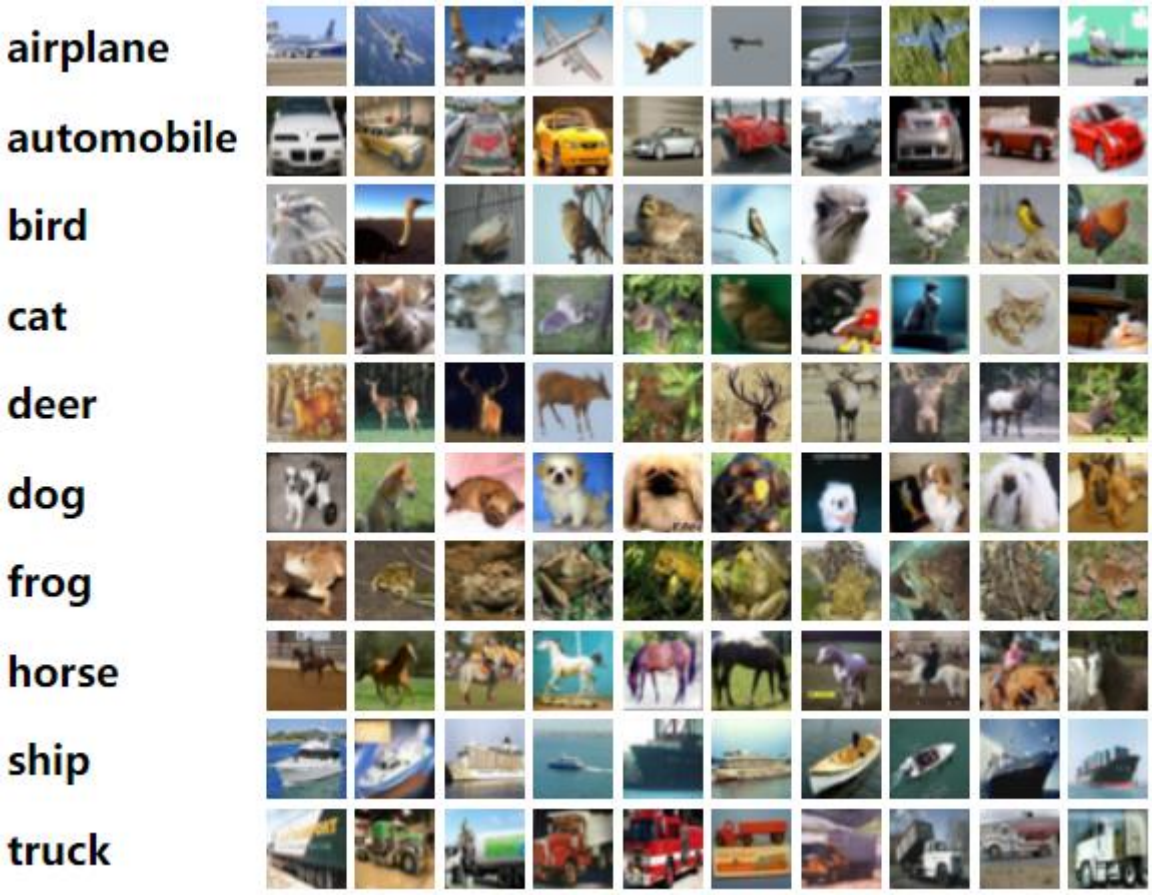
ENTREGABLE 4 – VISIÓN POR COMPUTADOR

CLASIFICACIÓN DE IMÁGENES - CIFAR10



ALUMNO: Simone Solieri

TUTORES: Isidre Royo, Valentina Borgonovi

1 INDICE

2 INTRODUCCIÓN

El objetivo de este proyecto es clasificar las imágenes del dataset CIFAR10, partiendo de una red neuronal muy sencilla proporcionada por los tutores, y presentar un mínimo de 10 experimentos cuya finalidad es investigar distintas combinaciones de elementos y técnicas para mejorar progresivamente la accuracy del modelo, hasta alcanzar la accuracy mínima requerida del 80% en el dataset de test.

Para considerar cumplido el objetivo del proyecto, es necesario que dicha accuracy se obtenga únicamente gracias a la propia arquitectura de la red, es decir, sin recurrir a técnicas avanzadas, como Data Augmentation o Transfer Learning.

Para el autor de este documento, este es el primer proyecto de Deep Learning abordado autónomamente, y el principal objetivo de este trabajo es poner a la prueba todos los conocimientos adquiridos en clase. Es ambición personal del alumno conseguir un modelo con accuracy mínima del 90%.

El lector encontrará un total de 12 experimentos. La primera mitad de ellos tiene como objetivo encontrar un modelo sólido que alcance la accuracy mínima del 80% solo con cambios en la arquitectura y los parámetros. La segunda mitad está dedicada a la implementación de técnicas avanzadas (especialmente Data Augmentation), para intentar alcanzar una accuracy del 90%. Considerando todos los experimentos, la accuracy máxima en el dataset de test lograda solo con la arquitectura es del 87,7%, mientras que la precisión máxima absoluta es del 90,7%, lograda con Data Augmentation.

Al principio de cada experimento se presentan los objetivos que se quieren conseguir, y un resumen de las implementaciones o cambios principales del experimento (por ejemplo los cambios principales aportados respecto al arquitectura precedente). Después se encuentra la ejecución detallada y una sección donde se comentan los resultados obtenidos, se evidencian los problemas encontrados, y se sacan las observaciones que introducen y justifican las implementaciones del experimento sucesivo.

En el último apartado del documento se encuentra una recopilación detallad de todas las conclusiones más importantes que se pueden extraer de este proyecto.

3 EXPERIMENTOS

3.1 EXPERIMENTO 0 - ARQUITECTURA BASE (60,6% ACCURACY)

Objetivos del experimento

El objetivo de este experimento es cuantificar cual es el punto de partida con el cual empezamos el proyecto. Se entrena un modelo utilizando la misma red que ha sido proporcionada por los tutores del proyecto y se evalúa su accuracy. De esta forma podremos entender la efectividad de los cambios implementados en los experimentos sucesivos.

Implementaciones y cambios realizados

Como dicho anteriormente, no se ha realizado ningún cambio en la arquitectura de la red. Simplemente se han añadido callbacks de Earlystopping y ModelCheckpoint para que el entrenamiento pare en el momento adecuado, guardando el mejor modelo disponible.

Arquitectura y parámetros

```
[ ]  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'))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 32)                262176

 dense_1 (Dense)             (None, 10)                330

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

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

Entrenamiento

```
# EarlyStopping
earlystop_loss = EarlyStopping(monitor='val_loss', patience=5)
earlystop_acc = EarlyStopping(monitor='val_accuracy', patience=5)

# ModelCheckpoint
checkpoint_path = '/content/drive/MyDrive/Master_Data_Science/Entregables/Entregable_04-Deep_Learning/Model
modelcheckpoint_best_acc = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True)
```

```
[ ]  history = model.fit(x_train_scaled, y_train,
                         epochs=500,
                         batch_size= 512,
                         validation_data=(x_val_scaled, y_val),
                         callbacks=[modelcheckpoint_best_acc, earlystop_loss, earlystop_acc])
```

```
[ ]  elapsed_time = datetime.timedelta(seconds=(time.perf_counter() - t))

     print('Tiempo de entrenamiento:', elapsed_time)
```

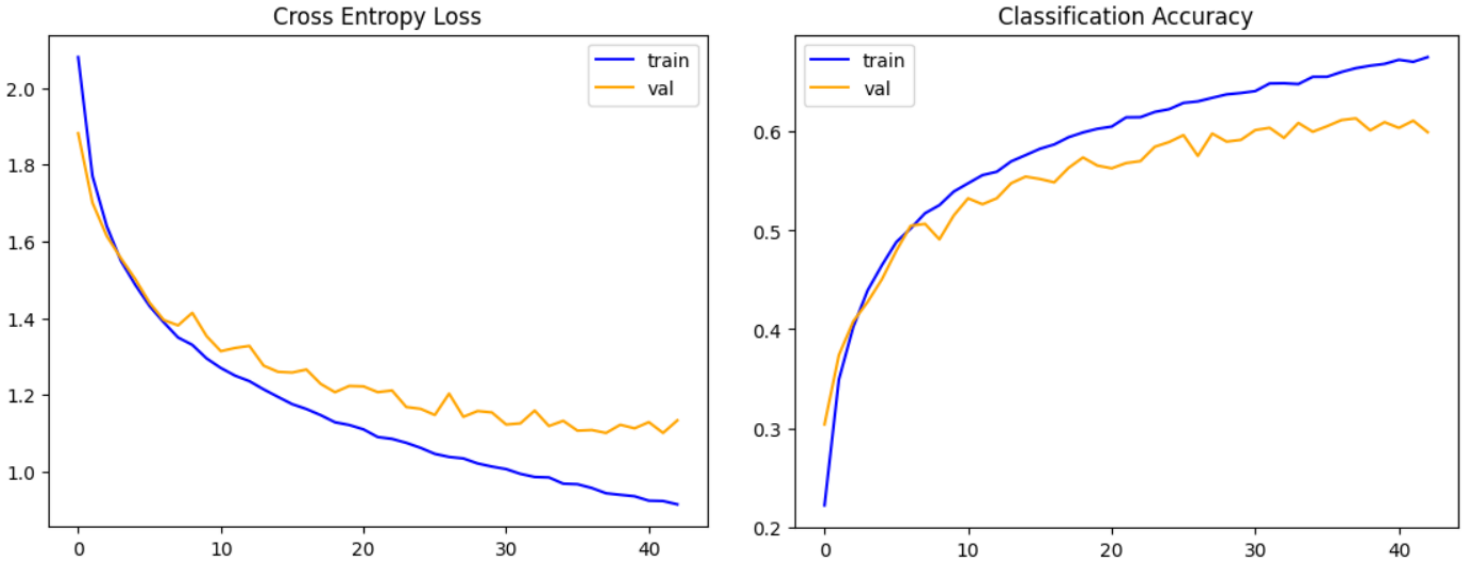
```
Tiempo de entrenamiento: 0:01:00.518927
```

Resultados

Accuracy en el dataset de Test

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

```
> 60.580
```



Observaciones sobre los resultados

El modelo inicial que nos viene dado está en evidente situación saca una accuracy bastante baja, solamente 60,5%.

Sobre todo se nota que la accuracy es muy baja también en el dataset de train. Eso denota que la arquitectura actual de la red neuronal es demasiado simple para clasificar correctamente las imágenes (UNDER-FITTING). Evidentemente, tanto la parte de extracción de features, como la parte de clasificación, carecen de la profundidad y complejidad necesarias para llevar a cabo la tarea.

Actualmente la red clasifica las imágenes basándose únicamente en las características extraídas por una sola capa de convolución, es decir, los bordes/formas de los sujetos. Para aumentar el rendimiento del modelo, el primer paso a seguir es sin duda aumentar la capacidad de extracción de características, agregando más bloques de convolución y max pooling. Obviamente, la parte de la red que se ocupa de la clasificación también tendrá que adaptarse al nivel de complejidad alcanzado por la parte de extracción de features. Eso se hará añadiendo más neuronas y/o capas densas.

3.2 EXPERIMENTO 1 - AÑADIR PROFUNDIDAD (74,8% ACCURACY)

Objetivos del experimento

El objetivo es construir una red con una complejidad que sea a la altura de la tarea que le pedimos. Idealmente queremos alcanzar la accuracy más alta posible, por el momento al menos en el dataset de train.

Implementaciones y cambios realizados

El enfoque del experimento consiste en aumentar la capacidad de la red tanto en extracción de features, como en clasificación. En el diseño de la arquitectura se ha intentado seguir la misma lógica que se puede encontrar en la gran mayoría de las redes para clasificación de imágenes que hemos visto durante el Master. En particular, en la arquitectura utilizada en este experimento, se pueden encontrar muchas similitudes con la famosa VGG-16. Siguen los detalles:

- Tres bloques de convoluciones y al termine de cada bloque un max-pooling para reducir los parámetros y quedarnos con la información más relevante. Los primeros dos bloques, que extraen características más genéricas (bordes y formas), son de 2 convoluciones; el tercer bloque (el que extrae texturas), en cambio, presenta 3 convoluciones. Las razones detrás de esta elección residen principalmente en la experiencia y la literatura, que parecen demostrar que las capas de convolución funcionan mejor cuando se disponen en bloques con cantidad de layers creciente, moviéndose del bottom hacía el top de la red. De esta forma, los bloques que extraen features de detalle (texturas, pequeños objetos etc.) tienen más capas de los que extraen características más genéricas.
- No se han puesto más de 3 bloques de convoluciones y max-pooling para evitar de reducir demasiado el número de pixeles, ya que nos quedaremos con imágenes de 2x2, y parece demasiado arriesgado concentrar toda la información en tan pocos pixeles.
- Los filtros de las convoluciones, en analogía con el patrón que se suele ver en la literatura, van de menor a mayor, se duplican de un bloque al otro (así las convoluciones que extraen features pequeñas aplican más filtros para capturar más detalles) y son múltiples de 8 (32, 64 ,128 etc.).
- También se ha aumentado el número de neuronas en la capa densa de clasificación para que la parte de la red que se encarga de la clasificación sea suficientemente compleja para aprovechar todas las características extraídas precedentemente. Para que el modelo no sea más complicado de lo que realmente hace falta, inicialmente se ha puesto solamente una capa densa de clasificación de 256 neuronas. Si los resultados evidenciarán que sigue habiendo underfitting, se añadirán más capas y neuronas en los experimentos sucesivos.

Arquitectura y parámetros

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 conv2d_6 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 256)               524544

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 961706 (3.67 MB)
Trainable params: 961706 (3.67 MB)
Non-trainable params: 0 (0.00 Byte)
```

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

Entrenamiento

```
history = model.fit(x_train_scaled, y_train,
                    epochs=100,
                    batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[modelcheckpoint_best_acc, earlystop_loss])
```

```
[ ] elapsed_time = datetime.timedelta(seconds=(time.perf_counter() - t))

    print('Tiempo de entrenamiento:', elapsed_time)
```

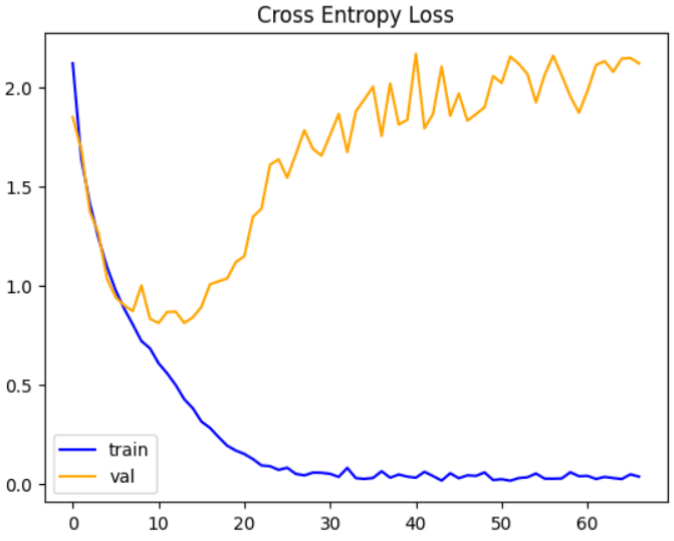
```
Tiempo de entrenamiento: 0:04:37.355798
```

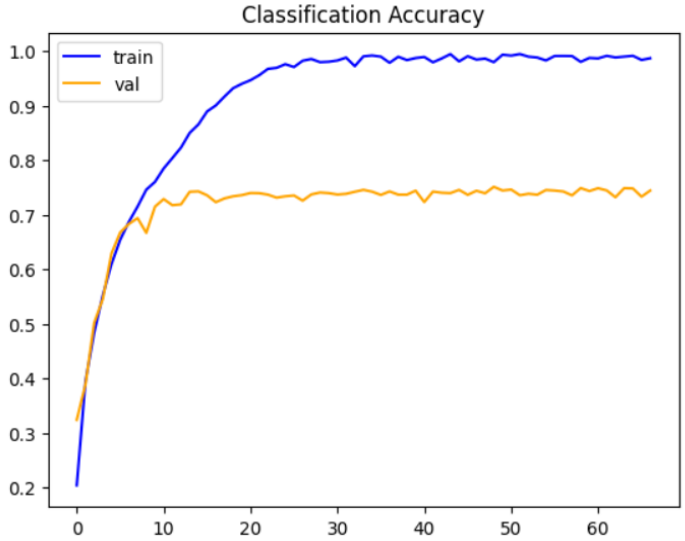
Resultados

Accuracy en el dataset de Test

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

```
> 74.760
```





Observaciones sobre los resultados

- El problema del underfitting ha sido solucionado: la accuracy de train ha subido muchísimo (desde 0.657 a 0.993), gracias a la mayor complejidad de la red.
- Muchísimo overfitting, con accuracy de validación de solamente 0.762
- Accuracy de test 76,2% --> objetivo todavía no cumplido (80% minimo).
- Se observa un accuracy muy buena en el dataset de train, pero oscila mucho y no alcanza a llegar a un valor estable, tanto la accuracy como el error. Esta instabilidad podría ser causada principalmente por un learning rate demasiado grande que no permite que el entrenamiento converja

correctamente en el punto mínimo de la función de coste. Eso se puede solucionar reduciendo el learning rate durante el entrenamiento, cuando el error deja de mejorar.

- El error de validación presenta una tendencia muy irregular: primero disminuye, llega a un mínimo y luego vuelve a subir, cuando en realidad el comportamiento esperado sería que una vez alcanzado el mínimo se mantenga más o menos constante, no pudiendo disminuir más a causa del sobreajuste. Esta irregularidad podría deberse a valores de salida demasiado altos en las capas más profundas de la red, a causa de la acumulación de varias capas con funciones de activación. Este problema es muy conocido en literatura y suele ser relevante en modelos con al menos 8 o 9 capas con funciones de activación (nuestra red tiene 8). La solución más frecuente consiste en aplicar la Batch Normalization.

En los siguientes dos experimentos se intentará estabilizar el entrenamiento, para crear unas condiciones más optímales para el correcto funcionamiento de las técnicas anti overfitting. Se ha elegido separar la aplicación de las dos estrategias mencionadas anteriormente en dos experimentos diferentes, para apreciar mejor los beneficios aportados por cada una de ellas.

3.3 Experimento 2 - Reducción progresiva del learning rate (76,9% accuracy)

Objetivos del experimento

- Estabilizar el entrenamiento y llegar a una correcta convergencia del mismo.
- Intentar subir aún más la accuracy de train antes de aplicar técnicas anti overfitting.

Implementaciones y cambios realizados

Introducido callback ReduceLROnPlateau para reducir el learning rate solamente en el momento en que el entrenamiento lo requiere, es decir, sin alargar demasiado el tiempo de entrenamiento y no capar la capacidad de aprendizaje en la parte inicial, en la cual el learning rate anterior parece ser adecuado.

Arquitectura y parámetros

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 conv2d_6 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 256)               524544

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 961706 (3.67 MB)
Trainable params: 961706 (3.67 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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

Entrenamiento

Dado que por el momento estamos interesados en estabilizar el entrenamiento y maximizar la accuracy de train, los callbacks monitorearán las métricas del dataset de train.

```
# Callbacks
earlystop_loss = EarlyStopping(monitor='loss', patience=10)
earlystop_acc = EarlyStopping(monitor='accuracy', patience=5)
reduce_lr_callback = ReduceLROnPlateau(monitor='loss',factor=0.1,patience=3, min_lr=0.0001)

# ModelCheckpoint
checkpoint_path = '/content/drive/MyDrive/Master_Data_Science/Entregables/Entregable_04-Deep_L
modelcheckpoint_best_acc = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_
```

```
history = model.fit(x_train_scaled, y_train,
                    epochs=200,
                    batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[modelcheckpoint_best_acc, reduce_lr_callback, earlystop_acc])
```

```
elapsed_time = datetime.timedelta(seconds=(time.perf_counter() - t))

print('Tiempo de entrenamiento:', elapsed_time)
```

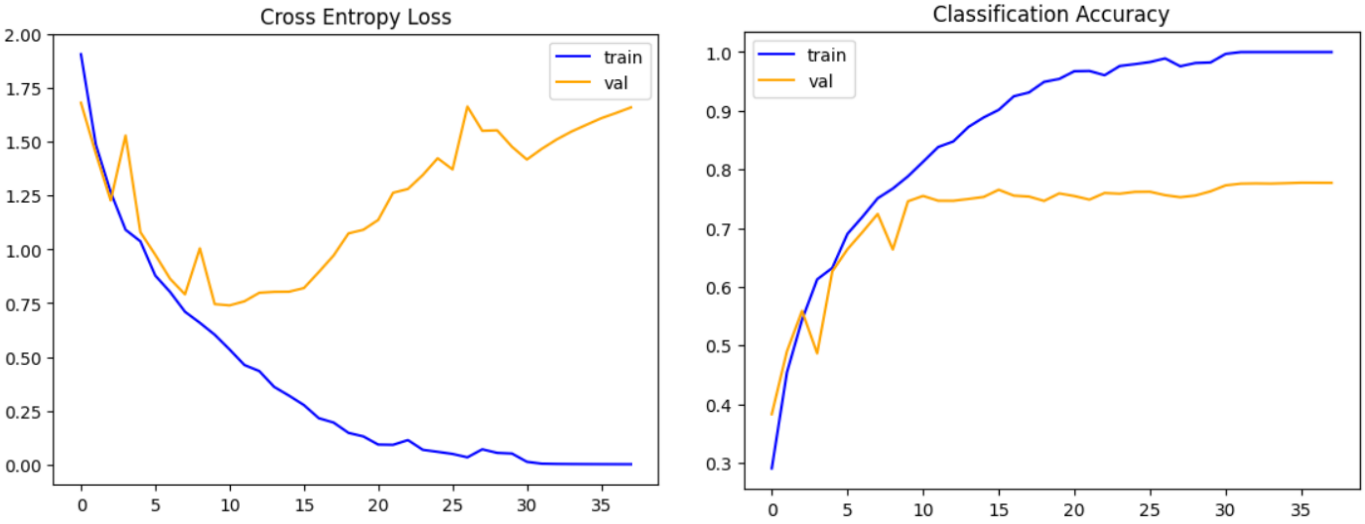
```
Tiempo de entrenamiento: 0:02:56.866348
```

Resultados

Accuracy en el dataset de Test

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

```
> 76.880
```



Observaciones sobre los resultados

- La introducción del callback ReduceLROnPlateau ha dado los resultados esperados y el entrenamiento converge a valores estables de accuracy tanto en train, cuanto en validación.
- Se ha alcanzado una accuracy de train próxima al 100%. El modelo está evidentemente haciendo muchísimo overfitting, pero, por el otro lado, este resultado nos indica que de momento la red no necesita ulterior complejidad.
- El error de validación, como esperado, sigue presentando la misma irregularidad del experimento anterior, ya que probablemente será solucionado con la implementación de la batch normalization.

3.4 Experimento 3 – Batch Normalization (76,3% accuracy)

Objetivos del experimento

Estabilizar el entrenamiento y el modelo antes de aplicar dropout y otras técnicas anti overfitting.

Implementaciones y cambios realizados

Se ha aplicado Batch Normalization después de cada capa con función de activación.

Arquitectura y parámetros

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

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dense(10, activation='softmax'))
```

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (Batch  (None, 32, 32, 32)        128
 Normalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 batch_normalization_1 (Bat  (None, 32, 32, 32)        128
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 batch_normalization_2 (Bat  (None, 16, 16, 64)        256
 chNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 batch_normalization_3 (Bat  (None, 16, 16, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 batch_normalization_4 (Bat  (None, 8, 8, 128)         512
 chNormalization)

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 batch_normalization_5 (Bat  (None, 8, 8, 128)         512
 chNormalization)

 conv2d_6 (Conv2D)           (None, 8, 8, 128)         147584

 batch_normalization_6 (Bat  (None, 8, 8, 128)         512
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 256)               524544

 batch_normalization_7 (Bat  (None, 256)               1024
 chNormalization)

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 965034 (3.68 MB)
Trainable params: 963370 (3.67 MB)
Non-trainable params: 1664 (6.50 KB)
```

Entrenamiento

```
# Callbacks
earlystop_loss = EarlyStopping(monitor='loss', patience=10)
earlystop_acc = EarlyStopping(monitor='accuracy', patience=5)
reduce_lr_callback = ReduceLROnPlateau(monitor='loss',factor=0.1,patience=3, min_lr=0.0001)

# ModelCheckpoint
checkpoint_path = '/content/drive/MyDrive/Master_Data_Science/Entregables/Entregable_04-Deep_Learning/Mod
modelcheckpoint_best_acc = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True)
```

```
history = model.fit(x_train_scaled, y_train,
                    epochs=200,
                    batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[modelcheckpoint_best_acc, reduce_lr_callback, earlystop_acc])
```

```
[ ]  elapsed_time = datetime.timedelta(seconds=(time.perf_counter() - t))

     print('Tiempo de entrenamiento:', elapsed_time)
```

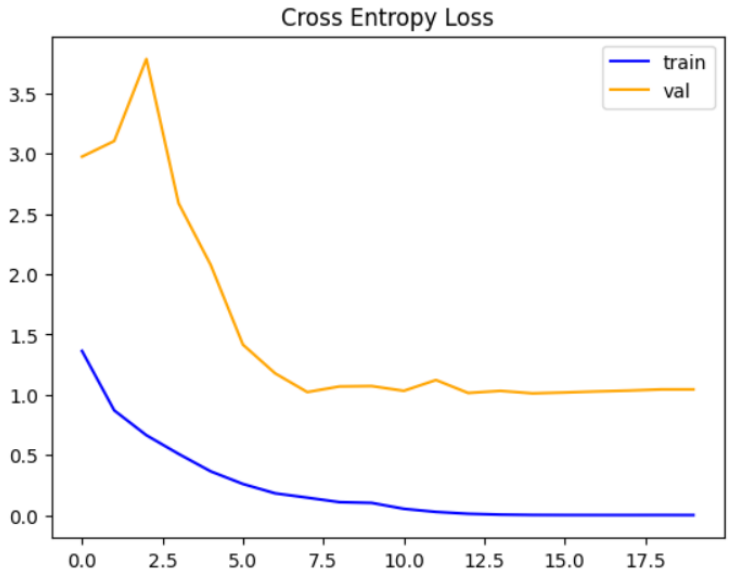
```
Tiempo de entrenamiento: 0:02:41.741210
```

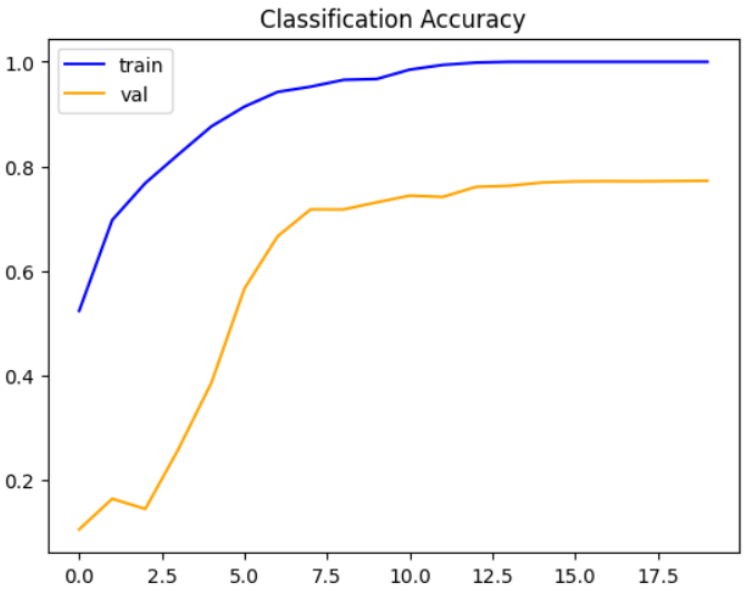
Resultados

Accuracy en el dataset de Test

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

```
> 76.290
```





Observaciones sobre los resultados

- Observando los resultados se nota evidentemente que la aplicación de la Batch Normalization ha estabilizado mucho el entrenamiento y ha conseguido regularizar el error de validación. El comportamiento de la curva ahora es como se esperaba.
- Hasta ahora hemos conseguido un modelo estable que pero todavía no alcanza el objetivo mínimo de accuracy en el dataset de test, a causa del overfitting. Es evidente que el modelo está memorizando perfectamente los datos de entrenamiento, pero no es capaz de generalizar correctamente para clasificar los datos que nunca ha visto. Este es un buen momento para empezar a aplicar técnicas especificas contra el overfitting, las cuales introduciendo ruido en el entrenamiento, harán acercar las curvas de validación y train.

3.5 EXPERIMENTO 4 – DROPOUT 30% (81,2% ACCURACY)

Objetivos del experimento

Reducir el overfitting y llegar al objetivo del 80% de accuracy en el dataset de test

Implementaciones y cambios realizados

Introducido Dropout del 30% después de cada bloque de convoluciones y en la parte de clasificación.

Arquitectura y parámetros

```
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'))
model.add(ks.layers.BatchNormalization())
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.BatchNormalization())
model.add(ks.layers.Conv2D(64, (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(0.3))

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

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.3))
model.add(ks.layers.Dense(10, activation='softmax'))
```

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (Batch  (None, 32, 32, 32)        128
 Normalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 batch_normalization_1 (Bat  (None, 32, 32, 32)        128
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

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

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 batch_normalization_2 (Bat  (None, 16, 16, 64)        256
 chNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 batch_normalization_3 (Bat  (None, 16, 16, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 batch_normalization_4 (Bat  (None, 8, 8, 128)         512
 chNormalization)

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 batch_normalization_5 (Bat  (None, 8, 8, 128)         512
 chNormalization)

 conv2d_6 (Conv2D)           (None, 8, 8, 128)         147584

 batch_normalization_6 (Bat  (None, 8, 8, 128)         512
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 dropout_2 (Dropout)         (None, 4, 4, 128)         0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 256)               524544

 batch_normalization_7 (Bat  (None, 256)               1024
 chNormalization)

 dropout_3 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 965034 (3.68 MB)
Trainable params: 963370 (3.67 MB)
Non-trainable params: 1664 (6.50 KB)
```

Entrenamiento

```
# # EarlyStopping
earlystop_val_acc = EarlyStopping(monitor='val_accuracy', patience=5)
earlystop_val_loss = EarlyStopping(monitor='val_loss', patience=5)


reduce_lr_callback = ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience=5, min_lr=0.0001)

# ModelCheckpoint
checkpoint_path = '/content/drive/MyDrive/Master_Data_Science/Entregables/Entregable_04-Deep_Learning/Models/Model_Che
modelcheckpoint_best_acc = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True)
```

```
history = model.fit(x_train_scaled, y_train,
                    epochs=300,
                    batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[modelcheckpoint_best_acc, reduce_lr_callback, earlystop_val_loss, earlystop_val_acc ])
```

```
[ ]  elapsed_time = datetime.timedelta(seconds=(time.perf_counter() - t))

     print('Tiempo de entrenamiento:', elapsed_time)
```

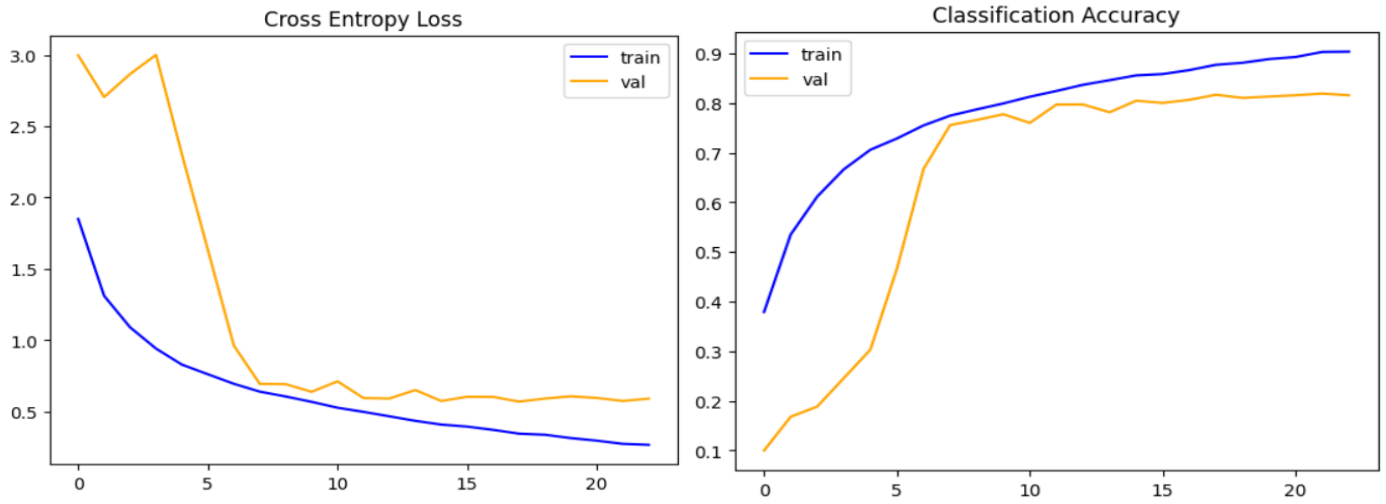
```
Tiempo de entrenamiento: 0:03:44.288125
```

Resultados

Accuracy en el dataset de Test

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

```
> 81.250
```



Observaciones sobre los resultados

La implementación del dropout, como esperado, ha reducido el overfitting y ha permitido obtener el primer modelo que alcanza la accuracy mínima requerida por el proyecto. Sin embargo sigue habiendo bastante overfitting ya que las curvas de las métricas de train y validación terminan siendo todavía bastante separadas. Esto sugiere que se podría aumentar un poco el porcentaje de dropout para que en cada Batch la cantidad

de neuronas “desactivadas” sea mayor, haciendo el modelo resultante más robusto y con una mayor capacidad de generalización. Por eso, el siguiente experimento se pondrá el dropout al 50%.

3.6 EXPERIMENTO 5 – DROPOUT 50% (84,9% ACCURACY)

Objetivos del experimento

Conseguir una mayor reducción del overfitting y con eso una mayor accuracy del modelo en el dataset de test.

Implementaciones y cambios realizados

- Introducido Dropout del 50% después de cada bloque de convoluciones y en la parte de clasificación.
- Bajado el learning rate mínimo dentro del callback ReduceLROnPlateau a 0.00001 para intentar aterrizar aún mejor dentro del mínimo de la función de coste.

La arquitectura se muestra en la siguiente pagina por falta de espacio.

Arquitectura y parámetros

```
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'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Conv2D(64, (3, 3), strides=1,
          activation='relu',padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (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(0.5))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1,
          activation='relu',padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), strides=1,
          activation='relu',padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (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(0.5))

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(10, activation='softmax'))
```

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

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_1 (BatchNormalization)	(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 (BatchNormalization)	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36928
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73856
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147584
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_6 (Conv2D)	(None, 8, 8, 128)	147584
batch_normalization_6 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 256)	524544
batch_normalization_7 (BatchNormalization)	(None, 256)	1024
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 10)	2570

```
Total params: 965034 (3.68 MB)
Trainable params: 963370 (3.67 MB)
Non-trainable params: 1664 (6.50 KB)
```

Entrenamiento

```
# # EarlyStopping
earlystop_val_acc = EarlyStopping(monitor='val_accuracy', patience=10)
earlystop_val_loss = EarlyStopping(monitor='val_loss', patience=10)


reduce_lr_callback = ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience=5, min_lr=0.00001, )

# ModelCheckpoint
checkpoint_path = '/content/drive/MyDrive/Master_Data_Science/Entregables/Entregable_04-Deep_Learning/Models/Model_Chec
modelcheckpoint_best_acc = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True)
```

```
history = model.fit(x_train_scaled, y_train,
                    epochs=300,
                    batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[modelcheckpoint_best_acc, reduce_lr_callback, earlystop_val_loss, earlystop_val_acc ])
```

```
[ ]  elapsed_time = datetime.timedelta(seconds=(time.perf_counter() - t))

     print('Tiempo de entrenamiento:', elapsed_time)
```

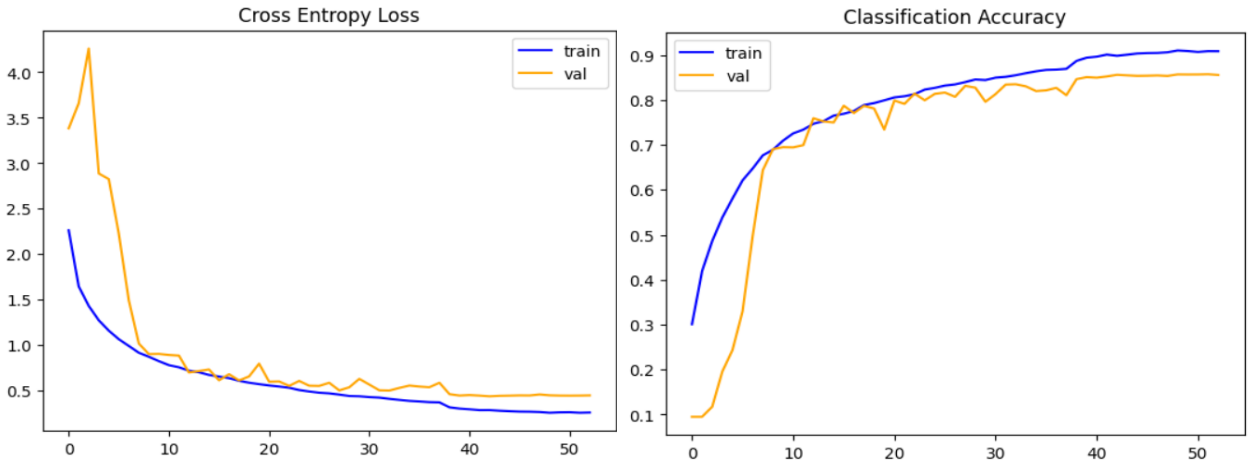
```
Tiempo de entrenamiento: 0:07:24.766500
```

Resultados

Accuracy en el dataset de Test

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

```
> 84.940
```



Observaciones sobre los resultados

Como se puede notar desde los gráficos, aumentar el porcentaje de dropout ha dado los resultados esperados: el overfitting ha sido mayormente reducido y el modelo ha ganado + 3,7 puntos de accuracy en el dataset de test. Ahora ya tenemos consolidado un modelo que cumple con los objetivos mínimos del proyecto, simplemente gracias a su arquitectura. A partir de este momento el objetivo será obtener un modelo que alcance el 90% de accuracy en el dataset de test. Antes de probar la implementación de técnicas avanzadas, vemos si esta arquitectura se puede optimizar un poco más: en el siguiente experimento se aplicará la técnica del Weight Decay para intentar disminuir ulteriormente el overfitting y subir la accuracy.

3.7 Experimento 6 – Weight Decay (86,9%)

Objetivos del experimento

Reducir aún más el overfitting y llegar a una mejor accuracy del modelo.

Implementaciones y cambios realizados

- Implementada regularización L2 de Keras con el factor de penalización por defecto (0,01).
- Cambiado el optimizador a SGD con momentum, ya que la experiencia indica que la regularización L2 funciona mejor con este optimizador.

Arquitectura y parámetros

```
model = ks.Sequential()
model.add(ks.layers.Conv2D(32, (3, 3), strides=1,
          activation='relu',kernel_regularizer=l2(0.01),
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(32, (3, 3), strides=1,
          activation='relu',kernel_regularizer=l2(0.01),
          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(64, (3, 3), strides=1,
          activation='relu',kernel_regularizer=l2(0.01),
          padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), strides=1,
          activation='relu',kernel_regularizer=l2(0.01),
          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), strides=1,
          activation='relu', kernel_regularizer=l2(0.01),
          padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), strides=1,
          activation='relu',kernel_regularizer=l2(0.01),
          padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), strides=1,
          activation='relu',kernel_regularizer=l2(0.01),
          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.Flatten())
model.add(ks.layers.Dense(256,
          activation='relu',kernel_regularizer=l2(0.01)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(10, activation='softmax'))
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_1 (BatchNormalization)	(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 (BatchNormalization)	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36928
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73856
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147584
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_6 (Conv2D)	(None, 8, 8, 128)	147584
batch_normalization_6 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 256)	524544
batch_normalization_7 (BatchNormalization)	(None, 256)	1024
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 10)	2570

Total params: 965034 (3.68 MB)
Trainable params: 963370 (3.67 MB)
Non-trainable params: 1664 (6.50 KB)

```
model.compile(optimizer= SGD(learning_rate=0.01, momentum=0.9),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Entrenamiento

```
# # EarlyStopping
earlystop_val_acc = EarlyStopping(monitor='val_accuracy', patience=50)
earlystop_val_loss = EarlyStopping(monitor='val_loss', patience=10)

reduce_lr_callback = ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience=5, min_lr=0.00001, )

# ModelCheckpoint
checkpoint_path = '/content/drive/MyDrive/Master_Data_Science/Entregables/Entregable_04-Deep_Learning/Models/Model_Chec
modelcheckpoint_best_acc = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True)
```

```
history = model.fit(x_train_scaled, y_train,
                    epochs=300,
                    batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[modelcheckpoint_best_acc, reduce_lr_callback, earlystop_val_loss, earlystop_val_acc ])
```

```
[ ]  elapsed_time = datetime.timedelta(seconds=(time.perf_counter() - t))

     print('Tiempo de entrenamiento:', elapsed_time)
```

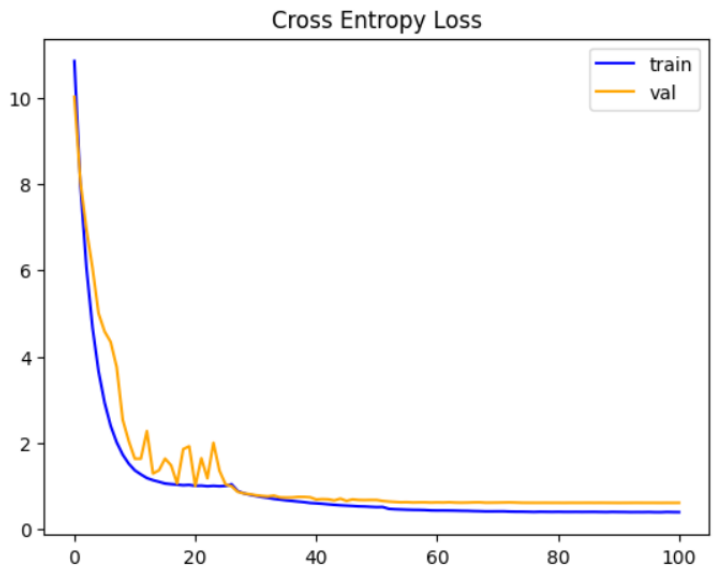
```
Tiempo de entrenamiento: 0:12:44.584543
```

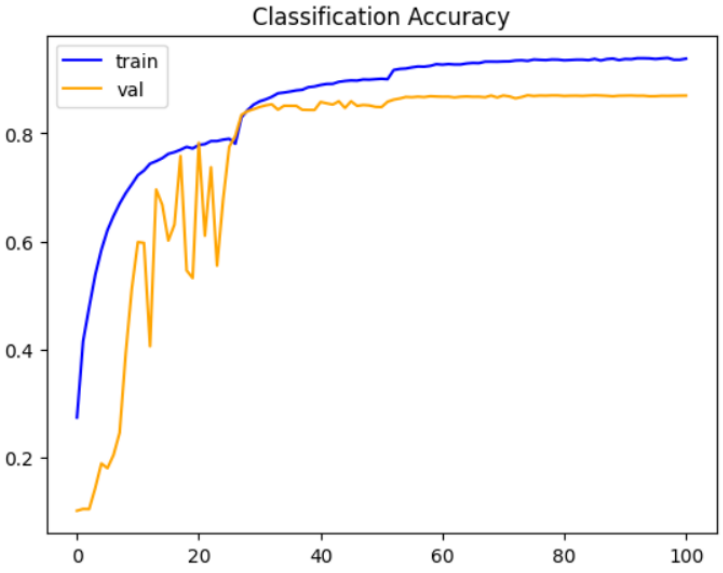
Resultados

Accuracy en el dataset de Test

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

```
> 86.970
```





Observaciones sobre los resultados

La introducción de la Weight Decay no parece haber dado los efectos deseados en cuanto a la reducción del overfitting, que incluso aumenta ligeramente. Sin embargo, se observa que los niveles de accuracy alcanzados por el modelo son mayores. Esto podría atribuirse al mayor learning rate inicial. De hecho, SGD tiene un learning rate predeterminado de 0,01 en comparación con el 0,001 de Adam. El aumento de este parámetro permitió razonablemente que el modelo saliera de un mínimo local al que probablemente convergía en los experimentos anteriores.

A partir de este momento se decide dejar la arquitectura así como está ahora, para empezar a introducir técnicas avanzadas. En el siguiente experimento se aplicará Data Augmentation a la red actual.

3.8 Experimento 7 – Arquitectura sencilla y Data Augmentation (82,3% accuracy)

Objetivos del experimento

Utilizar Data Augmentation para aumentar la cantidad y calidad de los datos de entrenamiento, para ayudar el modelo a generalizar mejor, reducir el overfitting y alcanzar una mejor accuracy en los datos con los cuales no ha sido entrenado.

Implementaciones y cambios

Se ha implementado la técnica de Data Augmentation utilizando exactamente la misma red del experimento anterior.

Arquitectura y parámetros

```
model = ks.Sequential()
model.add(ks.layers.Conv2D(32, (3, 3), strides=1,
          activation='relu',kernel_regularizer=l2(0.01),
          padding='same', input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(32, (3, 3), strides=1,
          activation='relu',kernel_regularizer=l2(0.01),
          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(64, (3, 3), strides=1,
          activation='relu',kernel_regularizer=l2(0.01),
          padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), strides=1,
          activation='relu',kernel_regularizer=l2(0.01),
          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), strides=1,
          activation='relu', kernel_regularizer=l2(0.01),
          padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), strides=1,
          activation='relu',kernel_regularizer=l2(0.01),
          padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), strides=1,
          activation='relu',kernel_regularizer=l2(0.01),
          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.Flatten())
model.add(ks.layers.Dense(256,
          activation='relu',kernel_regularizer=l2(0.01)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(10, activation='softmax'))
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_1 (BatchNormalization)	(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 (BatchNormalization)	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36928
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73856
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147584
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_6 (Conv2D)	(None, 8, 8, 128)	147584
batch_normalization_6 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 256)	524544
batch_normalization_7 (BatchNormalization)	(None, 256)	1024
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 10)	2570

Total params: 965034 (3.68 MB)
Trainable params: 963370 (3.67 MB)
Non-trainable params: 1664 (6.50 KB)

```
model.compile(optimizer= SGD(learning_rate=0.01, momentum=0.9),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
[ ]  batch_size_train = 512
```

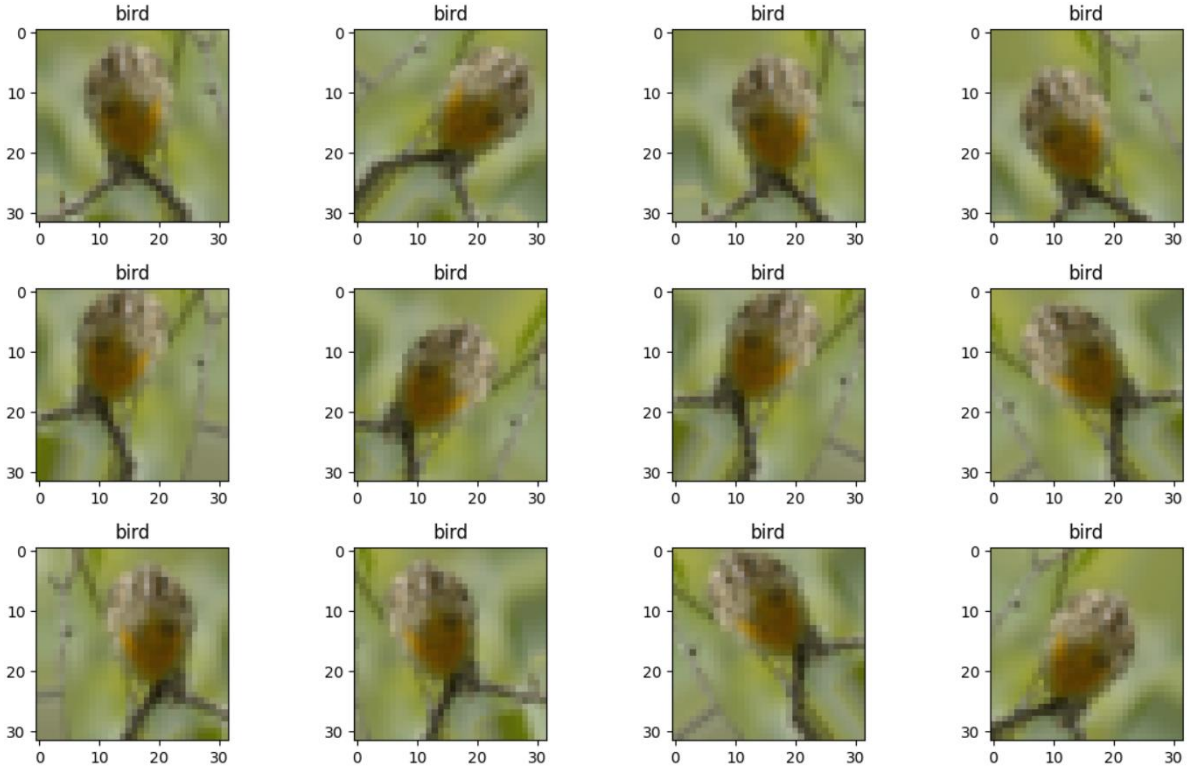
```
[ ]  batch_size_val = 128
```

```
[ ]  train_datagen = ImageDataGenerator(
        rescale=1./255,
        horizontal_flip=True,
        rotation_range=40,
        shear_range=0.2,
        height_shift_range=0.1,
        width_shift_range=0.1,
        fill_mode='nearest',
        # Definamos las transformaciones
        )

    train_generator = train_datagen.flow(
        x_train,
        y_train,
        batch_size=batch_size_train
    )
```

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

Imágenes creadas por ImageDataGenerator



Entrenamiento

```
# # EarlyStopping
earlystop_val_acc = EarlyStopping(monitor='val_accuracy', patience=50)
earlystop_val_loss = EarlyStopping(monitor='val_loss', patience=10)

reduce_lr_callback = ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience=5, min_lr=0.00001, )

# ModelCheckpoint
checkpoint_path = '/content/drive/MyDrive/Master_Data_Science/Entregables/Entregable_04-Deep_Learning/Model
modelcheckpoint_best_acc = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True)
```

```
history = model.fit(
        train_generator,
        steps_per_epoch= 40000 // batch_size_train,
        epochs=200,
        validation_data=validation_generator,
        validation_steps=10000 // batch_size_val,
        callbacks=[modelcheckpoint_best_acc, reduce_lr_callback, earlystop_val_loss, earlystop_val_acc ])
```

```
[ ]  elapsed_time = datetime.timedelta(seconds=(time.perf_counter() - t))

     print('Tiempo de entrenamiento:', elapsed_time)
```

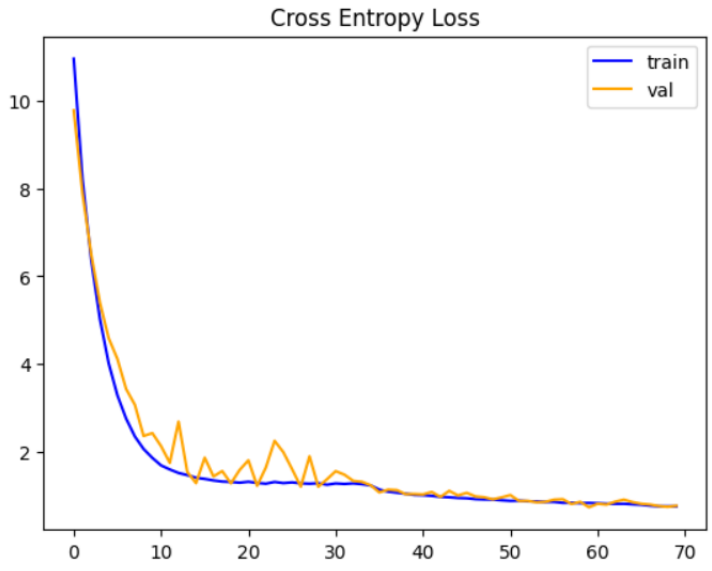
```
Tiempo de entrenamiento: 0:33:58.159262
```

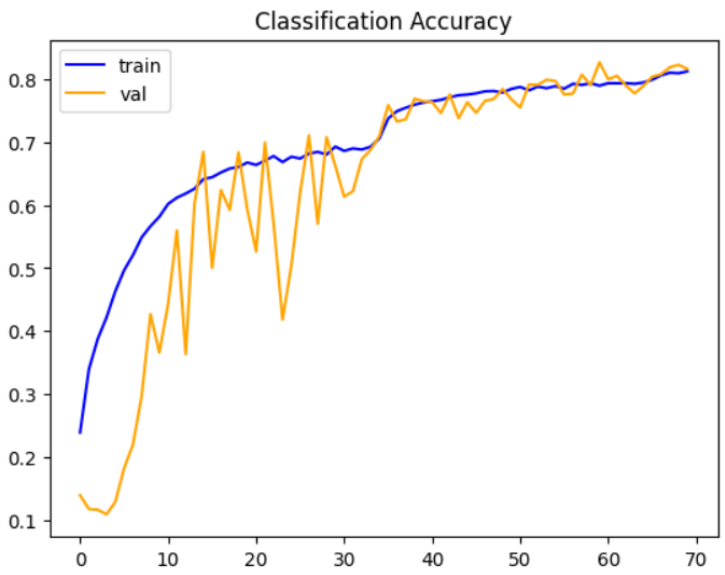
Resultados

Accuracy en el dataset de Test

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

```
> 82.300
```





Observaciones sobre los resultados

- La implementación de la Data Augmentation no ha dado los resultados esperados y el modelo en vez de mejorar ha empeorado.
- Dado que las trasformaciones dentro del ImageDataGenerator han sido intencionalmente seleccionadas para no generar grandes modificaciones o distorsiones en las imágenes generadas, es muy probable que la causa del empeoramiento del modelo no esté en una mala implementación de la técnica de Data Augmentation.
- El gráfico de la accuracy tiene un aspecto muy inusual, ya que la accuracy de validación supera la de train en muchos momentos distintos. Además, observamos cómo con la Data Augmentation la accuracy de train se ha reducido considerablemente en comparación con los experimentos anteriores.
  Esto parece indicar que a la red le está resultando difícil fitear las imágenes augmentadas, que son muchas más y mucho más generalizadas que las del dataset original. En otras palabras, parece que la red actual es demasiado simple para poder "resistir" al ruido introducido por la Data Augmentation.

Vistos los resultados de este experimento, el siguiente paso será individuar una nueva arquitectura que sea lo suficientemente compleja, como para sacar beneficio de la aplicación de la Data Augmentation.

3.9 Experimento 8 – Arquitectura más compleja y Data Augmentation (86,0% accuracy)

Introducción al experimento

Dados los resultados del experimento anterior, se buscó una nueva arquitectura más compleja, capaz de dar una accuracy similar a la obtenida con la arquitectura anterior, antes de la aplicación del Data Augmentation. Sobre todo, se aumentó la profundidad y densidad de la parte de clasificación. Se probaron diferentes configuraciones, en las que las únicas diferencias estaban en el número de capas y neuronas de la parte de clasificación y/o diversos números de filtros entre las convoluciones. Para evitar publicar experimentos repetitivos y muy similares, se ha decidido mostrar sólo la mejor arquitectura resultante de la serie de estos experimentos.

El Experimento 8 se dividirá en dos partes: la primera muestra el entrenamiento de la nueva arquitectura más compleja (sin Data Augmentation), mientras que la segunda parte muestra los resultados de la aplicación de Data Augmentation.

3.9.1 Parte 1: Arquitectura más compleja sin Data Augmentation (86,2% accuracy)

Objetivos del experimento

Construir un modelo solido con accuracy mínima de test del 85%, pero con suficiente complejidad para sacar provecho de un sucesivo Data Augmentation.

Implementaciones y cambios

Aumentada complejidad de la parte de clasificación: tres capas densas, las primeras dos de 1024 neuronas y la ultima de 512, con Dropout a 0,5 y Batch Normalization. Aumentado el número de filtros de los bloques de convoluciones.

Arquitectura y parámetros

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(64, (3, 3), strides=1,
          activation='relu',padding='same',
          input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), strides=1,
          activation='relu',padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1,
          activation='relu',padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (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(0.5))

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

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(1024, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(1024, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(10, activation='softmax'))
```

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

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1792
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36928
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)	73856
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147584
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)	295168
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 256)	1024
conv2d_5 (Conv2D)	(None, 8, 8, 256)	590080
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 256)	1024
conv2d_6 (Conv2D)	(None, 8, 8, 256)	590080
batch_normalization_6 (BatchNormalization)	(None, 8, 8, 256)	1024
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 1024)	4195328
batch_normalization_7 (BatchNormalization)	(None, 1024)	4096
dropout_3 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 1024)	1049600
batch_normalization_8 (BatchNormalization)	(None, 1024)	4096
dropout_4 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524800
batch_normalization_9 (BatchNormalization)	(None, 512)	2048
dropout_5 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130

```
Total params: 7525194 (28.71 MB)
Trainable params: 7517770 (28.68 MB)
Non-trainable params: 7424 (29.00 KB)
```

Entrenamiento

```
# # EarlyStopping
earlystop_val_acc = EarlyStopping(monitor= 'val_accuracy', patience=10)
earlystop_val_loss = EarlyStopping(monitor='val_loss', patience=10)
reduce_lr_callback = ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience=5, min_lr=0.00001, )

# ModelCheckpoint
checkpoint_path = '/content/drive/MyDrive/Master_Data_Science/Entregables/Entregable_04-Deep_Learning/Models/Model_Ch
modelcheckpoint_best_acc = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True)
```

```
history = model.fit(x_train_scaled, y_train,
                    epochs=150,
                    batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[modelcheckpoint_best_acc, reduce_lr_callback, earlystop_val_loss, earlystop_val_acc])
```

```
[ ]  elapsed_time = datetime.timedelta(seconds=(time.perf_counter() - t))

     print('Tiempo de entrenamiento:', elapsed_time)
```

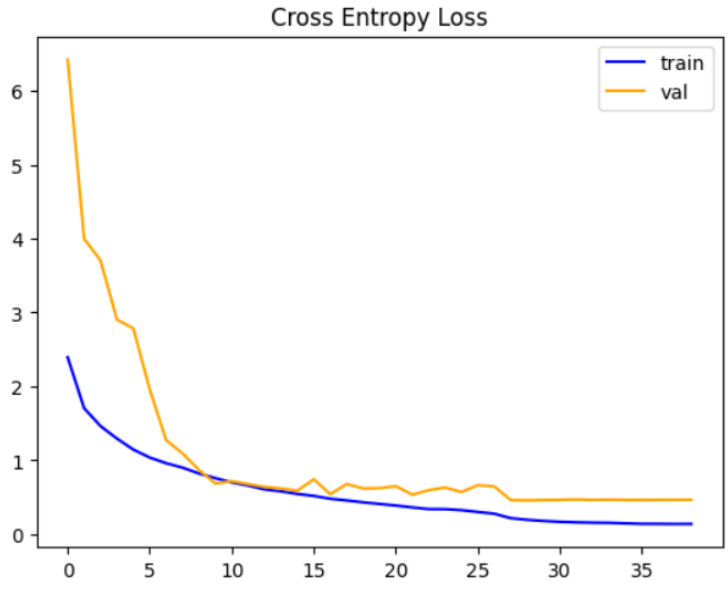
```
Tiempo de entrenamiento: 0:11:02.248051
```

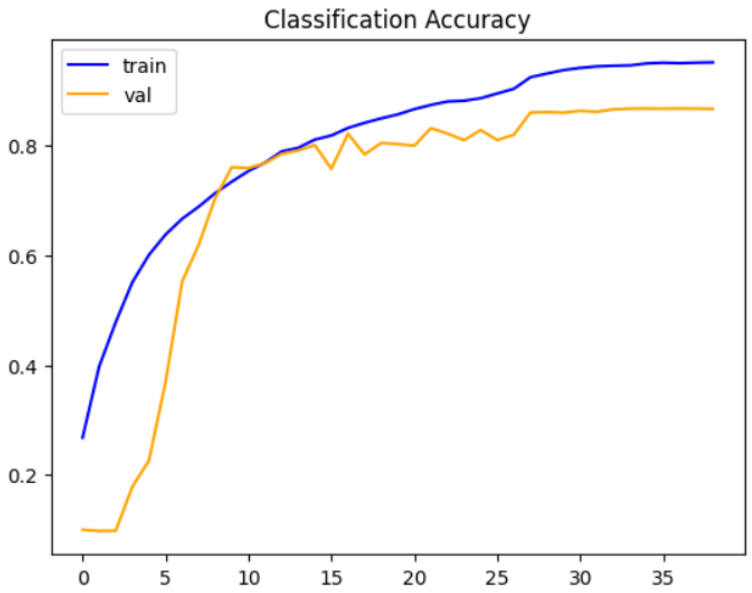
Resultados

Accuracy en el dataset de Test

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

```
> 86.200
```





Observaciones sobre los resultados

Hemos conseguido un modelo solido que alcanza una muy buena accuracy, parecida a la del experimento 7, además sin utilizar Weight Decay. A pesar del dropout al 50%, todavía hay bastante overfitting. La implementación de la Data Augmentation debería ayudar a reducirlo.

3.9.2 Parte 2: Arquitectura más compleja con Data Augmentation (86,0% accuracy)

Objetivos del experimento

Mejorar la capacidad de generalización del modelo y reducir el overfitting para alcanzar una mejor accuracy y robustez del modelo.

Implementaciones y cambios

Implementado Data Augmentation con la nueva arquitectura más compleja.

Arquitectura y parámetros

La arquitectura y el optimizador utilizados se pueden encontrar en la parte 1 de este experimento, ya que son los mismos. A continuación se muestra la configuración del generador de imágenes con ImageDataGenerator.

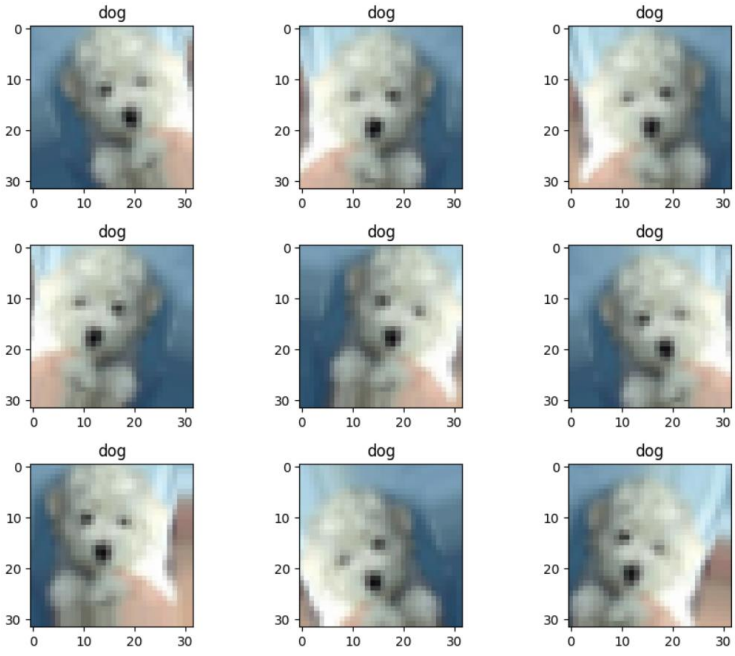
```
batch_size_train = 512
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    rotation_range=20,
    height_shift_range=0.1,
    width_shift_range=0.1,
    fill_mode='nearest',
    # Definamos las transformaciones
    )

train_generator = train_datagen.flow(
    x_train,
    y_train,
    batch_size=batch_size_train
)
```

```
batch_size_val = 128
```

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



Entrenamiento

```
# # EarlyStopping
earlystop_val_acc = EarlyStopping(monitor= 'val_accuracy', patience=15)
earlystop_val_loss = EarlyStopping(monitor='val_loss', patience=15)
reduce_lr_callback = ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience=5, min_lr=0.00001, )

# ModelCheckpoint
checkpoint_path = '/content/drive/MyDrive/Master_Data_Science/Entregables/Entregable_04-Deep_Learning/Model
modelcheckpoint_best_acc = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True)
```

```
history = model.fit(
        train_generator,
        steps_per_epoch= 40000 // batch_size_train,
        epochs=200,
        validation_data=validation_generator,
        validation_steps=10000 // batch_size_val,
        callbacks=[modelcheckpoint_best_acc, reduce_lr_callback, earlystop_val_loss, earlystop_val_acc ])
```

```
[ ]  elapsed_time = datetime.timedelta(seconds=(time.perf_counter() - t))

     print('Tiempo de entrenamiento:', elapsed_time)
```

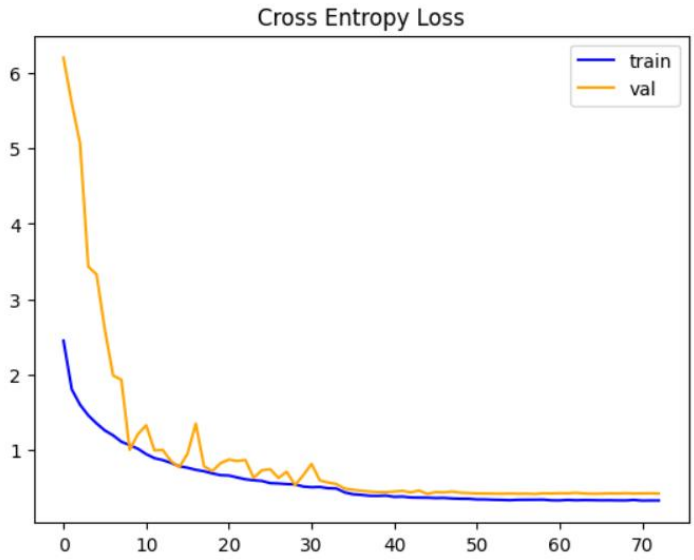
```
Tiempo de entrenamiento: 0:39:51.303679
```

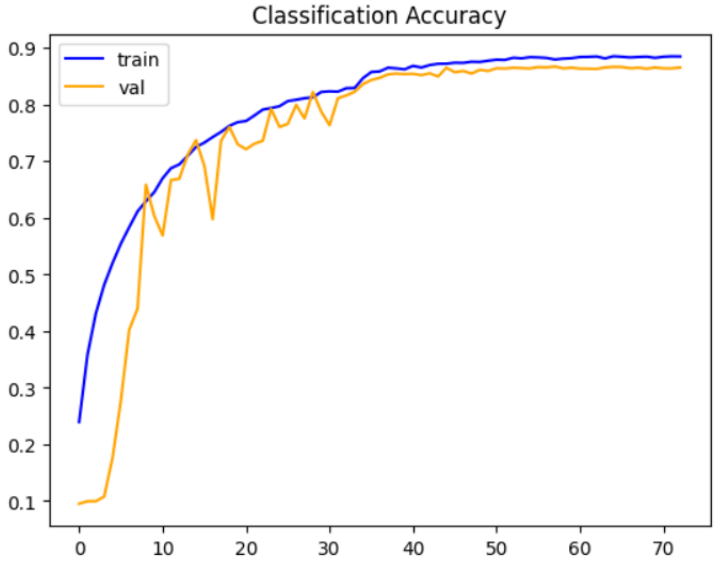
Resultados

Accuracy en el dataset de Test

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

```
> 86.040
```





Observaciones sobre los resultados

Se puede notar que gracias a la introducción del Data Augmentation el overfitting se ha reducido muchísimo.

Sin embargo, también se observa que la accuracy no ha mejorado, por el contrario, ha disminuido ligeramente. La explicación de este resultado podría ser que se ha alcanzado un mínimo local de la función de coste. En este caso, cualquier mejora en la calidad de los datos introducida por el Data Augmentation no lleva a mejoras sustanciales en el modelo, dado que el problema está en el camino seguido por el entrenamiento, no tanto en los datos.

Para verificar esta hipótesis e intentar salir de un posible mínimo local, en el próximo experimento se aumentará el learning rate inicial y la patience dentro de los callbacks.

3.10 Experimento 9 – Aumento del learning rate y Data Augmentation (89,5%)

Objetivos del experimento

Comprobar si estamos en un mínimo local de la función de coste e intentar salir de ello.

Implementaciones y cambios

- Aumentado el learning rate inicial a 0,01
- Aumentado la patience dentro de los callbacks.

Arquitectura y parámetros

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(64, (3, 3), strides=1,
          activation='relu',padding='same',
          input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), strides=1,
          activation='relu',padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1,
          activation='relu',padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (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(0.5))

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

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(1024, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(1024, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(10, activation='softmax'))
```

```
model.compile(optimizer= Adam(learning_rate=0.01),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1792
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36928
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)	73856
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147584
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)	295168
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 256)	1024
conv2d_5 (Conv2D)	(None, 8, 8, 256)	590080
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 256)	1024
conv2d_6 (Conv2D)	(None, 8, 8, 256)	590080
batch_normalization_6 (BatchNormalization)	(None, 8, 8, 256)	1024
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 1024)	4195328
batch_normalization_7 (BatchNormalization)	(None, 1024)	4096
dropout_3 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 1024)	1049600
batch_normalization_8 (BatchNormalization)	(None, 1024)	4096
dropout_4 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524800
batch_normalization_9 (BatchNormalization)	(None, 512)	2048
dropout_5 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130

Total params: 7525194 (28.71 MB)
Trainable params: 7517770 (28.68 MB)
Non-trainable params: 7424 (29.00 KB)

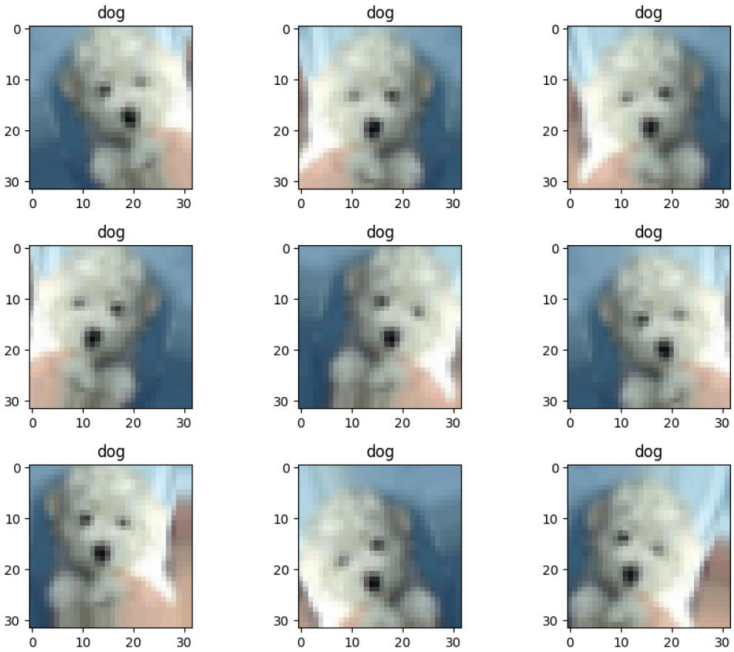
```
batch_size_train = 512
```

```
batch_size_val = 128
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    rotation_range=20,
    height_shift_range=0.1,
    width_shift_range=0.1,
    fill_mode='nearest',
    # Definamos las transformaciones
    )

train_generator = train_datagen.flow(
    x_train,
    y_train,
    batch_size=batch_size_train
)
```

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



Entrenamiento

```
# # EarlyStopping
earlystop_val_acc = EarlyStopping(monitor= 'val_accuracy', patience=30)
earlystop_val_loss = EarlyStopping(monitor='val_loss', patience=30)

reduce_lr_callback = ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience=20, min_lr=1e-5, )

# ModelCheckpoint
checkpoint_path = '/content/drive/MyDrive/Master_Data_Science/Entregables/Entregable_04-Deep_Learning/Mode
modelcheckpoint_best_acc = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True)
```

```
history = model.fit(
        train_generator,
        steps_per_epoch= 40000 // batch_size_train,
        epochs=200,
        validation_data=validation_generator,
        validation_steps=10000 // batch_size_val,
        callbacks=[modelcheckpoint_best_acc, reduce_lr_callback, earlystop_val_loss, earlystop_val_acc ])
```

```
    print('Tiempo de entrenamiento:', elapsed_time)
```

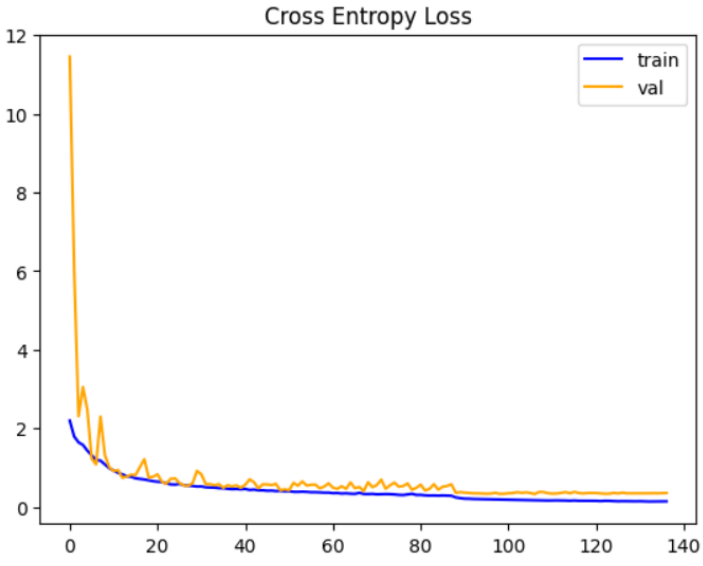
```
Tiempo de entrenamiento: 1:13:06.159567
```

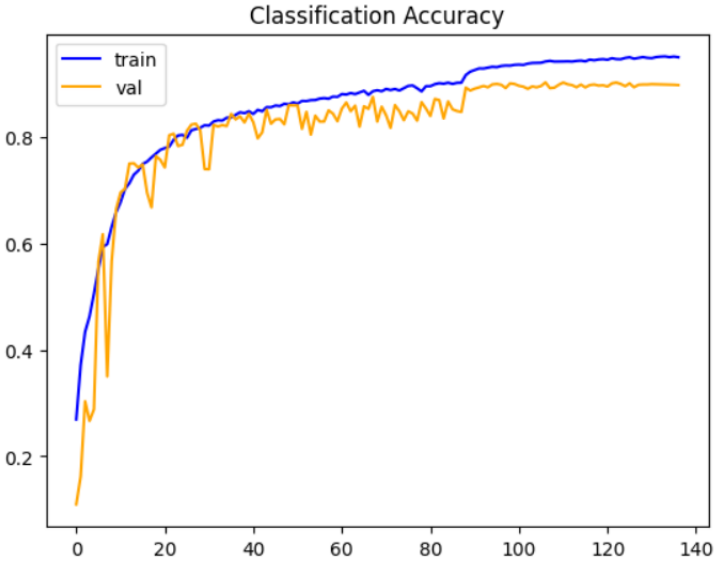
Resultados

Accuracy en el dataset de Test

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

```
> 89.490
```





Observaciones sobre los resultados

La combinación de mayor learning rate y Data Augmentation hizo posible alcanzar un nuevo nivel de accuracy de test, igual a 89,4. Evidentemente estábamos en un mínimo local y el aumento del learning rate permitió salir de ello. Desafortunadamente, con este experimento no es posible visualizar cuál es el aporte real del Data Augmentation por sí solo, por esta razón en el próximo experimento se entrenará la misma red, con los mismos parámetros, pero sin Data Augmentation, para cuantificar el mejoramiento aportado únicamente por el aumento del learning rate.

3.11 Experimento 10 – Aumento learning rate sin Data Augmentation (87,7% accuracy)

Objetivo del experimento

Cuantificar la aportación del sólo aumento del learning rate, para poder también evaluar mejor los beneficios aportados por el Data Augmentation.

Implementaciones y cambios

- Aumentado el learning rate inicial a 0,01
- Aumentado la patience dentro de los callbacks.
- NO se utiliza Data Augmentation

Arquitectura y parámetros

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(64, (3, 3), strides=1,
          activation='relu',padding='same',
          input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), strides=1,
          activation='relu',padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.5))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1,
          activation='relu',padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (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(0.5))

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

model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(1024, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(1024, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(10, activation='softmax'))
```

```
model.compile(optimizer= Adam(learning_rate=0.01),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1792
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36928
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)	73856
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147584
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)	295168
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 256)	1024
conv2d_5 (Conv2D)	(None, 8, 8, 256)	590080
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 256)	1024
conv2d_6 (Conv2D)	(None, 8, 8, 256)	590080
batch_normalization_6 (BatchNormalization)	(None, 8, 8, 256)	1024
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 1024)	4195328
batch_normalization_7 (BatchNormalization)	(None, 1024)	4096
dropout_3 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 1024)	1049600
batch_normalization_8 (BatchNormalization)	(None, 1024)	4096
dropout_4 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524800
batch_normalization_9 (BatchNormalization)	(None, 512)	2048
dropout_5 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130

Total params: 7525194 (28.71 MB)
Trainable params: 7517770 (28.68 MB)
Non-trainable params: 7424 (29.00 KB)

Entrenamiento

```
# # EarlyStopping
earlystop_val_acc = EarlyStopping(monitor= 'val_accuracy', patience=30)
earlystop_val_loss = EarlyStopping(monitor='val_loss', patience=30)
reduce_lr_callback = ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience=20, min_lr=1e-5, )

# ModelCheckpoint
checkpoint_path = '/content/drive/MyDrive/Master_Data_Science/Entregables/Entregable_04-Deep_Learning/Models/Model_Ch
modelcheckpoint_best_acc = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True)
```

```
history = model.fit(x_train_scaled, y_train,
                    epochs=200,
                    batch_size= 512,
                    validation_data=(x_val_scaled, y_val),
                    callbacks=[modelcheckpoint_best_acc, reduce_lr_callback, earlystop_val_loss, earlystop_val_acc])
```

```
[ ]  elapsed_time = datetime.timedelta(seconds=(time.perf_counter() - t))

     print('Tiempo de entrenamiento:', elapsed_time)
```

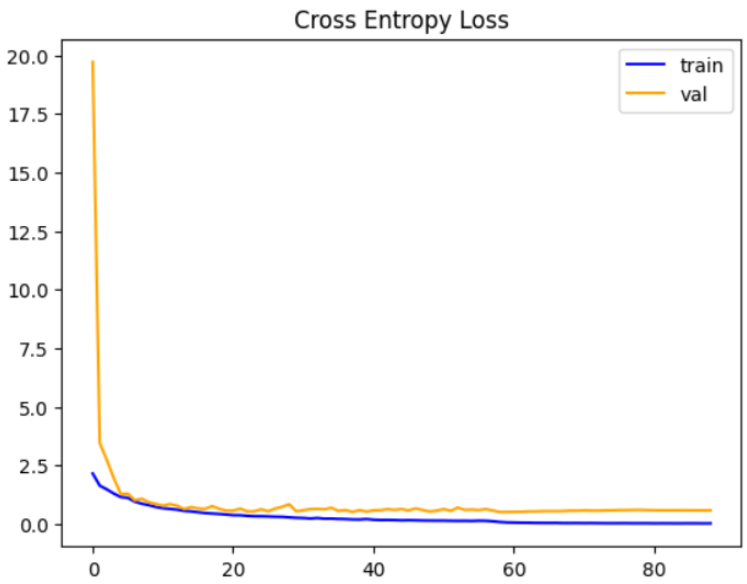
```
Tiempo de entrenamiento: 0:23:38.262891
```

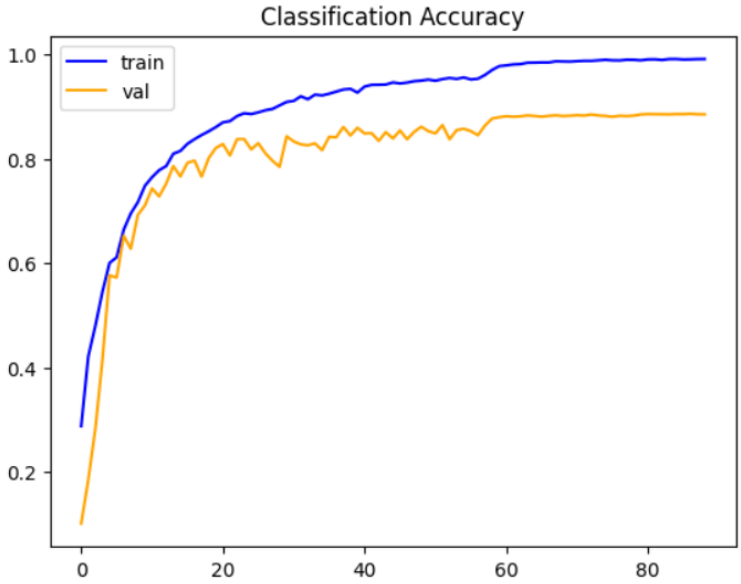
Resultados

Accuracy en el dataset de Test

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

```
> 87.710
```





Observaciones sobre los resultados

Gracias a este experimento, y comparando los resultados con los de los experimentos 8 y 9 se puede observar que:

- en comparación con la arquitectura inicial (parte 1 del experimento 8), el aumento del learning rate realmente permitió alcanzar un nuevo mínimo de la función de coste, ganando 1,5 puntos de accuracy en el dataset de test (de 86,2 a 87.7).
- Sin la aplicación de Data Augmentation, como se podría imaginar, hay mucho más overfitting en comparación con el experimento anterior.
- La aplicación de Data Augmentation en esta arquitectura y con estos parámetros permitió una buena reducción del sobreajuste e hizo ganar 1,8 puntos de accuracy en el dataset de test, alcanzando el 89,5%.

Ya que faltan solamente 0,5 puntos de accuracy para alcanzar el objetivo del 90%, en el próximo experimento se aportarán modificaciones para intentar conseguirlo.

3.12 Experimento 11 – Más convoluciones y Data Augmentation (90,7% accuracy)

Objetivos del experimento

Llegar al 90% de accuracy en el dataset de test.

Implementaciones y cambios

En el experimento anterior, la red contaba con tres bloques de convoluciones, que según la literatura llegan a extraer detalles de textura en las imágenes que procesan. Una idea para aumentar la calidad del modelo podría ser aumentar aún más la cantidad de características extraídas, de modo que el modelo tenga más información para usar en la clasificación. Para ello, en este experimento agregaremos un cuarto bloque de convoluciones, que podría comenzar a detectar objetos pequeños y otros pequeños detalles. Además, se agrega una capa de convolución adicional en los dos primeros bloques de la red para mejorar también la capacidad de extraer características básicas. El resto de la estructura permanece sin cambios, al igual que los parámetros.

```python
model = ks.Sequential()
model.add(ks.layers.Conv2D(64, (3, 3), strides=1,
          activation='relu',padding='same',
          input_shape=(32,32,3)))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (3, 3), strides=1,
          activation='relu', padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(64, (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(0.5))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1,
          activation='relu', padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (3, 3), strides=1,
          activation='relu', padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(128, (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(0.5))

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

model.add(ks.layers.Conv2D(512, (3, 3), strides=1,
          activation='relu', padding='same'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Conv2D(512, (3, 3), strides=1,
          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='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(1024, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(512, activation='relu'))
model.add(ks.layers.BatchNormalization())
model.add(ks.layers.Dropout(0.5))
model.add(ks.layers.Dense(10, activation='softmax'))
```

```
=================================================================
Total params: 15448842 (58.93 MB)
Trainable params: 15438986 (58.90 MB)
Non-trainable params: 9856 (38.50 KB)
_________________________________________________________________
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 64)	1792
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_1 (Conv2D)	(None, 32, 32, 64)	36928
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_2 (Conv2D)	(None, 32, 32, 64)	36928
batch_normalization_2 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_3 (Conv2D)	(None, 16, 16, 128)	73856
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_4 (Conv2D)	(None, 16, 16, 128)	147584
batch_normalization_4 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_5 (Conv2D)	(None, 16, 16, 128)	147584
batch_normalization_5 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_1 (Dropout)	(None, 8, 8, 128)	0
conv2d_6 (Conv2D)	(None, 8, 8, 256)	295168
batch_normalization_6 (BatchNormalization)	(None, 8, 8, 256)	1024
conv2d_7 (Conv2D)	(None, 8, 8, 256)	590080
batch_normalization_7 (BatchNormalization)	(None, 8, 8, 256)	1024
conv2d_8 (Conv2D)	(None, 8, 8, 256)	590080
batch_normalization_8 (BatchNormalization)	(None, 8, 8, 256)	1024
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
conv2d_9 (Conv2D)	(None, 4, 4, 512)	1180160
batch_normalization_9 (BatchNormalization)	(None, 4, 4, 512)	2048
conv2d_10 (Conv2D)	(None, 4, 4, 512)	2359808
batch_normalization_10 (BatchNormalization)	(None, 4, 4, 512)	2048
dropout_3 (Dropout)	(None, 4, 4, 512)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 1024)	8389632
batch_normalization_11 (BatchNormalization)	(None, 1024)	4096
dropout_4 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 1024)	1049600
batch_normalization_12 (BatchNormalization)	(None, 1024)	4096
dropout_5 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524800
batch_normalization_13 (BatchNormalization)	(None, 512)	2048
dropout_6 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130

```
model.compile(optimizer= Adam(learning_rate=0.01),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

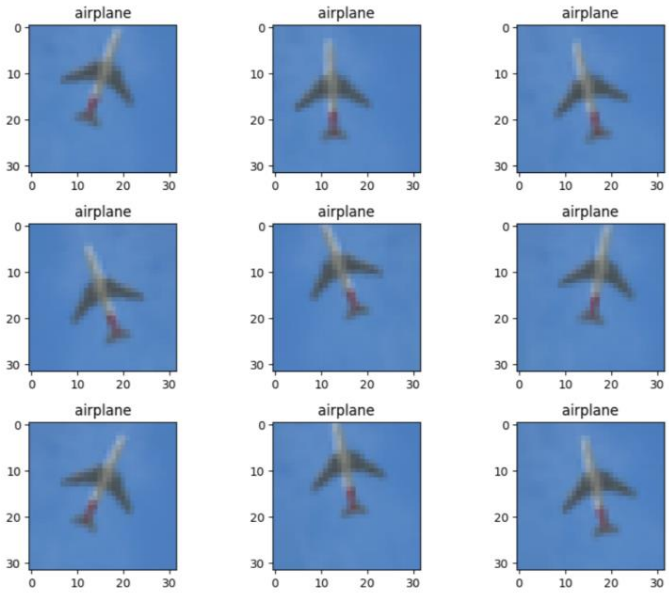
```
batch_size_train = 512
```

```
batch_size_val = 128
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    rotation_range=20,
    height_shift_range=0.1,
    width_shift_range=0.1,
    fill_mode='nearest',
    # Definamos las transformaciones
    )

train_generator = train_datagen.flow(
    x_train,
    y_train,
    batch_size=batch_size_train
)
```

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



Entrenamiento

```
# # EarlyStopping
earlystop_val_acc = EarlyStopping(monitor= 'val_accuracy', patience=30)
earlystop_val_loss = EarlyStopping(monitor='val_loss', patience=30)

reduce_lr_callback = ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience=20, min_lr=1e-5, )

# ModelCheckpoint
checkpoint_path = '/content/drive/MyDrive/Master_Data_Science/Entregables/Entregable_04-Deep_Learning/Mode
modelcheckpoint_best_acc = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True)
```

```
history = model.fit(
        train_generator,
        steps_per_epoch= 40000 // batch_size_train,
        epochs=200,
        validation_data=validation_generator,
        validation_steps=10000 // batch_size_val,
        callbacks=[modelcheckpoint_best_acc, reduce_lr_callback, earlystop_val_loss, earlystop_val_acc ])
```

```
[ ]  elapsed_time = datetime.timedelta(seconds=(time.perf_counter() - t))

     print('Tiempo de entrenamiento:', elapsed_time)
```

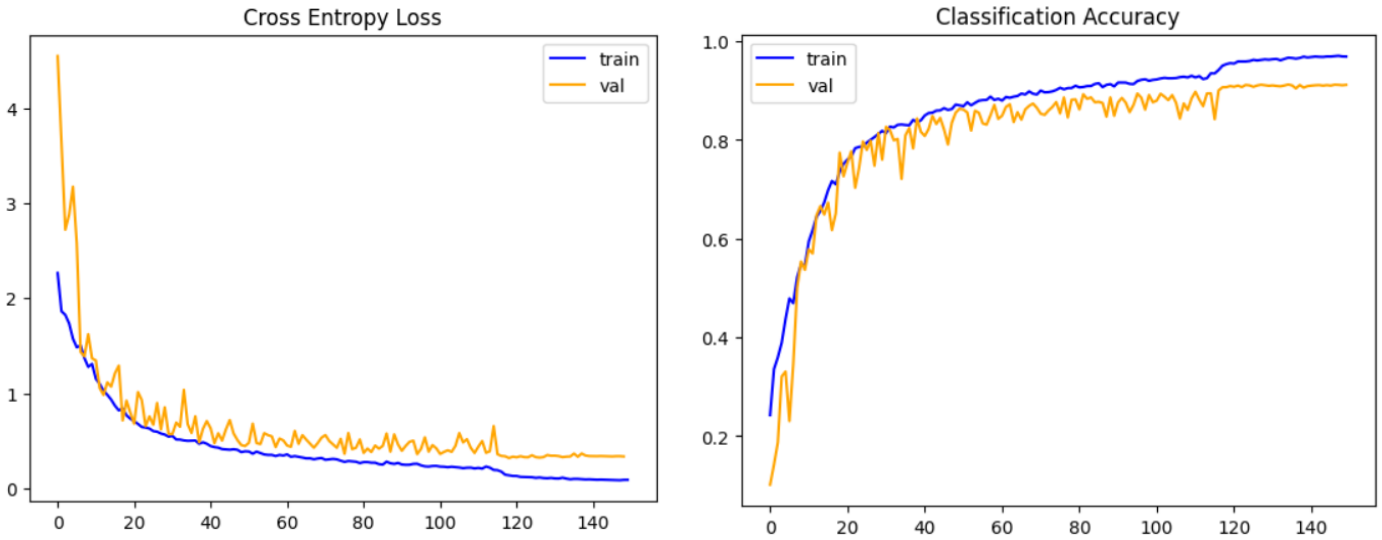
```
Tiempo de entrenamiento: 1:25:37.558605
```

Resultados

Accuracy en el dataset de Test

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

```
> 90.740
```



Observaciones sobre los resultados

Los resultados evidencian que la adición de más convoluciones ha hecho que haya más features disponibles para la clasificación y ha aumentado la accuracy del modelo, que por primera vez desde el inicio del proyecto supera el 90%.

Ciertamente, a partir de esta configuración se podrían probar muchas otras combinaciones para intentar aumentar aún más la precisión del modelo. Sin embargo, como a estas alturas ya se han conseguido todos los objetivos marcados al inicio del proyecto, se decide dedicar el último experimento a la implementación de otra técnica avanzada vista durante el Máster: el Transfer Learning.

3.13 Experimento 12 – Transfer Learning + Data Augmentation (85,9% accuracy)

Objetivo del experimento

Implementar la técnica del Transfer Learning, como la vimos en clase, y ver qué efectos tiene en el rendimiento del modelo.

Implementaciones y cambios

Transfer learning con VGG-16 + Data augmentation: La red utilizada para este experimento consiste en el uso de la misma arquitectura que VGG-16 hasta el tercer MaxPooling, dado que para las dimensiones de las imágenes de CIFAR10, la implementación de más de tres poolings nos dejaría con pocos píxeles. Se congelarán los pesos hasta el segundo MaxPooling, y se entrenarán todos los demás.

La parte restante de la red está hecha a medida para nuestro problema de clasificación específico, emulando prácticamente las mismas características de las arquitecturas que mejores resultados han dado hasta ahora en este proyecto. Los parámetros de Data Augmentation son los mismos de los experimentos precedentes.

Arquitectura y parámetros

```
[ ]  vgg = vgg16.VGG16(include_top=False, weights='imagenet', input_shape=(32,32,3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
58889256/58889256 [==============================] - 4s 0us/step
```

```
[ ]  output = vgg.get_layer('block3_pool').output
     new_output_layer = ks.layers.Conv2D(512, (3, 3), strides=1, activation='relu', padding='same')(output)
     vgg_model = Model(vgg.input, new_output_layer)
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 block1_conv1 (Conv2D)       (None, 32, 32, 64)        1792

 block1_conv2 (Conv2D)       (None, 32, 32, 64)        36928

 block1_pool (MaxPooling2D)  (None, 16, 16, 64)        0

 block2_conv1 (Conv2D)       (None, 16, 16, 128)       73856

 block2_conv2 (Conv2D)       (None, 16, 16, 128)       147584

 block2_pool (MaxPooling2D)  (None, 8, 8, 128)         0

 block3_conv1 (Conv2D)       (None, 8, 8, 256)         295168

 block3_conv2 (Conv2D)       (None, 8, 8, 256)         590080

 block3_conv3 (Conv2D)       (None, 8, 8, 256)         590080

 block3_pool (MaxPooling2D)  (None, 4, 4, 256)         0

 conv2d (Conv2D)             (None, 4, 4, 512)         1180160

=================================================================
Total params: 2915648 (11.12 MB)
Trainable params: 2655488 (10.13 MB)
Non-trainable params: 260160 (1016.25 KB)
_________________________________________________________________
```

```
entrenable = False

for layer in vgg_model.layers:
  if layer.name == 'block3_conv1':
    entrenable = True
  layer.trainable = entrenable
```

	Layer Type	Layer Name	Layer Trainable
0	<keras.src.engine.input_layer.InputLayer object at 0x7bcb829ff8b0>	input_1	False
1	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7bcb82a39600>	block1_conv1	False
2	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7bcb82a39d80>	block1_conv2	False
3	<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7bcb82a3ab30>	block1_pool	False
4	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7bcb82a3b370>	block2_conv1	False
5	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7bcb82a3bbe0>	block2_conv2	False
6	<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7bcb8206cbb0>	block2_pool	False
7	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7bcb82a3be50>	block3_conv1	True
8	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7bcb8206d8a0>	block3_conv2	True
9	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7bcb8206e3b0>	block3_conv3	True
10	<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x7bcb8206f460>	block3_pool	True
11	<keras.src.layers.convolutional.conv2d.Conv2D object at 0x7bcc0dcae290>	conv2d	True

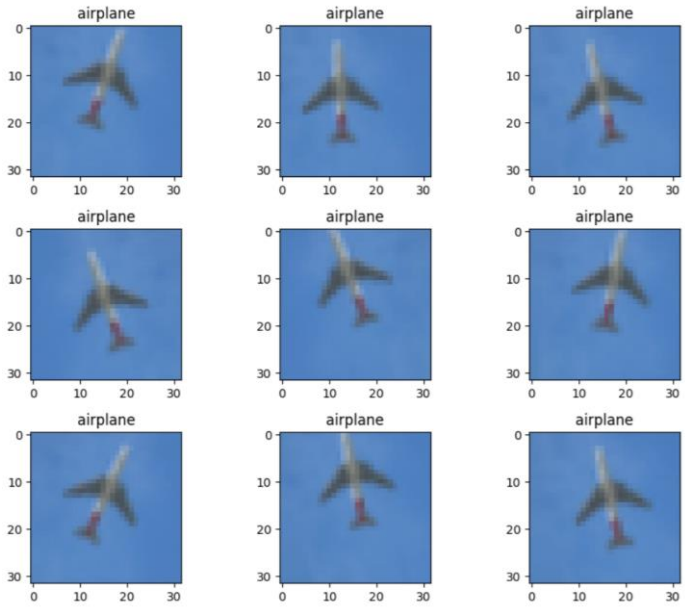
```
batch_size_train = 512
```

```
batch_size_val = 128
```

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    rotation_range=20,
    height_shift_range=0.1,
    width_shift_range=0.1,
    fill_mode='nearest',
    # Definamos las transformaciones
    )

train_generator = train_datagen.flow(
    x_train,
    y_train,
    batch_size=batch_size_train
)
```

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




```
model.compile(optimizer= Adam(learning_rate=0.01),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Entrenamiento

```
# # EarlyStopping
earlystop_val_acc = EarlyStopping(monitor= 'val_accuracy', patience=30)
earlystop_val_loss = EarlyStopping(monitor='val_loss', patience=30)
reduce_lr_callback = ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience=20, min_lr=1e-5, )

# ModelCheckpoint
checkpoint_path = '/content/drive/MyDrive/Master_Data_Science/Entregables/Entregable_04-Deep_Lea
modelcheckpoint_best_acc = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_on
```

```
[ ]  elapsed_time = datetime.timedelta(seconds=(time.perf_counter() - t))

     print('Tiempo de entrenamiento:', elapsed_time)
```

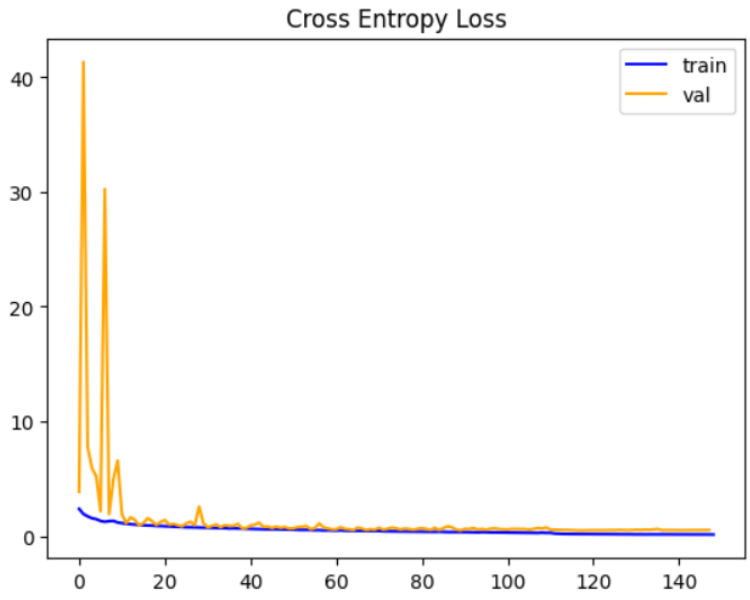
```
Tiempo de entrenamiento: 1:18:50.708430
```

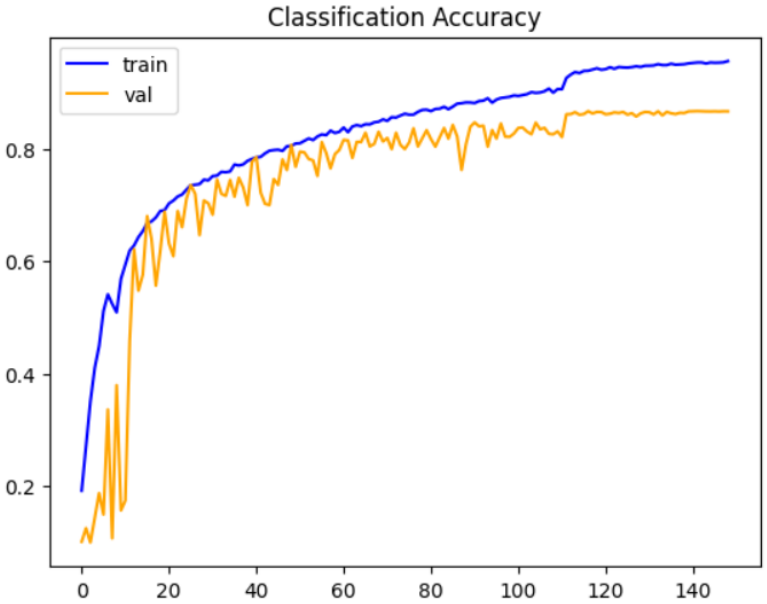
Resultados

Accuracy en el dataset de Test

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

```
> 85.910
```





Observaciones sobre los resultados

El modelo con Transfer Learning y Data augmentation "sólo" logró una precisión del 85,9%. El resultado obtenido está en línea con las expectativas; de hecho, con esta implementación de Transfer Learning, la arquitectura resultante es muy similar a la del experimento 11. Además, los pesos de los dos primeros bloques de convolución están optimizados para la clasificación de las mil clases de imágenes de IMAGENET, mientras que en nuestro caso tenemos un problema 100 veces más pequeño (10 clases). Por tanto, los pesos de la red neuronal entrenada desde cero son más adecuados para nuestro caso específico.

En definitiva este tipo de implementación para nuestro problema específico no aporta ninguna ventaja que justifique su uso.

El alumno es consciente de la existencia de otros métodos más avanzados de aplicación del Transfer Learning que sin duda podrían resultar mucho más útiles en este contexto. Sin embargo, por el momento no se dispone de las habilidades necesarias para implementarlas correctamente.

4 CONCLUSIONES

El proyecto logró todos los objetivos que se habían planteado. Se han realizado 12 experimentos, de los cuales 9 alcanzaron la accuracy mínima requerida del 80% y uno de estos superó el 90%.

A continuación se detallan las conclusiones que se pueden extraer de este proyecto.

- La arquitectura de una red neuronal es con diferencia su característica más importante y es lo primero que se debe optimizar para obtener un buen modelo.
- El learning rate es de fundamental importancia para el correcto entrenamiento de la red neuronal. Un learning rate no optimo puede afectar en gran medida la capacidad de aprendizaje y/o la convergencia del modelo, así como la capacidad de minimizar el error y salir de cualquier mínimo local.
- Elegir valores de patience adecuados en los callbacks es fundamental para optimizar el entrenamiento y obtener los resultados deseados.
- La variación de los porcentajes de dropout puede provocar cambios de comportamiento muy significativos, tanto en términos de accuracy como de duración del entrenamiento.
- La regularización L2 (Weight Decay) en este caso no dio los resultados deseados.

- La aplicación de Data Augmentation en una red demasiado simple empeora el rendimiento en lugar de mejorarlo, ya que la red no es capaz de fitear adecuadamente la gran cantidad de nuevos datos generados.
- La Data Augmentation no conduce a mejoras excepcionales cuando la cantidad y calidad de los datos iniciales ya son buenas.
- Aumentar la capacidad de extracción de features de la red, si se hace sabiamente, mejora la accuracy del modelo, siempre que la parte de clasificación también tenga la profundidad adecuada.
- El Transfer Learning, implementado como se vio en clase, no resultó útil en el caso de CIFAR10.