controlar el overfitting. Explicado en siguiente slide.

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=10)
```

Proyecto 9b

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

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

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

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

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

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

```
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_141 (Conv2D)         (None, 32, 32, 32)        896

 conv2d_142 (Conv2D)         (None, 32, 32, 32)        9248

 max_pooling2d_80 (MaxPoolin  (None, 16, 16, 32)       0
 g2D)

 dropout_80 (Dropout)        (None, 16, 16, 32)        0

 conv2d_143 (Conv2D)         (None, 16, 16, 64)        18496

 conv2d_144 (Conv2D)         (None, 16, 16, 64)        36928

 max_pooling2d_81 (MaxPoolin  (None, 8, 8, 64)         0
 g2D)

 dropout_81 (Dropout)        (None, 8, 8, 64)          0

 conv2d_145 (Conv2D)         (None, 8, 8, 128)         73856

 conv2d_146 (Conv2D)         (None, 8, 8, 128)         147584

 max_pooling2d_82 (MaxPoolin  (None, 4, 4, 128)        0
 g2D)

 dropout_82 (Dropout)        (None, 4, 4, 128)         0

 conv2d_147 (Conv2D)         (None, 4, 4, 256)         295168

 conv2d_148 (Conv2D)         (None, 4, 4, 256)         590080

 max_pooling2d_83 (MaxPoolin  (None, 2, 2, 256)        0
 g2D)

 dropout_83 (Dropout)        (None, 2, 2, 256)         0

 flatten_20 (Flatten)        (None, 1024)              0

 dense_40 (Dense)            (None, 64)                65600

 dense_41 (Dense)            (None, 10)                650

=================================================================
Total params: 1,238,506
Trainable params: 1,238,506
Non-trainable params: 0
```

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=10)
```

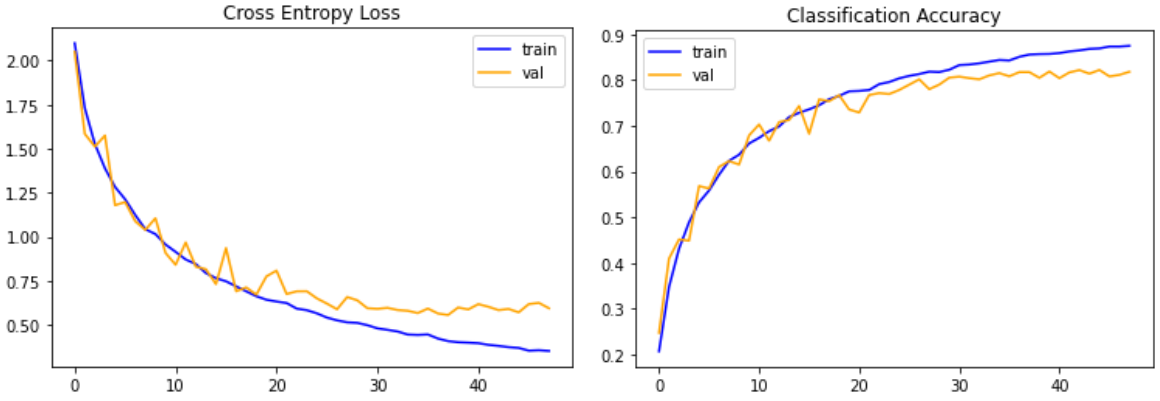
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

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

```
Epoch 48/200
79/79 [==============================] - 3s 33ms/step - loss: 0.2999 - accuracy: 0.8937 - val_loss: 0.5636 - val_accuracy: 0.8353
```

En set de test:

> 81.320



Para este proyecto realmente se han probado muchas combinaciones diferentes de dropout para poder mejorar el overfitting de la mejor manera. Se ha probado con el mismo % de dropout en todas las capas, con diferentes valores mayores, con valores ascendentes, etc.

La estrategia que ha dado mejores resultados ha sido la de aplicar un dropout mayor en cada capa e ir probando diferentes combinaciones de valores crecientes. Sorprende que el mejor resultado era con el 50% de desactivación de la última capa de dropout.

Parece que hay una relación entre el nombre de kernels de las capas de convolución y el overfitting que generan, por lo que es necesario más dropout para generalizar la solución. Con más kernels el nivel de detallede lo features debe ser mayor y por tanto el modelo debe aprender más características específicas de train que hace falta “romper” con el dropout para generalizar mejor.

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

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

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

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

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

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

Layer (type)	Output Shape	Param #
conv2d_24 (Conv2D)	(None, 32, 32, 32)	896
conv2d_25 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_12 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_12 (Dropout)	(None, 16, 16, 32)	0
conv2d_26 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_27 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_13 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_13 (Dropout)	(None, 8, 8, 64)	0
conv2d_28 (Conv2D)	(None, 8, 8, 128)	73856
conv2d_29 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_14 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_14 (Dropout)	(None, 4, 4, 128)	0
conv2d_30 (Conv2D)	(None, 4, 4, 256)	295168
conv2d_31 (Conv2D)	(None, 4, 4, 256)	590080
max_pooling2d_15 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_15 (Dropout)	(None, 2, 2, 256)	0
flatten_3 (Flatten)	(None, 1024)	0
dense_7 (Dense)	(None, 64)	65600
dense_8 (Dense)	(None, 64)	4160
dense_9 (Dense)	(None, 10)	650

```
Total params: 1,242,666
Trainable params: 1,242,666
Non-trainable params: 0
```

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=10)
```

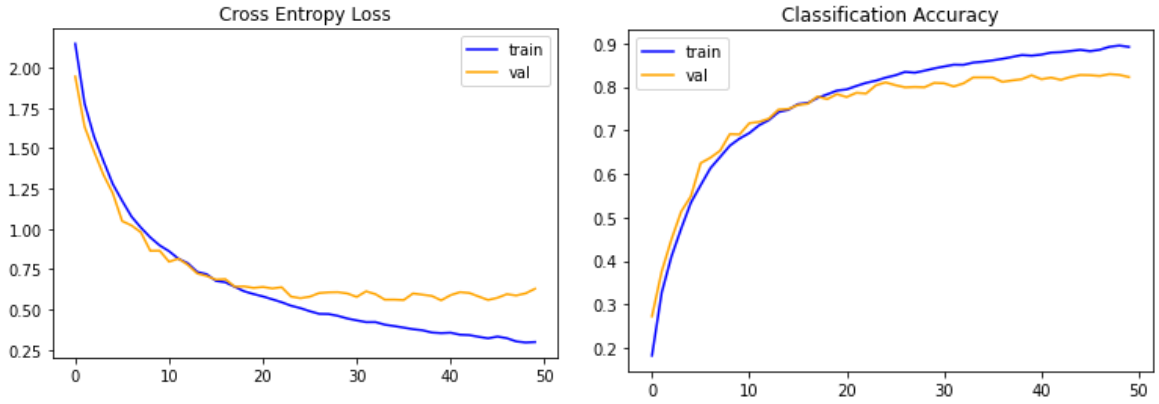
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

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

```
Epoch 50/200
79/79 [==============================] - 3s 33ms/step - loss: 0.2983 - accuracy: 0.8917 - val_loss: 0.6286 - val_accuracy: 0.8223
```

En set de test:

> 81.540



Añadiendo una capa extra para la parte de clasificación vemos que las curvas de validación son mucho más suaves y sin los picos del proyecto anterior por lo que es una buena mejora.

Con este proyecto hacemos la parte de clasificación de la CNN más profunda (y ahora que tenemos la parte de aprendizaje de características mucho mayor y compleja) era necesario aumentar la capacidad de clasificación de la CNN para poder crear más relaciones y relaciones más complicadas para clasificar mejor las imágenes.

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

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

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

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

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

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

Layer (type)	Output Shape	Param #
conv2d_48 (Conv2D)	(None, 32, 32, 32)	896
conv2d_49 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_24 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_36 (Dropout)	(None, 16, 16, 32)	0
conv2d_50 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_51 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_25 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_37 (Dropout)	(None, 8, 8, 64)	0
conv2d_52 (Conv2D)	(None, 8, 8, 128)	73856
conv2d_53 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_26 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_38 (Dropout)	(None, 4, 4, 128)	0
conv2d_54 (Conv2D)	(None, 4, 4, 256)	295168
conv2d_55 (Conv2D)	(None, 4, 4, 256)	590080
max_pooling2d_27 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_39 (Dropout)	(None, 2, 2, 256)	0
flatten_6 (Flatten)	(None, 1024)	0
dense_18 (Dense)	(None, 512)	524800
dense_19 (Dense)	(None, 512)	262656
dense_20 (Dense)	(None, 10)	5130

Total params: 1,964,842
Trainable params: 1,964,842
Non-trainable params: 0

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=10)
```

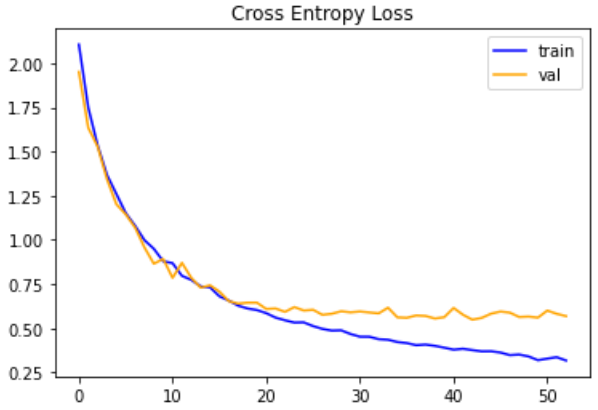
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

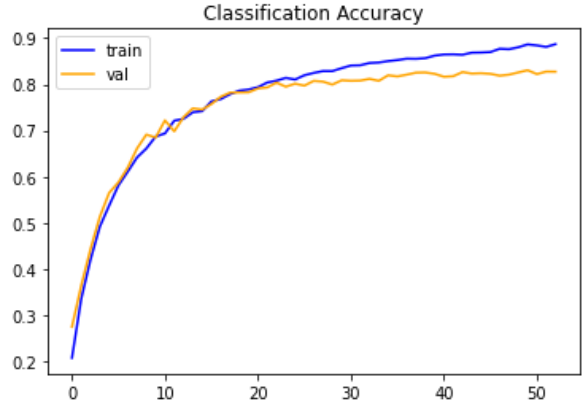
```
model.compile(optimizer='Adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
Epoch 53/200
79/79 [==============================] - 4s 49ms/step - loss: 0.3170 - accuracy: 0.8869 - val_loss: 0.5684 - val_accuracy: 0.8273
```

En set de test:

> 81.930





Se han aumentado las neuronas de las capas de clasificación para aumentar las relaciones internas que puede generar el modelo para clasificar las imágenes. Ha mejorado la accuracy y el modelo no presenta demasiado overfitting.

Se ha probado añadiendo dropouts en las capas finales de clasificación y el modelo emperoaba. Como se observa no hay un overfitting tan claro como con proyectos previos, así que estábamos “cargándonos” poder de predicción añadiendo más dropouts.

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

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

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

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

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

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

Layer (type)	Output Shape	Param #
conv2d_80 (Conv2D)	(None, 32, 32, 32)	896
conv2d_81 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d_40 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_55 (Dropout)	(None, 16, 16, 32)	0
conv2d_82 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_83 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_41 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_56 (Dropout)	(None, 8, 8, 64)	0
conv2d_84 (Conv2D)	(None, 8, 8, 128)	73856
conv2d_85 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_42 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_57 (Dropout)	(None, 4, 4, 128)	0
conv2d_86 (Conv2D)	(None, 4, 4, 256)	295168
conv2d_87 (Conv2D)	(None, 4, 4, 256)	590080
max_pooling2d_43 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_58 (Dropout)	(None, 2, 2, 256)	0
flatten_10 (Flatten)	(None, 1024)	0
dense_30 (Dense)	(None, 512)	524800
dense_31 (Dense)	(None, 512)	262656
dense_32 (Dense)	(None, 10)	5130

```
Total params: 1,964,970
Trainable params: 1,964,906
Non-trainable params: 64
```

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=10)
```

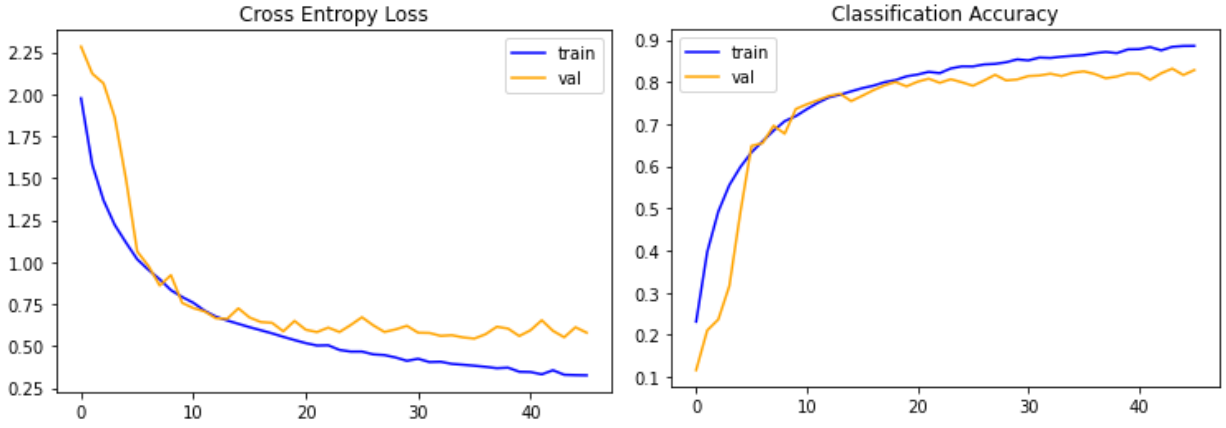
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

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

```
Epoch 46/200
79/79 [==============================] - 4s 51ms/step - loss: 0.3251 - accuracy: 0.8855 - val_loss: 0.5789 - val_accuracy: 0.8281
```

En set de test:

> 82.430



Se ha añadido una capa de Batch normalization para estandarizar/normalizar las salidas de las capas convolucionales y evitar la explosión de valores extremos. Esto ha acelerado el entrenamiento y ha mejorado la accuracy de train final, supongo que por trabajar inicialmente con valores menores y estandarizados.

Se ha probado de añadir más capas de Batch Normalization en las otras capas convolucionales del modelo, pero era contraproducente. El modelo empeoraba y las líneas de loss y accuracy eran mucho más irregulares. Normalizar los datos en cada convolución en modelos no muy profundos como este, no tiene sentido ya que está pensado para redes muy profundas en las que la explosión de los valores afecta más.

Proyecto 13

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

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

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

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dense(512, activation='relu'))

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

Layer (type)	Output Shape	Param #
conv2d_176 (Conv2D)	(None, 32, 32, 32)	896
conv2d_177 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_25 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d_88 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_103 (Dropout)	(None, 16, 16, 32)	0
conv2d_178 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_179 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_89 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_104 (Dropout)	(None, 8, 8, 64)	0
conv2d_180 (Conv2D)	(None, 8, 8, 128)	73856
conv2d_181 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_90 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_105 (Dropout)	(None, 4, 4, 128)	0
conv2d_182 (Conv2D)	(None, 4, 4, 256)	295168
conv2d_183 (Conv2D)	(None, 4, 4, 256)	590080
max_pooling2d_91 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_106 (Dropout)	(None, 2, 2, 256)	0
flatten_22 (Flatten)	(None, 1024)	0
dense_66 (Dense)	(None, 512)	524800
batch_normalization_26 (BatchNormalization)	(None, 512)	2048
dense_67 (Dense)	(None, 512)	262656
dense_68 (Dense)	(None, 10)	5130

Total params: 1,967,018
Trainable params: 1,965,930
Non-trainable params: 1,088

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=10)
```

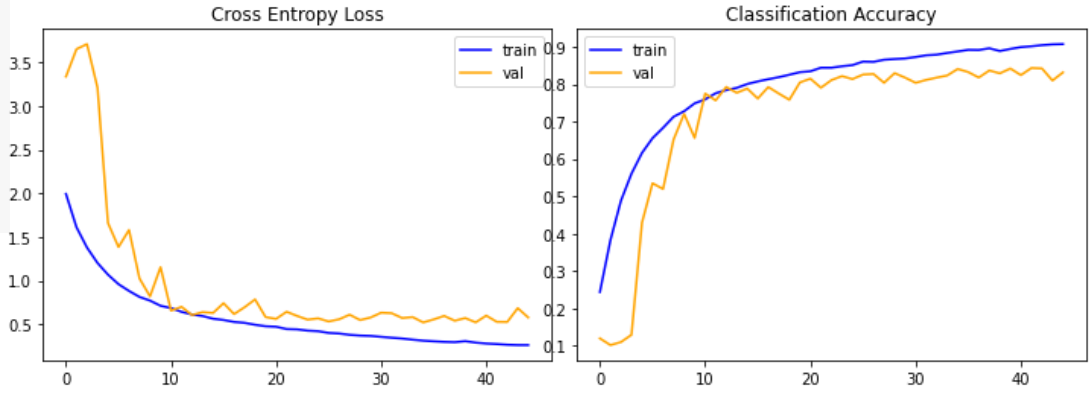
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

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

```
Epoch 45/200
79/79 [==============================] - 4s 52ms/step - loss: 0.2628 - accuracy: 0.9071 - val_loss: 0.5789 - val_accuracy: 0.8314
```

En set de test:

> 83.040



Se ha añadido capa de batch normalization entre las capas de clasificación. Al igual que antes, se han probado diferentes combinaciones y ésta era la que daba mejores resultados.

En las capas de aprendizaje de features no tenía sentido incluir más capa de estandarización ya que no mejoraba el modelo, pero se ha estandarizado la capa de clasificación para normalizar los valores y el resultado final es mejor.

Se aprecia que las curvas de validación son más irregulares, sobretodo al inicio, pero después se suavizan. Las irregularidades del inicio puede que sean debido a los pesos iniciales aleatorios asignados al modelo para empezar, que al estandarizarlos, tengan poco sentido y creen unos primeros modelos muy malos hasta que el aprendizaje no arranque y aprenda.

Proyecto 14

```
model = ks.Sequential()

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

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

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

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dense(512, activation='relu'))

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

Layer (type)	Output Shape	Param #
conv2d_200 (Conv2D)	(None, 32, 32, 32)	896
conv2d_201 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_31 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d_100 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_115 (Dropout)	(None, 16, 16, 32)	0
conv2d_202 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_203 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_101 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_116 (Dropout)	(None, 8, 8, 64)	0
conv2d_204 (Conv2D)	(None, 8, 8, 128)	73856
conv2d_205 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_102 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_117 (Dropout)	(None, 4, 4, 128)	0
conv2d_206 (Conv2D)	(None, 4, 4, 256)	295168
conv2d_207 (Conv2D)	(None, 4, 4, 256)	590080
max_pooling2d_103 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_118 (Dropout)	(None, 2, 2, 256)	0
flatten_25 (Flatten)	(None, 1024)	0
dense_75 (Dense)	(None, 512)	524800
batch_normalization_32 (BatchNormalization)	(None, 512)	2048
dense_76 (Dense)	(None, 512)	262656
dense_77 (Dense)	(None, 10)	5130

Total params: 1,967,018
Trainable params: 1,965,930
Non-trainable params: 1,088

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=10)
```

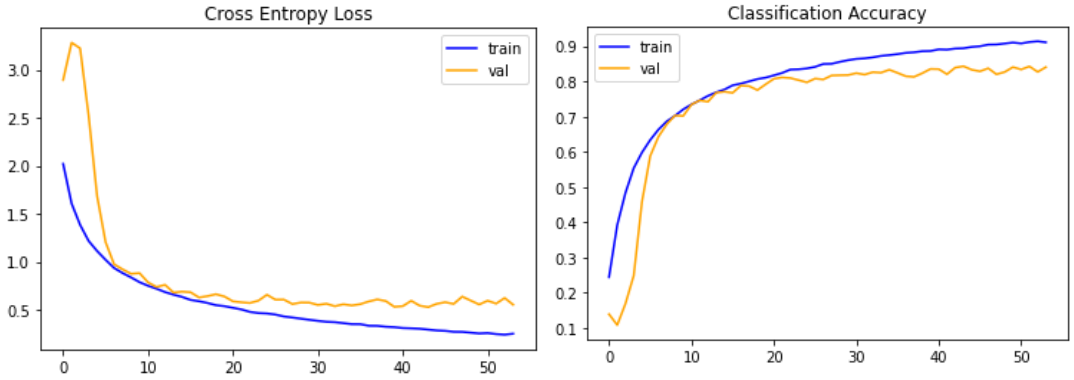
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

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

```
Epoch 54/200
79/79 [==============================] - 4s 52ms/step - loss: 0.2499 - accuracy: 0.9106 - val_loss: 0.5522 - val_accuracy: 0.8402
```

En set de test:

> 83.550



Añadido el inicializador de pesos “he-uniform” para todas las capas convolucionales para evitar los saltos e irregularidades iniciales, ya que inicializa los pesos siguiendo una distribución uniforme. El resultado es muy bueno tanto en la suavización de las curvas de validación como en el resultado final de accuracy.

Proyecto 15

```
model = ks.Sequential()

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

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

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

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dense(512, activation='relu'))

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

Layer (type)	Output Shape	Param #
conv2d_200 (Conv2D)	(None, 32, 32, 32)	896
conv2d_201 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_31 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d_100 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_115 (Dropout)	(None, 16, 16, 32)	0
conv2d_202 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_203 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_101 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_116 (Dropout)	(None, 8, 8, 64)	0
conv2d_204 (Conv2D)	(None, 8, 8, 128)	73856
conv2d_205 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_102 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_117 (Dropout)	(None, 4, 4, 128)	0
conv2d_206 (Conv2D)	(None, 4, 4, 256)	295168
conv2d_207 (Conv2D)	(None, 4, 4, 256)	590080
max_pooling2d_103 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_118 (Dropout)	(None, 2, 2, 256)	0
flatten_25 (Flatten)	(None, 1024)	0
dense_75 (Dense)	(None, 512)	524800
batch_normalization_32 (BatchNormalization)	(None, 512)	2048
dense_76 (Dense)	(None, 512)	262656
dense_77 (Dense)	(None, 10)	5130

Total params: 1,967,018
Trainable params: 1,965,930
Non-trainable params: 1,088

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=20)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=20)
```

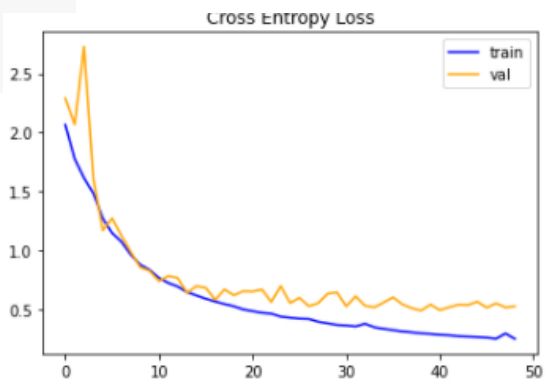
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 128,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

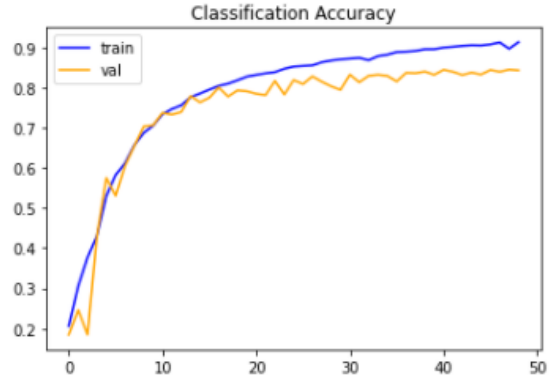
```
model.compile(optimizer='Adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
Epoch 54/200
313/313 [==============================] - 5s 17ms/step - loss: 0.2163 - accuracy: 0.9225 - val_loss: 0.5082 - val_accuracy: 0.8490
```

En set de test:

> 84.300





Batch size disminuido a 128. Con esto se quería conseguir que entraran en cada step menos imágenes (40.000 imágenes / 128 batch_size = 313 steps) y por tanto el entreno se hiciera en más pasos.

Con un batch size mayor el entreno es más rápido y puede converger antes, pero reduciéndolo se quiere un entreno más lento y controlado, además de obtener modelos más generalizados.

Proyecto 15b

```
model = ks.Sequential()

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

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

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

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dense(512, activation='relu'))

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

Layer (type)	Output Shape	Param #
conv2d_200 (Conv2D)	(None, 32, 32, 32)	896
conv2d_201 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_31 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d_100 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_115 (Dropout)	(None, 16, 16, 32)	0
conv2d_202 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_203 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_101 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_116 (Dropout)	(None, 8, 8, 64)	0
conv2d_204 (Conv2D)	(None, 8, 8, 128)	73856
conv2d_205 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_102 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_117 (Dropout)	(None, 4, 4, 128)	0
conv2d_206 (Conv2D)	(None, 4, 4, 256)	295168
conv2d_207 (Conv2D)	(None, 4, 4, 256)	590080
max_pooling2d_103 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_118 (Dropout)	(None, 2, 2, 256)	0
flatten_25 (Flatten)	(None, 1024)	0
dense_75 (Dense)	(None, 512)	524800
batch_normalization_32 (BatchNormalization)	(None, 512)	2048
dense_76 (Dense)	(None, 512)	262656
dense_77 (Dense)	(None, 10)	5130

Total params: 1,967,018
Trainable params: 1,965,930
Non-trainable params: 1,088

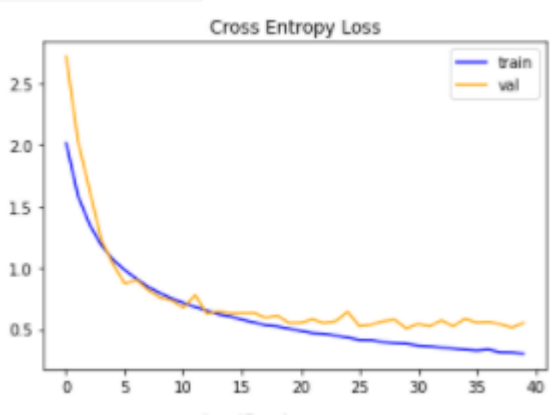
```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=20)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=20)
```

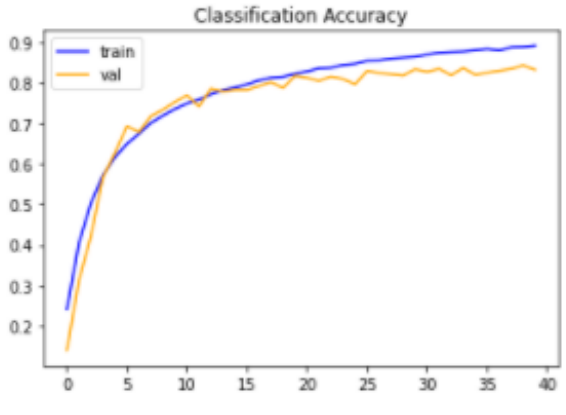
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 128,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

```
new_adam = Adam(learning_rate=0.0005)

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

En set de test:

> 83.170





Se ha probado el learning rate del optimizador Adam a 0.0005 (default era 0.001) pero el modelo no mejora. Observamos que las curvas son más suaves ya que el ratio de aprendizaje al ser más bajo, va a un ritmo más pausado al hacer el descenso de gradiente para encontrar el mínimo.

Proyecto 16 (data augm)

```
model = ks.Sequential()

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

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

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

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dense(512, activation='relu'))

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

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_200 (Conv2D)          (None, 32, 32, 32)        896

conv2d_201 (Conv2D)          (None, 32, 32, 32)        9248

batch_normalization_31 (Bat  (None, 32, 32, 32)        128
chNormalization)

max_pooling2d_100 (MaxPooli  (None, 16, 16, 32)        0
ng2D)

dropout_115 (Dropout)        (None, 16, 16, 32)        0

conv2d_202 (Conv2D)          (None, 16, 16, 64)        18496

conv2d_203 (Conv2D)          (None, 16, 16, 64)        36928

max_pooling2d_101 (MaxPooli  (None, 8, 8, 64)          0
ng2D)

dropout_116 (Dropout)        (None, 8, 8, 64)          0

conv2d_204 (Conv2D)          (None, 8, 8, 128)         73856

conv2d_205 (Conv2D)          (None, 8, 8, 128)         147584

max_pooling2d_102 (MaxPooli  (None, 4, 4, 128)         0
ng2D)

dropout_117 (Dropout)        (None, 4, 4, 128)         0

conv2d_206 (Conv2D)          (None, 4, 4, 256)         295168

conv2d_207 (Conv2D)          (None, 4, 4, 256)         590080

max_pooling2d_103 (MaxPooli  (None, 2, 2, 256)         0
ng2D)

dropout_118 (Dropout)        (None, 2, 2, 256)         0

flatten_25 (Flatten)         (None, 1024)              0

dense_75 (Dense)             (None, 512)               524800

batch_normalization_32 (Bat  (None, 512)               2048
chNormalization)

dense_76 (Dense)             (None, 512)               262656

dense_77 (Dense)             (None, 10)                5130

=================================================================
Total params: 1,967,018
Trainable params: 1,965,930
Non-trainable params: 1,088
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=15,
    zoom_range=0.1,
    shear_range=0.2,
    horizontal_flip=True,
    width_shift_range=0.2,
    height_shift_range=0.2,
    )

train_generator = train_datagen.flow(
    x_train,
    y_train_encoded,
    batch_size=128
    )
```

```
validation_datagen = ImageDataGenerator(
    rescale=1./255
    )
validation_generator = validation_datagen.flow(
    x_val,
    y_val_encoded,
    batch_size=100
)


test_datagen = ImageDataGenerator(
    rescale=1./255
    )
test_generator = test_datagen.flow(
    x_test,
    y_test_encoded,
    batch_size=100
)
```

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=30)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=30)
```

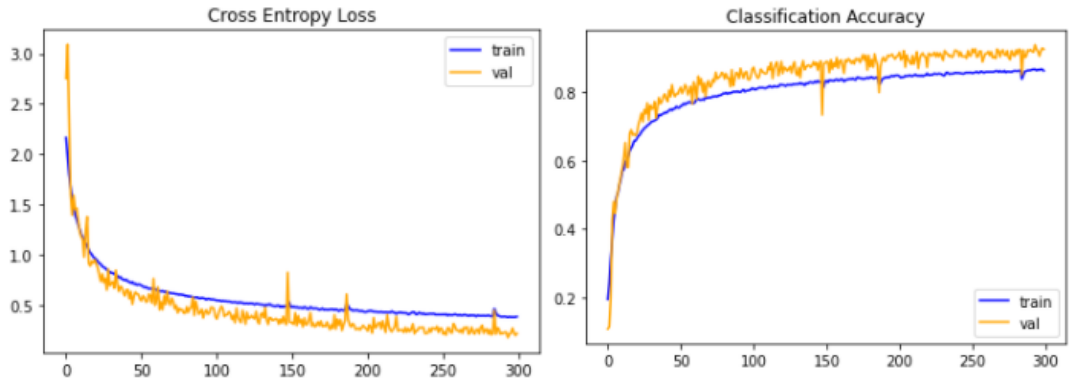
```
model.compile(optimizer='Adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(train_generator,
                    epochs=300,
                    validation_data=validation_generator,
                    use_multiprocessing=False,
                    steps_per_epoch=100,
                    validation_steps=100,
                    # callbacks=[callback_val_loss, callback_val_accuracy]
                    )
```

```
Epoch 300/300
100/100 [==============================] - 16s 158ms/step - loss: 0.3881 - accuracy: 0.8626 - val_loss: 0.2204 - val_accuracy: 0.9254
```

En set de test:

> 87.000



Se ha aplicado data augmentation a las imágenes que entran al modelo aplicando flip horizontal, pequeñas rotaciones y zooms. Con esto conseguimos que el modelo vea imágenes modificadas de las existentes, para poder generalizar mejor los features extraídos ya que son como imágenes nuevas.

No se ha aplicado ningún callback ya que se veía inicialmente que los entrenamientos se detenían demasiado rápido. Se ha optado por entrenar durante 300 epochs para asegurar que no se detenía demasiado temprano.

Aparecen irregularidades en las métricas de validación y vemos que, extrañamente, el set de validación obtiene mejores métricas que el set de training.

Proyecto 17

```
model = ks.Sequential()


model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu', kernel_regularizer=l2(0.001), padding='same', input_shape=(32,32,3), kernel_initializer='he_uniform'))
model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu', kernel_regularizer=l2(0.001), padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.25))

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

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu', kernel_regularizer=l2(0.001), padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu', kernel_regularizer=l2(0.001), padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', kernel_regularizer=l2(0.001), padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', kernel_regularizer=l2(0.001), padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu', kernel_regularizer=l2(0.001)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dense(512, activation='relu', kernel_regularizer=l2(0.001)))

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

Layer (type)	Output Shape	Param #
conv2d_200 (Conv2D)	(None, 32, 32, 32)	896
conv2d_201 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_31 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d_100 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_115 (Dropout)	(None, 16, 16, 32)	0
conv2d_202 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_203 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_101 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_116 (Dropout)	(None, 8, 8, 64)	0
conv2d_204 (Conv2D)	(None, 8, 8, 128)	73856
conv2d_205 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_102 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_117 (Dropout)	(None, 4, 4, 128)	0
conv2d_206 (Conv2D)	(None, 4, 4, 256)	295168
conv2d_207 (Conv2D)	(None, 4, 4, 256)	590080
max_pooling2d_103 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_118 (Dropout)	(None, 2, 2, 256)	0
flatten_25 (Flatten)	(None, 1024)	0
dense_75 (Dense)	(None, 512)	524800
batch_normalization_32 (BatchNormalization)	(None, 512)	2048
dense_76 (Dense)	(None, 512)	262656
dense_77 (Dense)	(None, 10)	5130

Total params: 1,967,018
Trainable params: 1,965,930
Non-trainable params: 1,088

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=30)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=30)
```

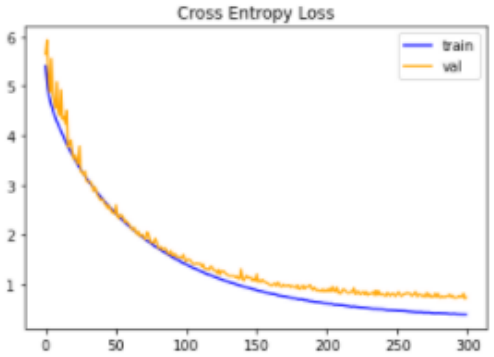
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    use_multiprocessing=False, batch_size= 128,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

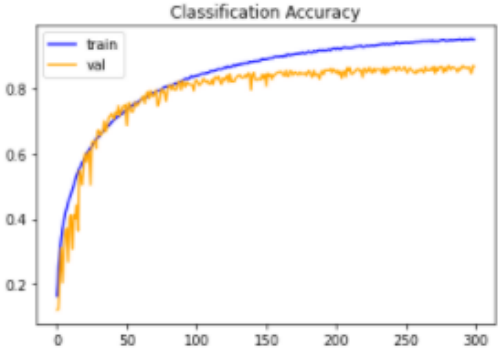
```
my_sgd = SGD(learning_rate=0.001, momentum=0.9)

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

```
Epoch 300/300
313/313 [==============================] - 5s 17ms/step - loss: 0.3886 - accuracy: 0.9505 - val_loss: 0.7315 - val_accuracy: 0.8676
```

En set de test:

> 86.000





En entrenos largos (sin callbacks de early stopping) se observa que el modelo aún presentaba overfitting, así que se ha optado por añadir regularizadores L2 a todas las capas convolucionales y densas de la arquitectura. Al añadir el regularizador también se ha cambiado el optimizador "Adam" por el SGD ya que funciona mejor con L2.

Con esto conseguimos una mejora generalización ya que se eliminan los features más dominantes y se tiende a menor overfitting. Con SGD y momentum conseguimos que durante el entreno el modelo pueda superar los mínimos locales encontrados mediante la inercia aplicada.

Proyecto 18 (data augm)

```
model = ks.Sequential()


model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu', kernel_regularizer=l2(0.001),  padding='same', input_shape=(32,32,3), kernel_initializer='he_uniform'))
model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu', kernel_regularizer=l2(0.001),  padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.25))

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

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu', kernel_regularizer=l2(0.001), padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu', kernel_regularizer=l2(0.001), padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', kernel_regularizer=l2(0.001), padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu', kernel_regularizer=l2(0.001), padding='same', kernel_initializer='he_uniform'))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='relu', kernel_regularizer=l2(0.001)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dense(512, activation='relu', kernel_regularizer=l2(0.001)))

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

```
my_sgd = SGD(learning_rate=0.001, momentum=0.9)

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

```
acc_max = 0
acc = 0
while acc < 0.90:
  # history =
  model.fit(train_generator, steps_per_epoch = 100,
                      validation_steps=100, use_multiprocessing=False,
                      epochs=1, validation_data=validation_generator,
                      callbacks=[callback_val_loss])

  _, acc = model.evaluate(test_generator, verbose=0)

  if acc > acc_max:
    acc_max = acc
    model.save('model_DA_2.h5')
    print('> %.3f' % (acc * 100.0))
```

Layer (type)	Output Shape	Param #
conv2d_200 (Conv2D)	(None, 32, 32, 32)	896
conv2d_201 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_31 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d_100 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_115 (Dropout)	(None, 16, 16, 32)	0
conv2d_202 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_203 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_101 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_116 (Dropout)	(None, 8, 8, 64)	0
conv2d_204 (Conv2D)	(None, 8, 8, 128)	73856
conv2d_205 (Conv2D)	(None, 8, 8, 128)	147584
max_pooling2d_102 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_117 (Dropout)	(None, 4, 4, 128)	0
conv2d_206 (Conv2D)	(None, 4, 4, 256)	295168
conv2d_207 (Conv2D)	(None, 4, 4, 256)	590080
max_pooling2d_103 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_118 (Dropout)	(None, 2, 2, 256)	0
flatten_25 (Flatten)	(None, 1024)	0
dense_75 (Dense)	(None, 512)	524800
batch_normalization_32 (BatchNormalization)	(None, 512)	2048
dense_76 (Dense)	(None, 512)	262656
dense_77 (Dense)	(None, 10)	5130

```
Total params: 1,967,018
Trainable params: 1,965,930
Non-trainable params: 1,088
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=25,
    zoom_range=0.1,
    shear_range=0.2,
    horizontal_flip=True,
    width_shift_range=0.15,
    height_shift_range=0.15,
    )

train_generator = train_datagen.flow(
    x_train,
    y_train_encoded,
    batch_size=400
)
```

```
validation_datagen = ImageDataGenerator(
    rescale=1./255
    )
validation_generator = validation_datagen.flow(
    x_val,
    y_val_encoded,
    batch_size=100
)


test_datagen = ImageDataGenerator(
    rescale=1./255
    )
test_generator = test_datagen.flow(
    x_test,
    y_test_encoded,
    batch_size=100
)
```

```
from tensorflow.keras.callbacks import EarlyStopping

callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=30)
callback_val_loss = EarlyStopping(monitor="val_loss", patience=30)
```

En set de test:

> 90.060

Igual que en el proyecto 16 se ha aplicado data augmentation pero esta vez usando el optimizador SGD y regulaizadores L2 ya que daban mejores resultados, se han variado un poco los parámetros del aumento de datos (derivado de prueba y error) y también se ha aplicado la función while adjunta arriba para registrar las mejoras del modelo y guardar el modelo mejorado cada vez (encontrado en internet). Con esta función el entreno no depende de los epochs estipulados ya que va entrenando hasta mejorar la accuracy y llegar, en nuestro caso, a <0.90, por tanto intentar superar una accuracy del 90%.

El entreno ha sido larguísimo, se ha ejecutado durante 19336.9s == 5,37h durante la noche.

El punto en contra de este entreno es que no se obtienen las curvas de train/validation para graficar.

```
100/100 [==============================] - 29s 196ms/step - loss: 5.5507 - accuracy: 0.1385 - val_loss: 5.4855 - val_accuracy: 0.1009
> 10.040
100/100 [==============================] - 20s 197ms/step - loss: 5.3181 - accuracy: 0.1791 - val_loss: 5.6829 - val_accuracy: 0.1071
> 10.670
100/100 [==============================] - 19s 187ms/step - loss: 5.2004 - accuracy: 0.2072 - val_loss: 5.5287 - val_accuracy: 0.1228
> 12.250
100/100 [==============================] - 19s 188ms/step - loss: 5.1253 - accuracy: 0.2260 - val_loss: 5.3450 - val_accuracy: 0.1619
> 16.250
100/100 [==============================] - 20s 199ms/step - loss: 5.0699 - accuracy: 0.2411 - val_loss: 5.2404 - val_accuracy: 0.1844
> 18.060
100/100 [==============================] - 19s 188ms/step - loss: 5.0209 - accuracy: 0.2535 - val_loss: 5.2142 - val_accuracy: 0.2005
> 19.590
100/100 [==============================] - 19s 188ms/step - loss: 4.9758 - accuracy: 0.2636 - val_loss: 5.1823 - val_accuracy: 0.2152
> 21.050
100/100 [==============================] - 19s 189ms/step - loss: 4.9374 - accuracy: 0.2717 - val_loss: 5.2773 - val_accuracy: 0.1993
100/100 [==============================] - 19s 188ms/step - loss: 4.8955 - accuracy: 0.2815 - val_loss: 4.9766 - val_accuracy: 0.2685
> 27.290
100/100 [==============================] - 19s 188ms/step - loss: 4.8518 - accuracy: 0.2968 - val_loss: 4.8961 - val_accuracy: 0.2782
> 28.170
100/100 [==============================] - 19s 190ms/step - loss: 4.8175 - accuracy: 0.3073 - val_loss: 4.8699 - val_accuracy: 0.2897
> 29.190
100/100 [==============================] - 19s 187ms/step - loss: 4.7792 - accuracy: 0.3154 - val_loss: 4.9928 - val_accuracy: 0.2556
100/100 [==============================] - 19s 188ms/step - loss: 4.7439 - accuracy: 0.3253 - val_loss: 5.0234 - val_accuracy: 0.2721
100/100 [==============================] - 19s 188ms/step - loss: 4.7035 - accuracy: 0.3353 - val_loss: 5.2575 - val_accuracy: 0.2207
100/100 [==============================] - 19s 190ms/step - loss: 4.6780 - accuracy: 0.3442 - val_loss: 4.6840 - val_accuracy: 0.3430
> 34.650
100/100 [==============================] - 19s 189ms/step - loss: 4.6519 - accuracy: 0.3479 - val_loss: 4.8978 - val_accuracy: 0.2880
100/100 [==============================] - 19s 189ms/step - loss: 4.6228 - accuracy: 0.3550 - val_loss: 4.7774 - val_accuracy: 0.3141
```

```
100/100 [==============================] - 20s 204ms/step - loss: 0.4246 - accuracy: 0.9012 - val_loss: 0.2958 - val_accuracy: 0.9461
100/100 [==============================] - 20s 203ms/step - loss: 0.4186 - accuracy: 0.9028 - val_loss: 0.2715 - val_accuracy: 0.9559
100/100 [==============================] - 20s 203ms/step - loss: 0.4251 - accuracy: 0.9009 - val_loss: 0.2580 - val_accuracy: 0.9626
100/100 [==============================] - 20s 202ms/step - loss: 0.4214 - accuracy: 0.9030 - val_loss: 0.3043 - val_accuracy: 0.9434
100/100 [==============================] - 20s 205ms/step - loss: 0.4162 - accuracy: 0.9051 - val_loss: 0.2502 - val_accuracy: 0.9659
100/100 [==============================] - 20s 204ms/step - loss: 0.4210 - accuracy: 0.9006 - val_loss: 0.3576 - val_accuracy: 0.9253
100/100 [==============================] - 20s 205ms/step - loss: 0.4183 - accuracy: 0.9026 - val_loss: 0.2476 - val_accuracy: 0.9658
100/100 [==============================] - 20s 205ms/step - loss: 0.4189 - accuracy: 0.9039 - val_loss: 0.3244 - val_accuracy: 0.9346
100/100 [==============================] - 20s 203ms/step - loss: 0.4216 - accuracy: 0.9001 - val_loss: 0.3444 - val_accuracy: 0.9301
100/100 [==============================] - 20s 205ms/step - loss: 0.4188 - accuracy: 0.9029 - val_loss: 0.3215 - val_accuracy: 0.9383
100/100 [==============================] - 20s 204ms/step - loss: 0.4214 - accuracy: 0.9009 - val_loss: 0.2786 - val_accuracy: 0.9553
100/100 [==============================] - 20s 204ms/step - loss: 0.4134 - accuracy: 0.9058 - val_loss: 0.3126 - val_accuracy: 0.9390
100/100 [==============================] - 20s 204ms/step - loss: 0.4153 - accuracy: 0.9050 - val_loss: 0.3117 - val_accuracy: 0.9422
100/100 [==============================] - 20s 205ms/step - loss: 0.4178 - accuracy: 0.9054 - val_loss: 0.3672 - val_accuracy: 0.9211
100/100 [==============================] - 21s 205ms/step - loss: 0.4176 - accuracy: 0.9043 - val_loss: 0.2920 - val_accuracy: 0.9482
100/100 [==============================] - 20s 205ms/step - loss: 0.4164 - accuracy: 0.9038 - val_loss: 0.2271 - val_accuracy: 0.9734
> 90.060
```

Conclusiones

Se ha conseguido superar el 90% de accuracy en el set de test. Todos los sets están perfectamente balanceados por lo que la accuracy es una buena métrica de medición de la performance del modelo.

Considero que la accuracy conseguida es muy buena ya que el modelo desarrollado es bastante básico, con algo menos de 2M de parámetros, con 22 capas de las cuales 8 son de convolución, 4 de maxpooling, 4 de dropout, 2 de batch normalization, 3 dense y 1 flatten.

Viendo en perspectiva los proyectos, en el proyecto 3 cuando se añadía otro bloque de convolución+MaxPooling, los resultados no eran del todo malos y se podrían haber pulido más adelante con dropout pero teniendo más capacidad de aprendizaje.

Como punto final, ha faltado por implementar un modelo con Transfer Learning pero por falta de tiempo y errores en las primeras pruebas realizadas, no se ha podido probar satisfactoriamente.