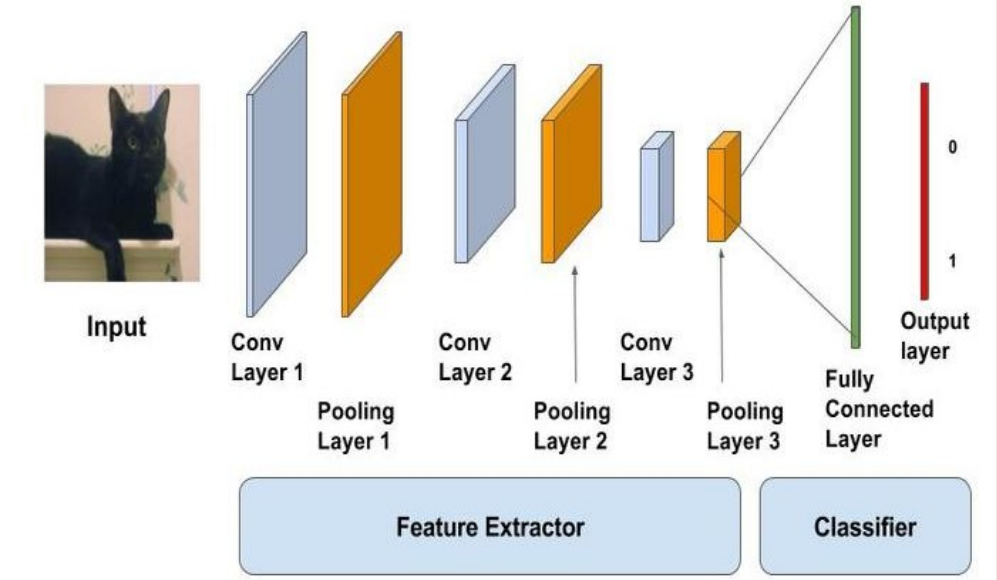
ML_CLASIFICACION_CNN_03	Clasificación de imágenes utilizando CNN en Keras	ML

En esta práctica basada en aprendizaje profundo (Deep Learning) se continúa con el funcionamiento de Redes Neuronales basadas en Convolución (CNN – Convolutional Neural Networks) y cómo utilizarlas para tareas de clasificación de imágenes. También mediante la técnica de "aumento de datos" (data augmentation) se expone como se mejora el rendimiento de la red neuronal.



Esquema de funcionamiento de una CNN

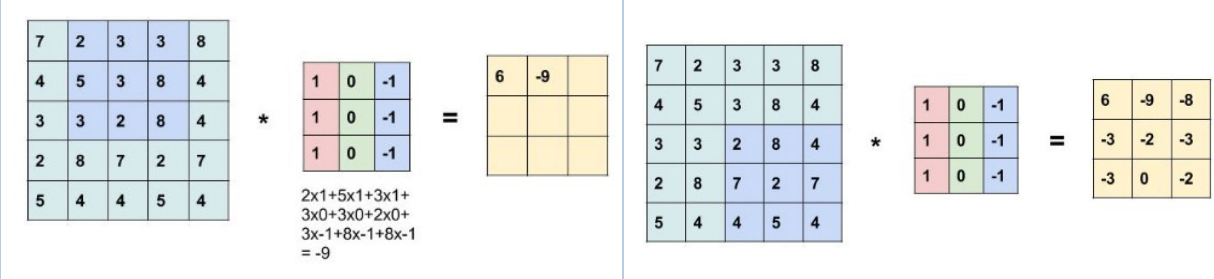
Como dato de entrada tenemos una imagen. Mediante una serie de capas (layers) de convolución y pooling (agrupamiento), se realiza el proceso de extracción de características ('Feature Extractor'). Posteriormante mediante un clasificador ('Classifier') basado en una capa completamente conectada, se averigua como resultado de salida si se trata de un determinado animal o no.

Capa de convolución: Las capas de convolución actúan como los 'ojos' de la red neuronal, buscando determinadas características que producen alta activación de las neuronas. El proceso de convolución se realiza en la imagen completa (n x m) mediante la aplicación de una matriz de ponderación o kernel de tamaño k x k. Esta matriz se aplica en la imagen completa (excepto en los píxeles de borde donde no es posible aplicarla).

Como resultado del proceso de convolución se obtiene una imagen (n-2 x m-2) que se corresponde con el mapa de activación. En esta nueva imagen cada pixel se corresponde con la intensidad de activación de la neurona de la red localizada en la misma posición del pixel.

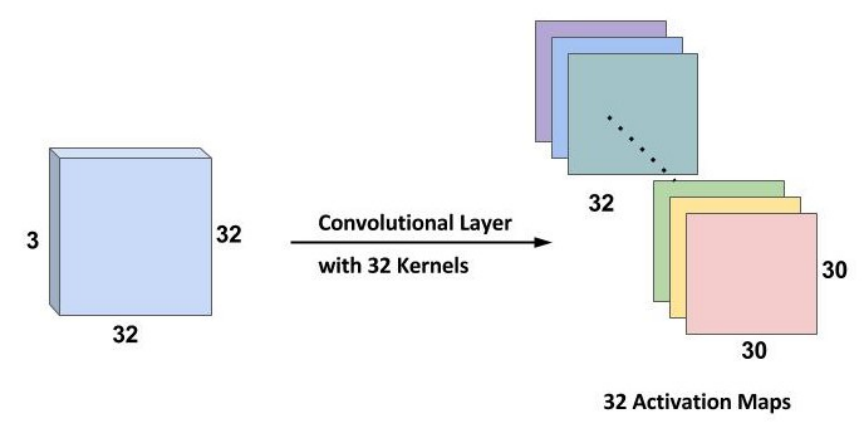
Esta imagen sirve de entrada para la siguiente capa de convolución.

Un ejemplo del proceso de convolución sobre una imagen de 5 x 5 con un kernel de 3 x 3 es el siguiente. Como resultado se obtiene un mapa de activación de (5-2 x 5-2) = (3 x 3)



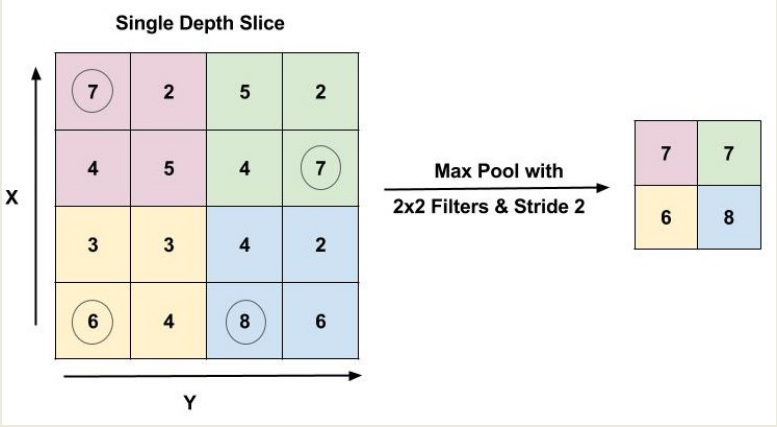
Por razones de conveniencia, se suele añadir a la imagen resultante un borde de ceros '0' (zero-padding) de forma que la imagen tenga las mismas dimensiones que la imagen original de entrada.

Una capa de convolución puede incluir varios 'kernels' dando como resultado varios mapas de activación. Un esquema de una capa de convolución con 32 kernels es el siguiente:



Después de la matriz de convolución se aplica una matriz de agrupamiento ('pooling'). Existen varias formas de agrupamiento, por ejemplo la matriz de máximo (max pooling) que lo que hace es reducir de forma importante el tamaño de la imagen (mapa de activación).

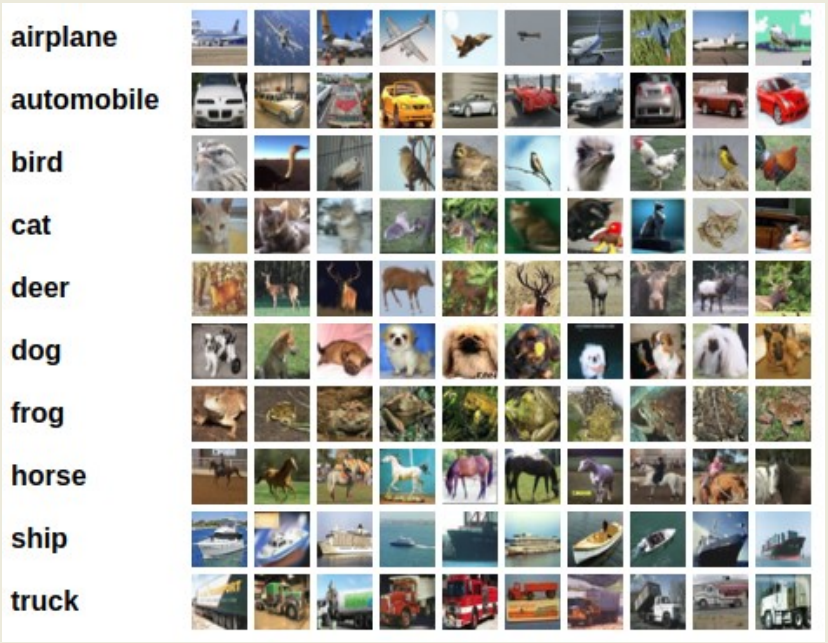
Un ejemplo de cómo actúa la matriz de agrupamiento sobre una matriz resultado del proceso de convolución es el siguiente:



Este proceso se aplica varias veces para la extracción de características, reduciendo el tamaño de las imágenes resultado en cada capa.

Finalmente una capa densa completamente conectada se utiliza para la clasificación.

La base de datos CIFAR10 está integrada en Keras. Contiene un total de 50.000 imágenes para entrenamiento y 10.000 imágenes para test. Dispone de 10 categorías (aviones, automóviles, pájaros, gatos, ciervos, perros, ranas, caballos, barcos y camiones). Las imágenes tienen una resolución de 32 x 32 píxeles. Un ejemplo de imágenes es el siguiente:



Realizamos ahora la programación en Python utilizando la librería Keras disponible para Deep Learning.

SOLUCIÓN

Importar las librerías:

```
import matplotlib.pyplot as plt
import keras
from keras.datasets import cifar10

from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Dropout, Flatten
```

Definir el número de categorías (clases):

```
# Número de categorías (clases)
num_classes=10
```

Cargar la base de datos CIFAR10.

```
# Cargar la base de datos
(X_train,y_train),(X_test,y_test)=cifar10.load_data()
print('X_train: ',X_train.shape)
print(X_train.shape[0],' ejemplos de entrenamiento')
print('X_test: ',X_test.shape)
print(X_test.shape[0],' ejemplos de test')
```

Transformar y normalizar los datos de entrenamiento y de test.

```
# Transformar los datos de entrenamiento y test
X_train=X_train.astype('float32')
X_test=X_test.astype('float32')
X_train/=255
X_test/=255

# Convertir los vectores de clases (etiquetas) en matrices binarias (0,1)
y_train=keras.utils.to_categorical(y_train,num_classes)
y_test=keras.utils.to_categorical(y_test,num_classes)
```

Definir una función para crear el modelo de red neuronal basado en convolución

```
def createModel():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=input_shape))
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
```

```
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(nClasses, activation='softmax'))

    return model
```

El modelo consta de 6 capas de convolución y una capa densa conectada. Se añaden al principio dos capas de convolución con 32 filtros / kernels con un tamaño de 3 x 3, que dan como resultado imágenes de 30 x 30 pixeles. Después se aplica una capa de agrupamiento (max pooling) con un tamaño de 2 x 2, que reduce el tamaño hasta 15 x 15 píxeles.

Posteriormente, se aplican dos nuevas capas de convolución que dan como resultado imágenes de 13 x 13 y seguido una capa de 'max pooling' que reduce el tamaño a 6 x 6.

De nuevo otras dos capas de convolución y agrupamiento lo reducen a 1 pixel (1 valor). Esto se realiza con la capa 'Flatten' que convierte el array 3D a 1D de tamaño 2 x 2 x 64 = 256.

Por último una capa densa permite la clasificación entre las 10 categorias utilizando una función de activación del tipo 'softmax'.

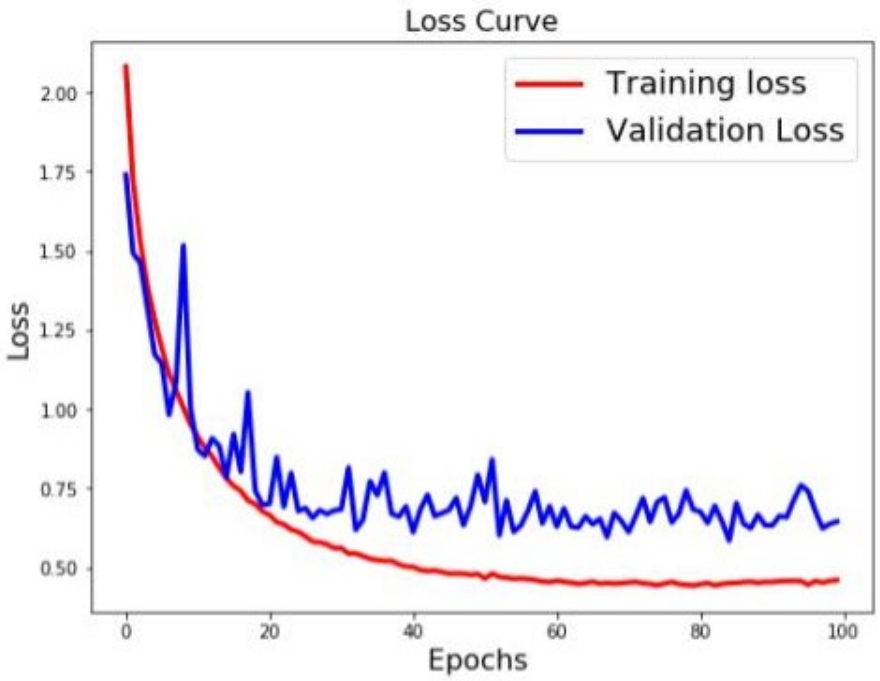
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 6, 6, 64)          36928
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 2, 2, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               131584
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 276,138
Trainable params: 276,138
Non-trainable params: 0
```

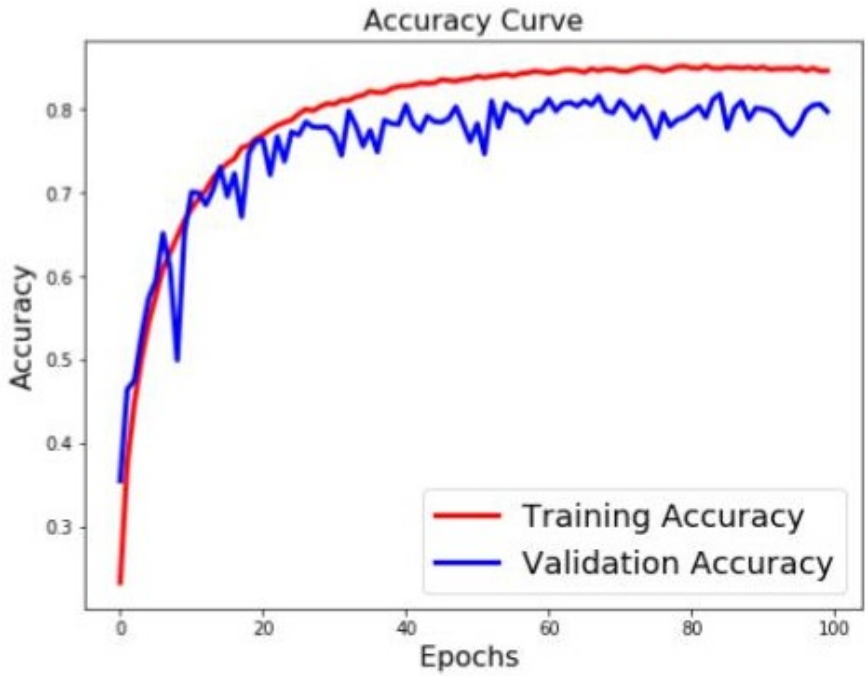
Entrenar la red neuronal utilizando las imágenes de entrenamiento:

```
# Crear el modelo de red neuronal de convolución
model1 = createModel()
batch_size = 256
epochs = 100

# Consultar la arquitectura de la red
model1.summary()

# Definir la pérdida, el optimizador y la métrica:
model1.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])

# Entrenar el modelo
history = model1.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1,
                     validation_data=(X_test, y_test))

# Evaluar para los datos de test
model1.evaluate(X_test, y_test)
```

Representar la pérdida (loss) y la precisión (accuracy):

```
# Visualizar la pérdida y la precisión

# Curvas de pérdida
plt.figure(figsize=[8,6])
plt.plot(history.history['loss'],'r',linewidth=3.0)
plt.plot(history.history['val_loss'],'b',linewidth=3.0)
plt.legend(['Pérdida de Entrenamiento', 'Pérdida de Validación'],fontsize=18)
plt.xlabel('Iteraciones ',fontsize=16)
plt.ylabel('Pérdida',fontsize=16)
plt.title('Curvas de Pérdida',fontsize=16)

# Curvas de precisión
plt.figure(figsize=[8,6])
plt.plot(history.history['acc'],'r',linewidth=3.0)
plt.plot(history.history['val_acc'],'b',linewidth=3.0)
plt.legend(['Precisión de entrenamiento', 'Precisión de Validación'],fontsize=18)
plt.xlabel('Iteraciones ',fontsize=16)
plt.ylabel('Precisión',fontsize=16)
plt.title('Curvas de Precisión',fontsize=16)
```





Se observa una diferencia entre los valores de entrenamiento y de validación (test) lo que muestra un cierto sobreajuste ('overfitting').

Para reducir este sobreajuste utilizamos 'data augmentation'. Además de la 'regularización' es otra técnica utilizada para reducir el sobreajuste.

El proceso de 'aumento de datos' (data augmentation) consiste en **crear de forma artificial nuevas imágenes** a partir de las existentes, cambiando su tamaño, orientación, … Esto es posible con la librería Keras mediante el uso de la instancia 'ImageDataGenerator'.

```
# APLICACIÓN DE LA TÉCNICA DE AUMENTO DE DATOS (DATA AUGMENTATION)

# Importar la librería para generar de forma artificial nuevas imágenes

from keras.preprocessing.image import ImageDataGenerator

ImageDataGenerator(
    rotation_range=10.,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.,
    zoom_range=.1,
    horizontal_flip=True,
    vertical_flip=True)
```

Ejemplo de imágenes generadas de forma artificial

Entrenar ahora la red neuronal pero incluyendo las imágenes artificiales generadas.

```
model2 = createModel()

model2.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])


batch_size = 256
epochs = 100
```

Definir los parámetros para la generación de nuevas imágenes

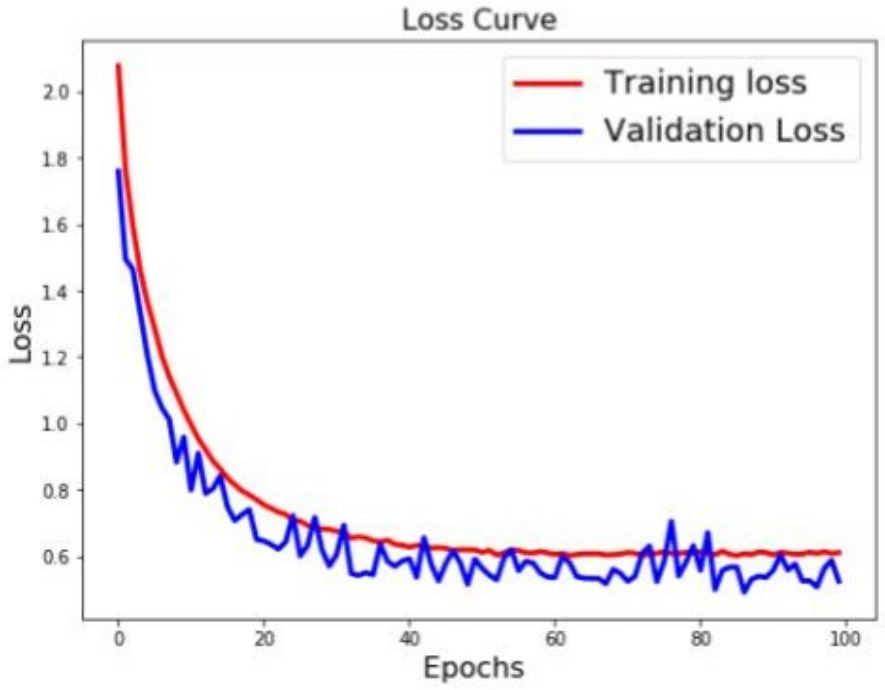
```
# Definir los parámetros para la generación de nuevas imágenes
datagen = ImageDataGenerator(
  zoom_range=0.2, # Zoom aleatorio en las imágenes
  rotation_range=10,  # Rotación aleatoria de las imágenes en el rango (grados, 0 to 180)
  width_shift_range=0.1,  # Modificación aleatoria del ancho (fracción del ancho total)
  height_shift_range=0.1,  # Modificación aleatoria del alto (fracción del alto total)
  horizontal_flip=True,  # Espejo horizontal aleatorio de las imágenes
  vertical_flip=False)  # Espejo vertical aleatorio de las imágenes

# Ajustar el modelo utilizando el generador de imágenes datagen.flow().

history2 = model2.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),
steps_per_epoch=int(np.ceil(train_data.shape[0] / float(batch_size))),
epochs=epochs,validation_data=(X_test, y_test),workers=4)

# Evaluar el nuevo modelo de red neuronal con aumento de datos
model2.evaluate(X_test, y_test)
```

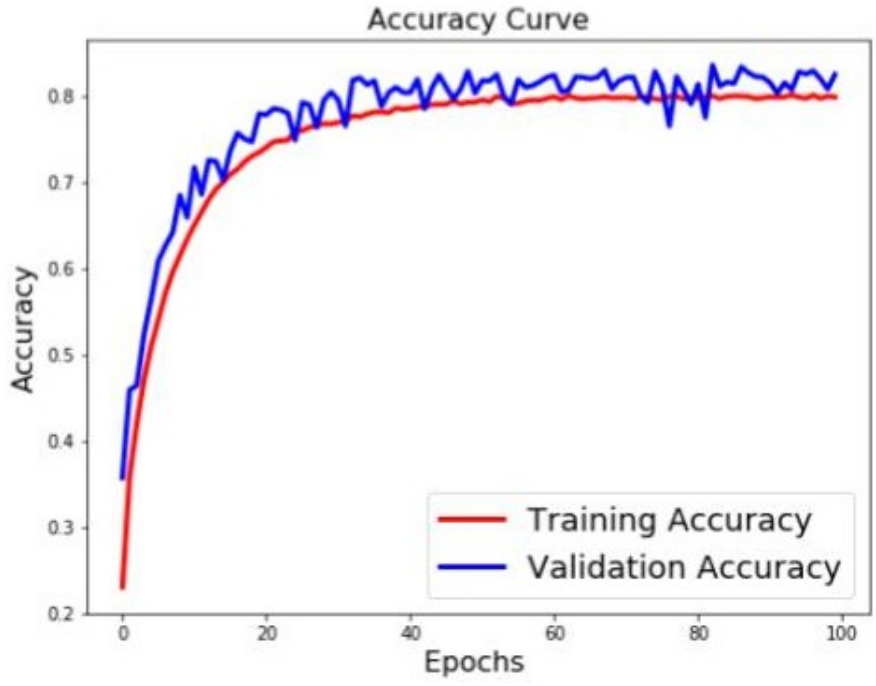
Visualizar los resultados (pérdida y precisión)

```
# Visualizar los resultados
# Evaluar la pérdida y precisión del ajuste

# Curvas de pérdida
plt.figure(figsize=[8,6])
plt.plot(history2.history['loss'],'r',linewidth=3.0)
plt.plot(history2.history['val_loss'],'b',linewidth=3.0)
plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.title('Loss Curves',fontsize=16)

# Curvas de precisión
plt.figure(figsize=[8,6])
plt.plot(history2.history['acc'],'r',linewidth=3.0)
plt.plot(history2.history['val_acc'],'b',linewidth=3.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Accuracy',fontsize=16)
plt.title('Accuracy Curves',fontsize=16)
```

En este caso, al aplicar aumento de datos no existe tanta diferencia entre el proceso de entrenamiento y el de validación lo que implica que no se produje sobreajuste ('overfitting') y el modelo se ha generalizado muy bien.

Esto se debe a que al utilizar peores datos (por ejemplo, imágenes invertidas 'flipped images'), en el proceso de validación y clasificación se registran mejores resultados.


Loss Curve
Training loss
Validation Loss
Loss
Epochs
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0
20
40
60
80
100


Accuracy Curve
Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100