Deep Learning - CNN

CONVUTIONAL NEURAL NETWORK

Fernández Fochs, Carla
NUCLIO DIGITAL SCHOOL | 09 / 21

El objetivo de esta práctica es reducir al máximo la *accuracy* mediante el ajuste de la arquitectura de la red neuronal.

Proyecto_0:

Partimos inicialmente de una red compuesta por **una única Capa Convolucional**.

- En la primera utilizaremos **32 filtros/kernels** de tamaño **3x3** (el kernel más óptimo) con *padding* "same", por lo tanto la imagen no la vamos a reducir de tamaño. Se aplica una función de activación **Relu** y para terminar un **MaxPooling 2x2**.
- Tenemos una Capa Oculta de **32 neuronas** y con una función de activación **Relu**.
- Tenemos una Capa de Salida de **10 neuronas de salida** porque queremos clasificar las siguientes 10 etiquetas: *Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship* y *Truck*. La función de activación es la **softmax** ya que es un problema de *muticlass*.

```
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'))
```

También partiremos de los siguientes hiperparámetros:

- Epochs : **20**
- Tamaño del batch: **512**

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

A continuación, podemos ver el resumen del modelo de partida:

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

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 32)                262176

 dense_1 (Dense)             (None, 10)                330

=================================================================
Total params: 263,402
Trainable params: 263,402
Non-trainable params: 0
_________________________________________________________________
```

Que nos da una *accuracy* del:

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

```
> 62.100
```

Proyecto_1:

Para poder mejorar la *accuracy* empezaremos ajustando algunos parámetros de forma sencilla. Únicamente añadiremos una segunda Capa Convolucional de manera que nuestra red estará compuesta por **2 Capas Convolucionales**.

- 1ª parte tiene la misma configuración que anteriormente.
- 2ª parte añadimos una **segunda Capa Convolucional** con la misma configuración de la primera capa.
- Ajustamos los *epochs* a 100 y configuramos el *early-stopping*.

Justificación: Añadimos una segunda convolución para que el modelo pueda detectar más patrones y así mejorar un poco más la clasificación de las imágenes.

```
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)))

# Añadimos una convulación de 32 filtros
model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(32, activation='relu'))
model.add(ks.layers.Dense(10, activation='softmax'))
```

Hemos reducido los parámetros de 263.402 a 136.954

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 32, 32, 32)        896

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 16, 16, 16)        4624

 flatten_2 (Flatten)         (None, 4096)              0

 dense_4 (Dense)             (None, 32)                131104

 dense_5 (Dense)             (None, 10)                330

=================================================================
Total params: 136,954
Trainable params: 136,954
Non-trainable params: 0
_________________________________________________________________
```

Con sólo esta adición podemos ver que el accuracy nos mejora un **2,66%**:

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

```
> 64.760
```

Proyecto_2:

Seguimos con **2 Capas Convolucionales** pero ahora se realizan los siguientes cambios.

- 1ª parte tiene la misma configuración que anteriormente.
- 2ª parte modificamos la **segunda Capa Convolucional** añadiéndole **100 filtros** de tamaño **3x3**.

Justificación: Si añadimos más filtros el modelo podrá detectar características más complejas de las imágenes.

```
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)))

# Cambiamos por una convulación de 100 filtros
model.add(ks.layers.Conv2D(100, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Flatten())
model.add(ks.layers.Dense(32, activation='relu'))
model.add(ks.layers.Dense(10, activation='softmax'))
```

Se observa una mejora de la *accuracy*:

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

```
> 67.330
```

Proyecto_3:

Ahora añadimos una capa más de manera que ahora tendremos **3 Capas Convolucionales** pero ahora se realizan los siguientes cambios.

- 1ª parte tiene la misma configuración que anteriormente.
- 2ª parte tiene la misma configuración que anteriormente.
- 3ª parte añadimos otra Capa Convolucional con la misma configuración de la capa anterior, de **100 filtros** de tamaño **3x3**.

Justificación: Si añadimos otra capa con un número elevando filtros el modelo podrá detectar cada vez más características y patrones de las imágenes.

```
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)))

# Cambiamos por una convulación de 100 filtros
model.add(ks.layers.Conv2D(100, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(100, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))


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

Se observa una mejora de la *accuracy*:

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

```
> 69.440
```

Proyecto_4:

Seguimos con la misma configuración anterior pero añadimos un ***MaxPooling* de 2x2** después de la tercera red convuncional para reducir la cantidad de datos, pero preservando la información más relevante de esta Capa Convolucional

- 1ª parte tiene la misma configuración que anteriormente.
- 2ª parte tiene la misma configuración que anteriormente.
- 3ª parte añadimos otra Capa Convolucional con la misma configuración de la capa anterior, de **100 filtros** de tamaño **3x3**.

Justificación: Añadimos un ***Max-pooling* de 2x2** para reducir la cantidad de datos y preservar la información más relevante que nos servirá para detección de parámetros relevantes.

```
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)))

# Cambiamos por una convulación de 100 filtros
model.add(ks.layers.Conv2D(100, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(100, (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_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 32, 32, 32)        896

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_7 (Conv2D)           (None, 16, 16, 100)       28900

 conv2d_8 (Conv2D)           (None, 16, 16, 100)       90100

 max_pooling2d_3 (MaxPooling  (None, 8, 8, 100)        0
 2D)

 flatten_2 (Flatten)         (None, 6400)              0

 dense_4 (Dense)             (None, 32)                204832

 dense_5 (Dense)             (None, 10)                330

=================================================================
Total params: 325,058
Trainable params: 325,058
Non-trainable params: 0
_________________________________________________________________
```

Se observa una mejora de la *accuracy*:

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

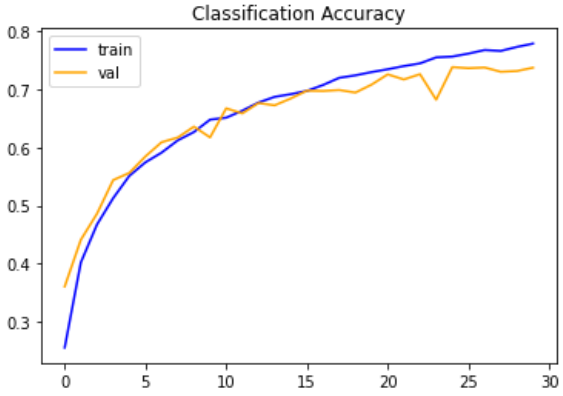
```
> 71.540
```

Proyecto_5:

Conservamos la configuración anterior pero ahora realizamos los siguientes ajustes:

- Añadimos ***drop-outs*** detrás de la primera y segunda convolución para mejorar el overfitting. Primero de todo, sólo desactivamos aleatoriamente un **20%** de conexiones entre neuronas, de esta manera le complicamos un poco el entrenamiento a nuestro modelo para la extracción de las *features*. Lo que se conseguimos con esto es que ninguna neurona memorice parte de la entrada; que es precisamente lo que sucede cuando tenemos overfitting[1]. Lo utilizamos para que las neuronas no se adapten en exceso a otras de su misma capa.
- Además, añadimos otro ***Max-pooling* de 2x2** detrás de la segunda convolución para reducir la cantidad de datos y quedarnos con las características más relevantes (fijémonos que en el modelo anterior no lo teníamos).

```
model = ks.Sequential()

model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

# Cambiamos por una convulación de 100 filtros
model.add(ks.layers.Conv2D(100, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(100, (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'))
```



Observamos una mejora del overfitting, debido a que la curva de la *acurracy* de validación se acerca un poco más a la curva del *accuracy* del train.

Se observa una mejora de la *accuracy*:

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

```
> 72.950
```

[1] https://medium.com/metadatos/t%C3%A9cnicas-de-regularizaci%C3%B3n-b%C3%A1sicas-para-redes-neuronales-b48f396924d4

Proyecto_6

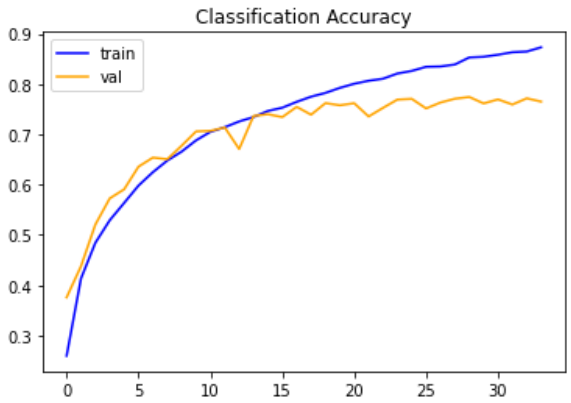
Ahora cambiamos la configuración de segunda y tercera capa convolucional.

- 1ª parte seguimos con la capa de **32 filtros** 3x3, el **MaxPoolling** de **2x2** y los ***drop-outs*** configurados al **0.2** de probabilidad.
- 2ª parte cambiamos la capa de 100 filtros por a una de **64 filtros** (conservando también el l **MaxPooling** de **2x2** y los ***drop-outs*** configurados al **0.2** de probabilidad).
- 3ª parte cambiamos la capa de 100 filtros por a una de **128 filtros** (conservando también el l **MaxPooling** de **2x2** y los ***drop-outs*** configurados al **0.2** de probabilidad).
- 4ª parte añadimos otra **Capa Full-Connected** de **256** neuronas y aumentamos las neuronas de la Capa Full-Connected anterior (que solo tenía 32 neuronas) y ambas con una activación **Relu.** Conjuntamente añadimos ***2 drop-outs*** de 0.2 de probabilidad detrás para ponerle más difícil la clasificación al modelo. Antes no incluíamos los drop-outs en la clasificación, solamente en las Capas Convolucionales.

```
model = ks.Sequential()

# CAPAS CONVOLUCIONALES: EXTRACCIÓN DE CARACTERÍSTICAS

model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))


# Añadimos una convulación otra convulación de 64 filtros
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

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

model.add(ks.layers.Flatten())
# RED NEURONAL : CLASIFICACIÓN
# units = OUTPUT shape
model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.Dropout(0.2))
model.add(ks.layers.Dense(256, activation='relu'))
model.add(ks.layers.Dropout(0.2))
model.add(ks.layers.Dense(10, activation='softmax'))
```

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

```
> 76.030
```



Tenemos más *overfiting,* se tendría que controlar más el overfitting. Podemos añadir un porcentaje más elevado de *drop-outs* para romper conexiones entre neuronas y dificultarle en entrenamiento al modelo.

Proyecto_7:

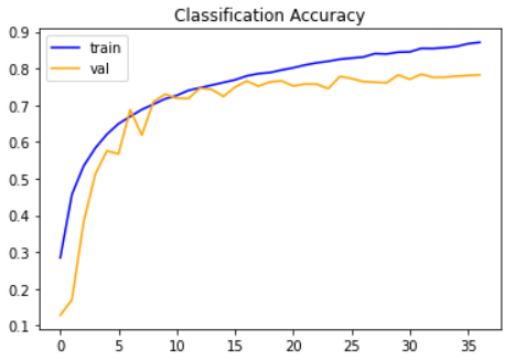
Nos pasamos a Kaggle por los límites del GoogleColab con la GPU.

Seguimos con la configuración anterior de **3 Capas Convolucionales** de **32**, **64** y **128** filtros respectivamente de **tamaño 3x3** cada una de ellas. Después de capa una de estas capas se aplica un **MaxPooling de 2x2.** Los cambios que se realizan pues respecto a la configuración anterior son los siguientes:

- 1º aplicamos el *Batch Normalization*, encargada de normalizar los datos obtenidos tras calcular una capa. Con esto, acabamos teniendo unos valores normalizados que están en [0, 1]. Además conseguimos reducir la distribución de valores, por lo que los valores son más estables. Así que, las capas posteriores pueden más fácil ajustar los pesos.
- Aumentamos el *drop-out al 0.3* de la primera capa, que son las capas dónde se detectan patrones más "sencillos".

```
model = ks.Sequential()

# CAPAS CONVOLUCIONALES: EXTRACCIÓN DE CARACTERÍSTICAS

model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
# Reducción de la dimensión de las características, nos vamos a quedar con las características más importantes que detectó cada filtro
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(BatchNormalization())
model.add(ks.layers.Dropout(0.3))

# Añadimos una convulación otra convulación de 64 filtros
model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(BatchNormalization())
model.add(ks.layers.Dropout(0.3))

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

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

# RED NEURONAL : CLASIFICACIÓN
model.add(ks.layers.Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(ks.layers.Dropout(0.2))
model.add(ks.layers.Dense(256, activation='relu'))

# Normalizamos los datos
model.add(BatchNormalization())
model.add(ks.layers.Dropout(0.2))


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

Vemos que el *overfitting* no es muy alto, por lo tanto no vamos mal encaminados. Tendremos que ajustar algún parámetro más para poder mejorar el modelo. Vemos que la *accuracy* tampoco ya mejorado tanto, pero se observa alguna mejora:




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

```
> 77.430
```

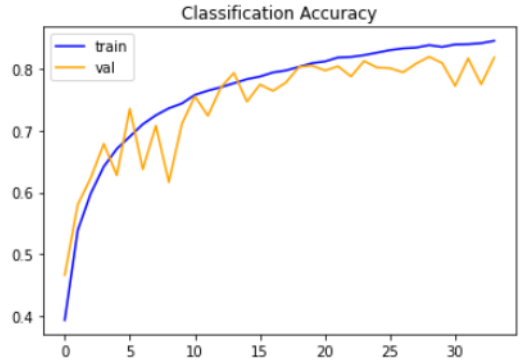
Proyecto_8:

- 1º Colocamos los **Batch Normalization** justo después de las capas de convolución, para normalizar los datos después de haber creado el mapa de características (ya que la teoría nos dice que es mejor ponerlo justo detrás).
- 2º En la tercera **Capa Convolucional** de **128 filtros** de **3x3** añadimos un **Batch Normalization** justo después y al final de todo un **dropout de 0.3**.
- 3º Añadimos otra cuarta **Capa Convolucional con 256 filtros** de **3x3,** junto con el **Batch Normalization** y el **Maxpooling**. Al igual que en todas las capas anteriores, añadimos un *drop-out* de 0.3 al final para desconectar neuronas y así dificultarle un poco más al modelo.
- 4º Vamos a probar de reducir el droupout a 0.1 (en el proyecto anterior la teníamos a 0.2).
- 5º además reducimos el ***batch size*** a **32** para pasarlos por los datos a la red neuronal de forma más lenta y consecutivamente aumentaremos el número de ***epochs*** a **1.250** 40.000/32 = 1.250).

```
model = ks.Sequential()

# CAPAS CONVOLUCIONALES: EXTRACCIÓN DE CARACTERÍSTICAS
model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(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', input_shape=(32,32,3)))
model.add(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', input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

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

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

# RED NEURONAL : CLASIFICACIÓN
model.add(ks.layers.Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(ks.layers.Dropout(0.2))
model.add(ks.layers.Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(ks.layers.Dropout(0.1))

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

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
batch_normalization_18 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
dropout_18 (Dropout)         (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_19 (Batc (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
dropout_19 (Dropout)         (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_20 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
dropout_20 (Dropout)         (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 4, 4, 256)         295168
_________________________________________________________________
batch_normalization_21 (Batc (None, 4, 4, 256)         1024
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 2, 2, 256)         0
_________________________________________________________________
dropout_21 (Dropout)         (None, 2, 2, 256)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               262400
_________________________________________________________________
batch_normalization_22 (Batc (None, 256)               1024
_________________________________________________________________
dropout_22 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_10 (Dense)             (None, 256)               65792
_________________________________________________________________
batch_normalization_23 (Batc (None, 256)               1024
_________________________________________________________________
dropout_23 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                2570
=================================================================
Total params: 723,146
Trainable params: 721,162
Non-trainable params: 1,984
_________________________________________________________________
```

Podemos observar una gran mejora del *accuracy*, pero tal y como se observa en el gráfico al principio tenemos bastante *overfitting,* después a lo largo del tiempo se estabiliza un poco más




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

```
> 80.960
```

Proyecto_9:

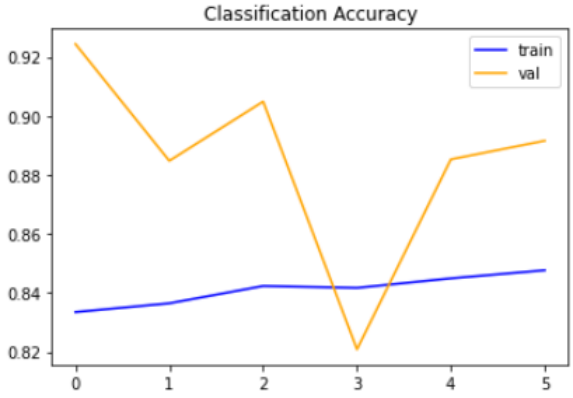
Seguimos con la misma configuración anterior per añadimos algún pequeño cambio:

- Incrementamos el valor del ***drop-out*** a 0.4 (antes estaba a 0.3) de la tercera **Capa Convolucional** de **128 filtros** (kernel 3x3).

```
model = ks.Sequential()

# CAPAS CONVOLUCIONALES: EXTRACCIÓN DE CARACTERÍSTICAS
model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(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', input_shape=(32,32,3)))
model.add(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', input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.4))

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

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

# RED NEURONAL : CLASIFICACIÓN
model.add(ks.layers.Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(ks.layers.Dropout(0.2))
model.add(ks.layers.Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(ks.layers.Dropout(0.1))

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

Justificación: De esta manera intentamos reducir los "picos" de overfitting que nos encontrábamos en la mitad del gráfico de *accuracy*. Por ese mismo motivo se ha seleccionado una capa intermedia para aumentar el *drop-out*.

Observamos que el accuracy nos aumenta respecto al proyecto anterior, sin embargo, el gráfico de la accuracy no parece ser muy adecuado. Así que seguiremos modificando parámetros.




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

```
> 81.680
```

Proyecto_10:

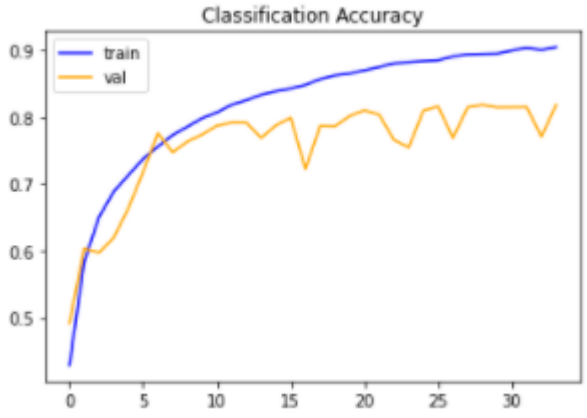
Seguimos con la misma configuración anterior per añadimos algunos cambios:

- Configuramos todas las Capas Convolucionales con un *drop-out* fijado a **0.2**, por lo tanto en este caso, probamos de añadir un poco más de neuronas a nuestro modelo.

Justificación: Anteriormente parece ser que la validación se aleja demasiado del train, lo que nos lleva a pensar que el modelo no está entrenando adecuadamente, puede que le estemos desconectando demasiadas neuronas, lo que impide que nuestro modelo detecte patrones en las capas ocultas, y por lo tanto no llega a un resultado acurado, ni se acerca. Por eso motivo se decide incluir más neuronas disminuyendo los *drop-outs*.

```
model = ks.Sequential()

# CAPAS CONVOLUCIONALES: EXTRACCIÓN DE CARACTERÍSTICAS
model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(64, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(128, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

model.add(ks.layers.Conv2D(256, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.2))

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

# RED NEURONAL : CLASIFICACIÓN
model.add(ks.layers.Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(ks.layers.Dropout(0.2))
model.add(ks.layers.Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(ks.layers.Dropout(0.1))

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

Tal y como se puede observar existe una mejora respecto al gráfico anterior, así pues incluir más neuronas ha mejorado la precisión del modelo. Sin embargo, seguimos teniendo un overfitting, sobretodo por el final.




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

```
> 81.950
```

Concluisones: Finalmente nos quedamos con un modelo con la siguiente arquitectura. Aunque podemos observar un poco de overfiiting, selecionamos el Proyecto_11 debido a que es el que tiene la accuracy más elevada, del **81.95%.**

Mejoras

Proyecto_11:

Procedemos a realizar un Data Aumentation para poderle facilitar al modelo de las imágenes con diferentes angulos de rotación, con inclinaciones diveras y con ampliaciones (zoom) de las imágenes, de manera aleatoria.

```
tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=True,
    samplewise_center=True,
    featurewise_std_normalization=True,
    samplewise_std_normalization=True,
    zca_whitening=True,
    zca_epsilon=1e-06,
    rotation_range=50,
    width_shift_range=0.2,
    height_shift_range=0.2,
    brightness_range=None,
    shear_range=0.2,
    zoom_range=0.2,
    channel_shift_range=0.2,
    fill_mode="nearest",
    cval=0.0,
    horizontal_flip=True,
    vertical_flip=True,
    rescale=None,
    preprocessing_function=None,
    data_format=None,
    validation_split=0.2,
    dtype=None,
)
```

- Todos los valores booleanos los asignamos a "True".
- **zca_epsilon:** le asigamos el valor por defecto.
- **rotation_range:** es el ángulo que vamos a rotar la imagen (0-180). Le asignaremos un valor 'complicado', por lo que empezaremos con un ángulo de **50º**.
- **width_shift** y **height_shift** los asignamos a 0.2 (son valores possibles entre [-1, 0 , +1]).
- **shear_range**: para aplicar aleatoriamente transformaciones de cizallamiento-
- **zoom_range**: para hacer zoom aleatoriamente dentro de las imágenes
- **horizontal_flip/vertical_flip**: es voltear aleatoriamente la mitad de las imágenes horizontalmente o verticalmente. Los los asignamos a "True" para que realice los flips.
- **fill_mode**: para rellenar píxeles recién creados, que pueden aparecer después de una rotación o un cambio de ancho / alto. Le asignamos el valor por defecto "nearest".

Además de realizar el ImageDataGenerator también modificaremos la arquitectura del modelo:

- 1º Añadimos otra capa convolucional de 32 filtros detrás de la primera capa de 32 filtros. Cambiamos el drop-put a 0.3 debido a que hemos aumentado el número de parámetros y consideremamos una buena opción desconectar algunas neuronas para ponerle más dificil al modelo.
- 2º Añadimos otra capa convolucional de 64 filtros detrás de la primera capa de 64 filtros. Igual que anteriormente, cambiamos el drop-put a 0.3 debido a que hemos aumentado el número de

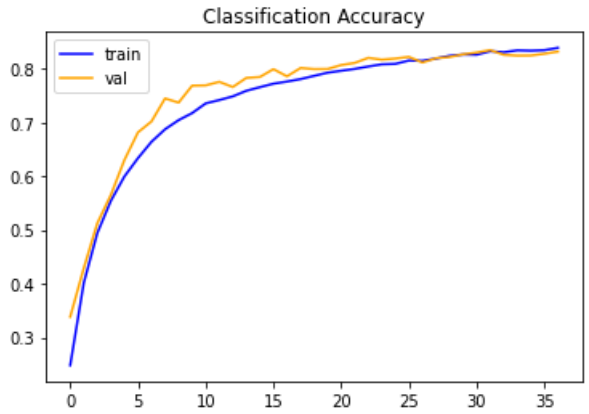
parámetros y consideremamos una buena opción desconectar algunas neuronas para ponerle más dificil al modelo.

- Cambiamos el drop-out a 0.3 para reducir los parámetros del modelo con un número resultante de 719.402.

```
model = ks.Sequential()

# CAPAS CONVOLUCIONALES: EXTRACCIÓN DE CARACTERÍSTICAS
model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.Conv2D(32, (3, 3), strides=1, activation='relu',
                           padding='same', input_shape=(32,32,3)))
model.add(ks.layers.MaxPooling2D((2, 2)))
model.add(ks.layers.Dropout(0.3))

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

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

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

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

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

# RED NEURONAL : CLASIFICACIÓN
model.add(ks.layers.Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(ks.layers.Dropout(0.2))
model.add(ks.layers.Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(ks.layers.Dropout(0.1))

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

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_19 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
dropout_21 (Dropout)         (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
dropout_22 (Dropout)         (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
dropout_23 (Dropout)         (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 4, 4, 128)         147584
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 2, 2, 128)         0
_________________________________________________________________
dropout_24 (Dropout)         (None, 2, 2, 128)         0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 2, 2, 256)         295168
_________________________________________________________________
batch_normalization_9 (Batch (None, 2, 2, 256)         1024
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 1, 1, 256)         0
_________________________________________________________________
dropout_25 (Dropout)         (None, 1, 1, 256)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_9 (Dense)              (None, 256)               65792
_________________________________________________________________
batch_normalization_10 (Batc (None, 256)               1024
_________________________________________________________________
dropout_26 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_10 (Dense)             (None, 256)               65792
_________________________________________________________________
batch_normalization_11 (Batc (None, 256)               1024
_________________________________________________________________
dropout_27 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                2570
=================================================================
Total params: 719,402
Trainable params: 717,866
Non-trainable params: 1,536
_________________________________________________________________
```

Podemos ver que el gráfico de la accuracy nos da un resultado muy bueno, debido a que la curva de validación se ajusta muy bien a la curva del train.



Se puede observar una mejora de la accuracy del 1,11%:

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

```
> 82.860
```