PROYECTO IV – VISIÓN POR COMPUTADOR

SANTIAGO EZEQUIEL VELASCO

1° Modelo

```
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.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, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 64)	0
conv2d_2 (Conv2D)	(None, 8, 8, 128)	73856
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 32)	65568
dense_1 (Dense)	(None, 10)	330

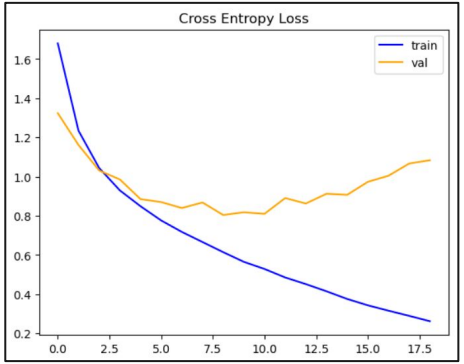
Total params: 159,146
Trainable params: 159,146
Non-trainable params: 0

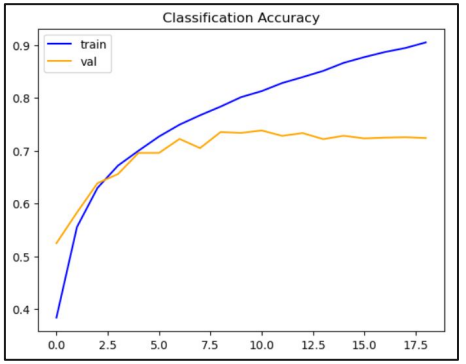
```
callback_val_loss = EarlyStopping(monitor="val_loss", patience=10)
callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=10)
```

```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    batch_size= 64, validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

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

```
> 72.320
```





En este primer modelo se realizaron las siguientes modificaciones con respecto al notebook original:

- Se agregaron dos capas convolucionales con 64 y 128 filtros, lo que permite extraer características más complejas y profundas.
- Se añadieron dos callbacks de EarlyStopping, lo que ayuda a reducir el overfitting y permite incrementar la cantidad de épocas de entrenamiento, ya que el modelo se detendrá automáticamente en el momento óptimo.
- Se aumentó la cantidad de épocas a 200.

Como resultado, podemos observar una mejora en el modelo, alcanzando un AUC de 73.3%. Sin embargo, también se evidencia un alto grado de overfitting.

2° Modelo

```
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(0.4))

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(0.4))


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

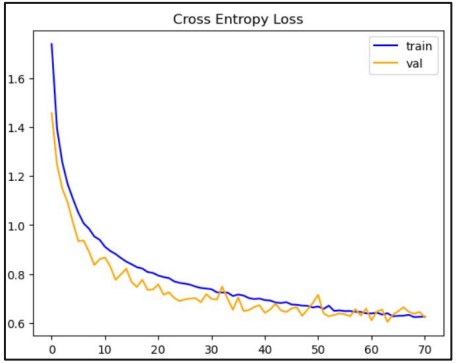
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 32)                65568
_________________________________________________________________
dense_1 (Dense)              (None, 10)                330
=================================================================
Total params: 159,146
Trainable params: 159,146
Non-trainable params: 0
_________________________________________________________________
```

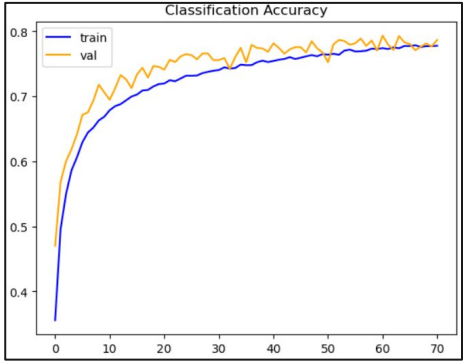
```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    batch_size= 64, validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

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

✓ 2.1s

```
> 77.430
```





En este seguro modelo se realizaron las siguientes modificaciones con respecto al modelo anterior:

- Se agregaron tres capas de dropout entre cada conjunto de capa convolucional y pooling

Como resultado, podemos observar una mejora en el modelo, alcanzando un AUC de 77.43% reduciendo en gran medida el overfitting respecto al modelo anterior

3° Modelo

```
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.BatchNormalization())
model.add(ks.layers.Conv2D(32, (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(0.4))

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.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(0.4))

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(0.4))

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


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

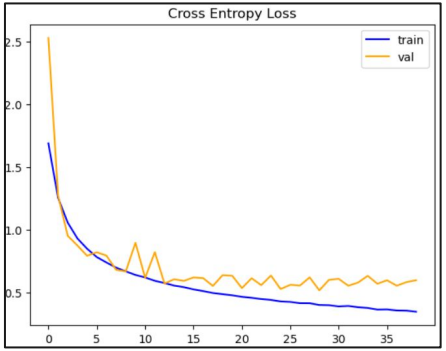
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNo	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_1 (Batch	(None, 32, 32, 32)	128
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18496
batch_normalization_2 (Batch	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 64)	0
conv2d_3 (Conv2D)	(None, 8, 8, 64)	36928
batch_normalization_3 (Batch	(None, 8, 8, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 64)	0
dropout_1 (Dropout)	(None, 4, 4, 64)	0
conv2d_4 (Conv2D)	(None, 4, 4, 128)	73856
batch_normalization_4 (Batch	(None, 4, 4, 128)	512
conv2d_5 (Conv2D)	(None, 4, 4, 128)	147584
batch_normalization_5 (Batch	(None, 4, 4, 128)	512
max_pooling2d_3 (MaxPooling2	(None, 2, 2, 128)	0
dropout_2 (Dropout)	(None, 2, 2, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 32)	16416
dense_1 (Dense)	(None, 10)	330

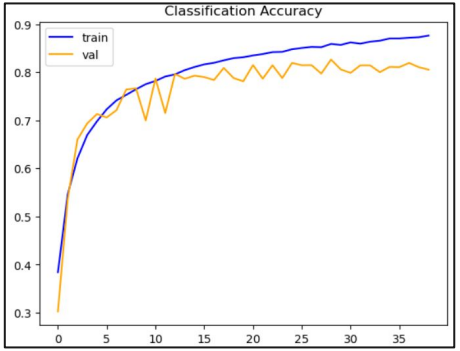
Total params: 305,546
Trainable params: 304,650
Non-trainable params: 896

```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    batch_size= 64, validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

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

```
> 80.360
```





En este tercer modelo se realizaron las siguientes modificaciones con respecto al modelo anterior:

- Se duplicaron las capas convolucionales en cada bloque, lo que permite extraer características más complejas antes de aplicar la capa de pooling.
- Se añadieron capas de BatchNormalization, que estabilizan el modelo y aceleran el entrenamiento.

Como resultado, podemos observar una mejora en el modelo, alcanzando un AUC de 80.36% pero nuevamente nos volvemos a encontrar con un elevado overfitting en el modelo, con una gran brecha entre el AUC de train del 87% contra los 80% del de test

4° Modelo

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(64, (3, 3), activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), activation='relu',
                           padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))


model.add(ks.layers.Conv2D(128, (3, 3), activation='relu',
                           padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), activation='relu',
                           padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))


model.add(ks.layers.Conv2D(256, (3, 3), activation='relu',
                           padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(256, (3, 3), activation='relu',
                           padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.6))


model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.65))
model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.Dropout(0.7))


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

Model: "sequential"

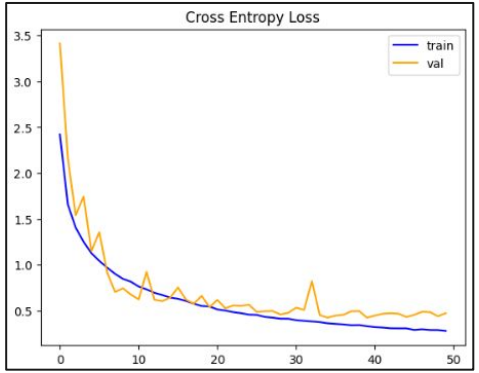
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_3 (BatchNormalization)	(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)	295,168
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 256)	1,024
conv2d_5 (Conv2D)	(None, 8, 8, 256)	590,080
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 256)	1,024
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, 128)	524,416
batch_normalization_6 (BatchNormalization)	(None, 128)	512
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 256)	33,024
dropout_4 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 10)	2,570

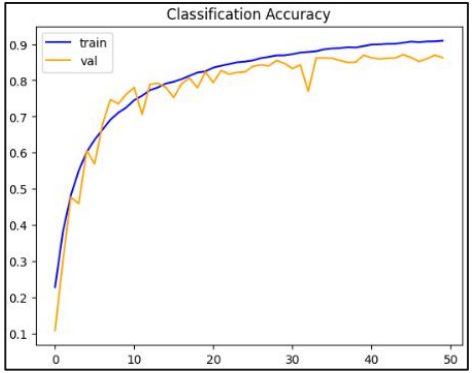
Total params: 1,709,514 (6.52 MB)
Trainable params: 1,707,466 (6.51 MB)
Non-trainable params: 2,048 (8.00 KB)

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

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

```
> 85.930
```





En este cuarto modelo se realizaron las siguientes modificaciones con respecto al modelo anterior:

- Se incrementó el batch_size, pasando de 64 a 128, aprendiendo de más datos a la vez, por lo que los ajustes en los pesos son más estables y menos aleatorios.
- Se le dio mayor profundidad a las capas convolucionales, pasando de 32→64→128 filtros a 64→128→256.
- Se le dieron mayor profundidad a las capas de Dropout, teniendo ahora capas desde 0.5 a 0.7, para poder evitar el overfitting lo máximo posible.
- Además se modificaron las capas densas del final, ahora contamos con dos capas de 128 y 256.

Todos estos cambios, generan una red neuronal más compleja (pasando de 305.546 a 1.709.519 parámetros) que le permite al modelo tener mejores resultados como se puede apreciar.

5° Modelo

```
model = ks.Sequential()

l2_reg = ks.regularizers.l2(0.0005)
model.add(ks.layers.Conv2D(64, (3, 3), activation='relu',
                           padding='same', input_shape=(32, 32, 3),
                           kernel_regularizer=l2_reg,
                           kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), activation='relu',
                           padding='same', kernel_regularizer=l2_reg,
                           kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))


model.add(ks.layers.Conv2D(128, (3, 3), activation='relu',
                           padding='same', kernel_regularizer=l2_reg,
                           kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), activation='relu',
                           padding='same', kernel_regularizer=l2_reg,
                           kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))


model.add(ks.layers.Conv2D(256, (3, 3), activation='relu',
                           padding='same', kernel_regularizer=l2_reg,
                           kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(256, (3, 3), activation='relu',
                           padding='same', kernel_regularizer=l2_reg,
                           kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.6))


model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(128, activation='relu', kernel_regularizer=l2_reg,
                          kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.65))
model.add(ks.layers.Dense(256, activation='relu', kernel_regularizer=l2_reg,
                          kernel_initializer='he_uniform'))
model.add(ks.layers.Dropout(0.7))



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

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_1 (BatchNormalization)	(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)	73,856
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_3 (BatchNormalization)	(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)	295,168
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 256)	1,024
conv2d_5 (Conv2D)	(None, 8, 8, 256)	590,080
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 256)	1,024
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, 128)	524,416
batch_normalization_6 (BatchNormalization)	(None, 128)	512
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 256)	33,024
dropout_4 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 10)	2,570

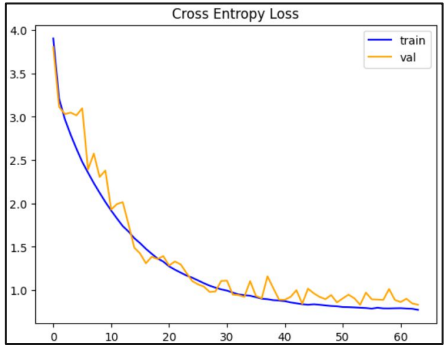
Total params: 1,709,514 (6.52 MB)
Trainable params: 1,707,466 (6.51 MB)
Non-trainable params: 2,048 (8.00 KB)

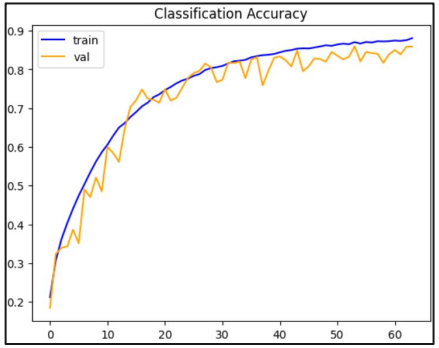
```
new_sgd = SGD(learning_rate = 0.01, momentum=0.9)

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

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

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

```
> 85.770
```





Como vimos en el modelo anterior aún se seguía manteniendo un cierto overfitting, para evitar seguirles dando peso a las capas de dropout, se decidió incorporar los parametros de **kernel_regularizer** y **kernel_initializer.**

Estos parámetros evitan que los pesos crezcan demasiado rápido y los distribuye de una mejor manera, reduciendo la complejidad del modelo y evitando el overfitting.

Por otro lado, como sabemos estos parámetros no se llevan muy bien con el optimizador Adam, por lo tanto, se utilizó el optimizador SGD fijando parámetros de learning_rate y momentum, tal como se puede observar.

Este modelo, aunque puede generar un AUC ligeramente menor, reduce significativamente el overfitting, permitiendo que las curvas de entrenamiento y validación converjan de manera más estable y consistente.

6° Modelo

```
model = ks.Sequential()

l2_reg = ks.regularizers.l2(0.0001)
model.add(ks.layers.Conv2D(64, (3, 3), activation='relu',
                           padding='same', input_shape=(32, 32, 3),
                           kernel_regularizer=l2_reg,
                           kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), activation='relu',
                           padding='same', kernel_regularizer=l2_reg,
                           kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))


model.add(ks.layers.Conv2D(128, (3, 3), activation='relu',
                           padding='same', kernel_regularizer=l2_reg,
                           kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), activation='relu',
                           padding='same', kernel_regularizer=l2_reg,
                           kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))


model.add(ks.layers.Conv2D(256, (3, 3), activation='relu',
                           padding='same', kernel_regularizer=l2_reg,
                           kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(256, (3, 3), activation='relu',
                           padding='same', kernel_regularizer=l2_reg,
                           kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.6))


model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(256, activation='relu', kernel_regularizer=l2_reg,
                          kernel_initializer='he_uniform'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.7))
model.add(ks.layers.Dense(256, activation='relu', kernel_regularizer=l2_reg,
                          kernel_initializer='he_uniform'))
model.add(ks.layers.Dropout(0.7))

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

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_1 (BatchNormalization)	(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)	73,856
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_3 (BatchNormalization)	(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)	295,168
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 256)	1,024
conv2d_5 (Conv2D)	(None, 8, 8, 256)	590,080
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 256)	1,024
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)	1,048,832
batch_normalization_6 (BatchNormalization)	(None, 256)	1,024
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 256)	65,792
dropout_4 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 10)	2,570

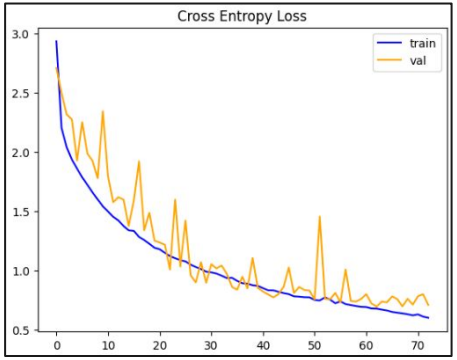
Total params: 2,267,210 (8.65 MB)
Trainable params: 2,264,906 (8.64 MB)
Non-trainable params: 2,304 (9.00 KB)

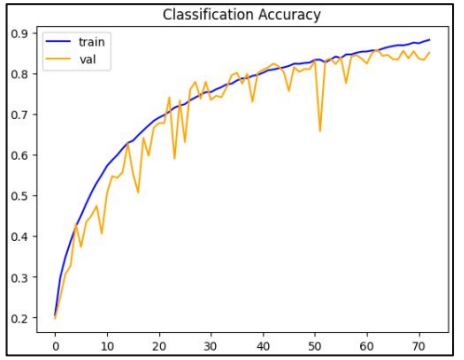
```
new_sgd = SGD(learning_rate = 0.01, momentum=0.9)

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

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

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

```
> 84.650
```





Respecto al modelo anterior, podemos observar los siguientes cambios:

- Reducción del parámetro **kernel_regularizer**, por lo que este modelo aplica una regularización más débil que la anterior
- Incremento en la cantidad de parámetros, debido al incremento de las capas densas finales.
- Pequeño incremento en la penúltima capa de dropout.

Como resultado de este modelo, podemos observar la sensibilidad del parámetro **kernel_regularizer**,

Aunque ambos modelos tiene valores de accuracy similares, podemos concluir que el modelo anterior es mejor ya que su convergencia más suave sugiere mejor generalización y menor sensibilidad a los datos.

7° Modelo

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(64, (3, 3), activation='swish',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.4))
model.add(ks.layers.Conv2D(64, (3, 3), activation='swish',
                           padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Conv2D(128, (3, 3), activation='relu',
                           padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Conv2D(128, (3, 3), activation='relu',
                           padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Conv2D(256, (3, 3), activation='swish',
                           padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.6))
model.add(ks.layers.Conv2D(256, (3, 3), activation='swish',
                           padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(ks.layers.Dropout(0.6))

model.add(ks.layers.Conv2D(512, (3, 3), activation='relu',
                           padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.6))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='swish'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.7))
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.7))

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

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization_9 (BatchNormalization)	(None, 32, 32, 64)	256
dropout_9 (Dropout)	(None, 32, 32, 64)	0
conv2d_8 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_10 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_10 (Dropout)	(None, 16, 16, 64)	0
conv2d_9 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_11 (BatchNormalization)	(None, 16, 16, 128)	512
dropout_11 (Dropout)	(None, 16, 16, 128)	0
conv2d_10 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_12 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_12 (Dropout)	(None, 8, 8, 128)	0
conv2d_11 (Conv2D)	(None, 8, 8, 256)	295,168
batch_normalization_13 (BatchNormalization)	(None, 8, 8, 256)	1,024
dropout_13 (Dropout)	(None, 8, 8, 256)	0
conv2d_12 (Conv2D)	(None, 8, 8, 256)	590,080
batch_normalization_14 (BatchNormalization)	(None, 8, 8, 256)	1,024
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_14 (Dropout)	(None, 4, 4, 256)	0
conv2d_13 (Conv2D)	(None, 4, 4, 512)	1,180,160
batch_normalization_15 (BatchNormalization)	(None, 4, 4, 512)	2,048
dropout_15 (Dropout)	(None, 4, 4, 512)	0
flatten_1 (Flatten)	(None, 8192)	0
dense_3 (Dense)	(None, 512)	4,194,816
batch_normalization_16 (BatchNormalization)	(None, 512)	2,048
dropout_16 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 512)	262,656
batch_normalization_17 (BatchNormalization)	(None, 512)	2,048
dropout_17 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 10)	5,130

Total params: 6,797,898 (25.93 MB)
Trainable params: 6,793,034 (25.91 MB)
Non-trainable params: 4,864 (19.00 KB)

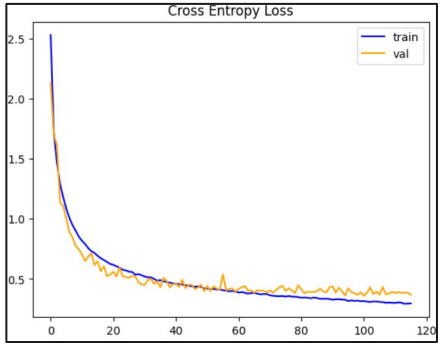
```
callback_val_loss = EarlyStopping(monitor="val_loss", patience=15)
callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=15)
```

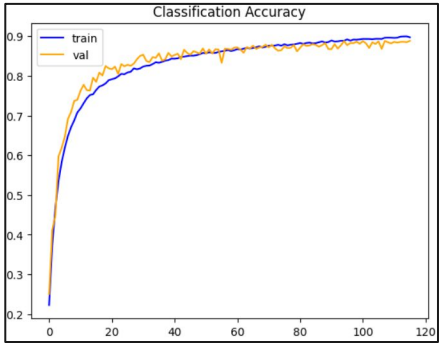
```
model.compile(optimizer='Adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(x_train_scaled, y_train, epochs=200,
                    batch_size= 64, validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

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

```
> 88.560
```





Acá nos encontramos con un modelo diferente a los anteriores:

- En este caso no hemos utilizado los parámetros de **kernel_regularizer** y **kernel_initializer.** Por lo tanto hemos vuelto a utilizar el optimizador Adam. El cual tiene ventajas aplicar momentum y aplica tasa de aprendizaje adaptativa.
- Se ha alternado el uso de funciones de activación, donde algunas tienen activación Relu y otras Swish.
- Se incrementó la patience de los callbacks, ya que se detectó que el modelo frenaba con anticipación, siendo que podia seguir mejorando.
- En este modelo contamos con una capa convolucional adicional de 512 filtros, aumentando su capacidad de extracción de características antes del aplanado.
- Este modelo aplica Dropout de forma más distribuida creando más capas de menos peso a lo largo de la red neuronal.
- Finalmente se incrementaron las últimas capas densas de 256 a 512 neuronas.

Como resultado, observamos que tenemos una red neuronal más compleja con casi 7 millones de parámetros. Llegando a un accuracy de 88.56% y con una correcta convergencia entre las curvas de train y test.

8° Modelo

```
model = ks.Sequential()

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


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


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


model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(128, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.Dropout(0.5))


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

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_1 (BatchNormalization)	(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)	73,856
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_3 (BatchNormalization)	(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)	295,168
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 256)	1,024
conv2d_5 (Conv2D)	(None, 8, 8, 256)	590,080
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 256)	1,024
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, 128)	524,416
batch_normalization_6 (BatchNormalization)	(None, 128)	512
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 256)	33,024
dropout_4 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 10)	2,570

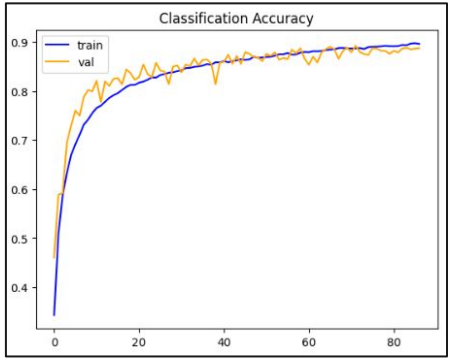
Total params: 1,709,514 (6.52 MB)
Trainable params: 1,707,466 (6.51 MB)
Non-trainable params: 2,048 (8.00 KB)

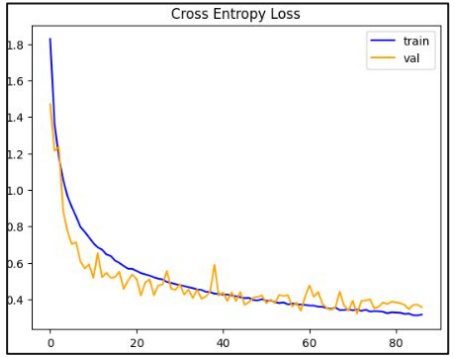
```
data_generator = ks.preprocessing.image.ImageDataGenerator(
          width_shift_range=0.1,
          height_shift_range=0.1,
          horizontal_flip=True,
          rotation_range=15,
          shear_range=0.1,
          fill_mode='nearest'
)
x_train_scaled_gen = data_generator.flow(x_train_scaled, y_train)
```

```
history = model.fit(x_train_scaled_gen, epochs=200,
                batch_size= 64, validation_data=(x_val_scaled, y_val),
                callbacks=[callback_val_loss, callback_val_accuracy])
```

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

```
> 88.430
```





En este modelo tenemos:

- Se introdujo data augmentation, generando modificaciones de desplazamiento (horizontal y verticalmente), imágenes inversas, rotación, deformación y relleno de áreas vacías
- Hemos incorporado una red neuronal más sencilla a las anteriores, con 1.709.514 parámetros, siendo una red similar a la utilizada en el modelo 4, pero con significativamente menos pesos en las capas de dropout.
- El optimizador utilizado fue adam.

Como resultado, vemos que con una red neuronal similar a la utilizada anteriormente y utilizando data augmentation, hemos logrado pasar de un 85% a un 88% de accuracy

9° Modelo

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(64, (3, 3), activation='swish',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), activation='swish',
                           padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(ks.layers.Dropout(0.3))

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

model.add(ks.layers.Conv2D(256, (3, 3), activation='swish',
                           padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(256, (3, 3), activation='swish',
                           padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(ks.layers.Dropout(0.5))

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

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(512, activation='swish'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.4))
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.4))

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

Model: "sequential"

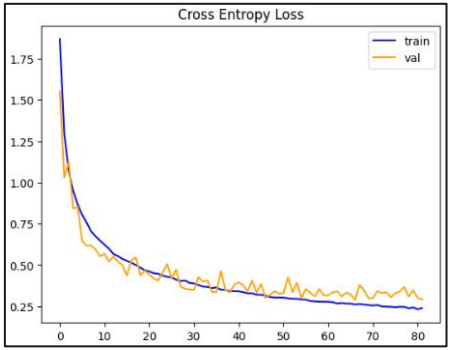
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_3 (BatchNormalization)	(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)	295,168
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 256)	1,024
conv2d_5 (Conv2D)	(None, 8, 8, 256)	590,080
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 256)	1,024
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
conv2d_6 (Conv2D)	(None, 4, 4, 512)	1,180,160
batch_normalization_6 (BatchNormalization)	(None, 4, 4, 512)	2,048
dropout_3 (Dropout)	(None, 4, 4, 512)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 512)	4,194,816
batch_normalization_7 (BatchNormalization)	(None, 512)	2,048
dropout_4 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 512)	262,656
batch_normalization_8 (BatchNormalization)	(None, 512)	2,048
dropout_5 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5,130

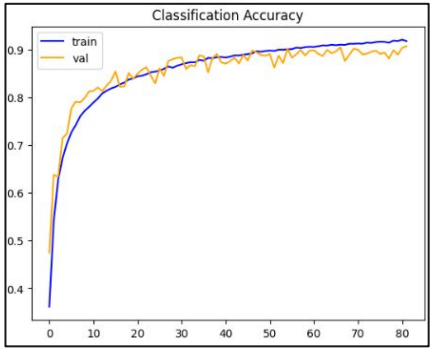
Total params: 6,797,898 (25.93 MB)
Trainable params: 6,793,034 (25.91 MB)
Non-trainable params: 4,864 (19.00 KB)

```
data_generator = ks.preprocessing.image.ImageDataGenerator(
          width_shift_range=0.1,
          height_shift_range=0.1,
          horizontal_flip=True,
          rotation_range=15,
          shear_range=0.1,
          fill_mode='nearest'
)
x_train_scaled_gen = data_generator.flow(x_train_scaled, y_train)
```

```
history = model.fit(x_train_scaled_gen, epochs=200,
                    batch_size= 64, validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

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

     > 90.450
```





Este modelo se basa en una mejora del modelo anterior donde se le agregó:

- En este hacemos uso alternado de las funciones de activación swish y relu, que como vimos en casos anteriores generaba buenos resultados.
- Además se agregó una capa de 512 filtros, generando una estructura similar a la del modelo 7. Cuenta con 6.797.898 parámetros, siendo una red con mayo complejidad.
- Se aplicó data augmentation con las mismas transformaciones del modelo anterior. Esto nota significativamente la disminución de los pesos en la red neuronal de las capas de dropout, mejorando significativamente el modelo.

Como resultado, con esta compleja red neuronal junto con el data augmentation hemos logrado arribar al 90.45% de accuracy.

10° Modelo

```
import tensorflow.keras as ks

model = ks.Sequential()

model.add(ks.layers.Conv2D(128, (3, 3), activation='swish',
                           padding='same',input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), activation='swish', padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(ks.layers.Dropout(0.3))

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

model.add(ks.layers.Conv2D(512, (3, 3), activation='swish', padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(512, (3, 3), activation='swish', padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Conv2D(1024, (3, 3), activation='relu', padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(1024, (3, 3), activation='relu', padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(1024, activation='swish'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.4))

model.add(ks.layers.Dense(1024, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.4))

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

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 32, 32, 128)	3,584
batch_normalization_10 (BatchNormalization)	(None, 32, 32, 128)	512
conv2d_9 (Conv2D)	(None, 32, 32, 128)	147,584
batch_normalization_11 (BatchNormalization)	(None, 32, 32, 128)	512
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0
dropout_6 (Dropout)	(None, 16, 16, 128)	0
conv2d_10 (Conv2D)	(None, 16, 16, 256)	295,168
batch_normalization_12 (BatchNormalization)	(None, 16, 16, 256)	1,024
conv2d_11 (Conv2D)	(None, 16, 16, 256)	590,080
batch_normalization_13 (BatchNormalization)	(None, 16, 16, 256)	1,024
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 256)	0
dropout_7 (Dropout)	(None, 8, 8, 256)	0
conv2d_12 (Conv2D)	(None, 8, 8, 512)	1,180,160
batch_normalization_14 (BatchNormalization)	(None, 8, 8, 512)	2,048
conv2d_13 (Conv2D)	(None, 8, 8, 512)	2,359,808
batch_normalization_15 (BatchNormalization)	(None, 8, 8, 512)	2,048
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 512)	0
dropout_8 (Dropout)	(None, 4, 4, 512)	0
conv2d_14 (Conv2D)	(None, 4, 4, 1024)	4,719,616
batch_normalization_16 (BatchNormalization)	(None, 4, 4, 1024)	4,096
conv2d_15 (Conv2D)	(None, 4, 4, 1024)	9,438,208
batch_normalization_17 (BatchNormalization)	(None, 4, 4, 1024)	4,096
dropout_9 (Dropout)	(None, 4, 4, 1024)	0
flatten_1 (Flatten)	(None, 16384)	0
dense_3 (Dense)	(None, 1024)	16,778,240
batch_normalization_18 (BatchNormalization)	(None, 1024)	4,096
dropout_10 (Dropout)	(None, 1024)	0
dense_4 (Dense)	(None, 1024)	1,049,600
batch_normalization_19 (BatchNormalization)	(None, 1024)	4,096
dropout_11 (Dropout)	(None, 1024)	0
dense_5 (Dense)	(None, 10)	10,250

Total params: 36,595,850 (139.60 MB)
Trainable params: 36,584,074 (139.56 MB)
Non-trainable params: 11,776 (46.00 KB)

```
data_generator = ks.preprocessing.image.ImageDataGenerator(
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    rotation_range=20,
    shear_range=0.15,
    zoom_range=0.2,
    fill_mode='nearest'
)

x_train_scaled_gen = data_generator.flow(x_train_scaled, y_train)
```

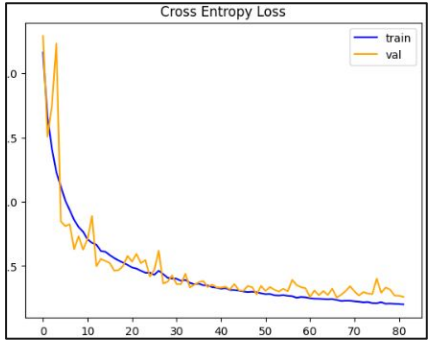
```
model.compile(optimizer='Adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

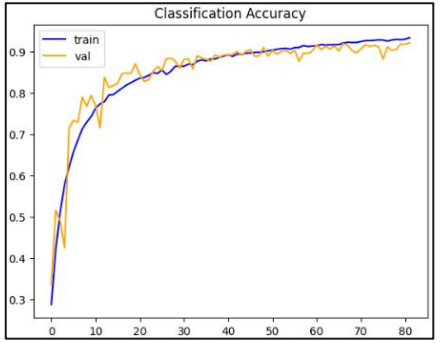
```
callback_val_loss = EarlyStopping(monitor="val_loss", patience=15)
callback_val_accuracy = EarlyStopping(monitor="val_accuracy", patience=15)
```

```
history = model.fit(x_train_scaled_gen, epochs=200,
                    batch_size= 64, validation_data=(x_val_scaled, y_val),
                    callbacks=[callback_val_loss, callback_val_accuracy])
```

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

```
> 91.350
```





Este es nuestro modelo ganador, el cual cuenta con una profundidad mayor que todas las anteriores. Esta cuenta con:

- **Pares** de capas convolucionales de 128, 256, 512 y 1024 filtros, creando una estructura más compleja que las anteriores y elevando en gran medida la cantidad de parámetros.
- Se ha incrementado las capas densas finales, teniendo en este caso dos capas de 1024 neuronas.
- Estas modificaciones sobre la red, suman una cantidad de parámetros de casi 37 millones.
- Se aplicó data augmentation con las mismas transformaciones de los modelos anteriores, y además se agregó variación de zoom del 20%.

Como resultado, hemos llegado a un modelo que tiene un AUC de **91.35%**, sobre el dataset de test. Superior a los anteriores. Pero la gran pregunta es: ¿En términos de eficiencia de recursos computacionales también es el mejor? ya que incrementamos casi en 5 veces la cantidad de parámetros respecto del modelo anterior y solo logramos incrementar en 0.9%

FIN