Intento 6

En este intento volvemos al batch_size de 32 y añadimos una tercera capa de convolución de 64 filtros (junto con otra de Maxpooling 2x2). También probamos a reducir el índice de olvido al 20% para poder evaluar mejor la aportación de esta nueva capa al modelo.

0. Descarga del dataset

In [1]:

```
# from google.colab import drive
# drive.mount('/content/drive')

train_ds_path  ='../../deeplearning-az/datasets/Part 2 - Convolutional Neu
test_ds_path   ='../../deeplearning-az/datasets/Part 2 - Convolutional Neu
cat_or_dog_path='../../deeplearning-az/datasets/Part 2 - Convolutional Neu

#train_ds_path  ='C:/Users/Usuario/Documents/Master/Aprendizaje Profundo/U
#test_ds_path   ='C:/Users/Usuario/Documents/Master/Aprendizaje Profundo/U
#cat_or_dog_path='C:/Users/Usuario/Documents/Master/Aprendizaje Profundo/U

#train_ds_path  ='.\\data\\training_set'
#test_ds_path   ='.\\data\\test_set'
#cat_or_dog_path='.\\data\\single_prediction\\cat_or_dog_1.jpg'
```

Primero, importar las librerías y paquetes

In [2]:

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import BatchNormalization
from keras.layers import GlobalAveragePooling2D
from keras.layers import Dropout
# Nota, algunas capas no están importadas aquí y se importan directamente 

import matplotlib.pyplot as plt
import tensorflow as tf
import os
import numpy as np
import random as rn
```

Fijamos seeds para poder reproducir resultados (aunque aun así a veces no lo conseguimos, probablementa haya inicializaciones que no dependan de estas seeds)

In [3]:

```
os.environ['PYTHONHASHSEED'] = '0'
np.random.seed(42)
rn.seed(12345)
tf.random.set_seed(1234)
```

1. Construcción del modelo CNN añadiendo un tamaño de imagen mayor

El tamaño de imagen que emplearemos será de 96x96, y el dropout rate es del 50%

In [4]:

```
frame_size = (96, 96)
"""
esta dupla nos permitirá parametrizar la resolución
de entrada de las imágenes
"""

def crear_clasificador_intento6():
    classifier = Sequential()
    classifier.add(Conv2D(filters = 32,kernel_size = (3, 3),
                     input_shape = (*frame_size, 3), activation = "relu")
    classifier.add(MaxPooling2D(pool_size = (2,2)))
    classifier.add(Conv2D(filters = 32,kernel_size = (3, 3), activation = '
    classifier.add(MaxPooling2D(pool_size = (2,2)))
    classifier.add(Conv2D(filters = 64,kernel_size = (3, 3), activation = '
    classifier.add(MaxPooling2D(pool_size = (2,2)))
    classifier.add(Flatten())
    classifier.add(Dense(units = 128, activation = "relu"))
    classifier.add(Dropout(0.2))
    classifier.add(Dense(units = 1, activation = "sigmoid"))
    return classifier
```

2. Entrenamiento del intento 6

En primer lugar instanciamos nuestro modelo y compilamos usando:

- Un optimizador Adam. La learning rate que emplea por defecto es 0.001
- Binary cross entropy como función de coste a minimizar.

In [5]:

```
classifier = crear_clasificador_intento6()
classifier.compile(optimizer = "adam",
                   loss = "binary_crossentropy",
                   metrics = ["accuracy"])
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 94, 94, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 47, 47, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 45, 45, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 22, 22, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 20, 20, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 6400)              0
_________________________________________________________________
dense (Dense)                (None, 128)               819328
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
```

```
Total params: 848,097
Trainable params: 848,097
Non-trainable params: 0
```

En segundo lugar, generamos los datasets de entrenamiento y test. Emplearemos un **tamaño de batch de 32**

In [6]:

```
from keras.preprocessing.image import ImageDataGenerator

batch_size=32

train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

training_dataset = train_datagen.flow_from_directory(train_ds_path,
                                                     target_size=frame_siz
                                                     batch_size=batch_size
                                                     class_mode='binary')

testing_dataset = test_datagen.flow_from_directory(test_ds_path,
                                                   target_size=frame_size,
                                                   batch_size=batch_size,
                                                   class_mode='binary')
```

```
Found 8000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```

Definimos el callback y realizamos el entrenamiento con las condiciones descritas en la sección de introducción.

In [7]:

```
callbacks = [
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta = 0, pa
]

history1 = classifier.fit(x=training_dataset,
                         steps_per_epoch=8000/batch_size,
                         epochs=100,
                         validation_data=testing_dataset,
                         validation_steps=2000/batch_size,
                         workers=4) # "Si pedimos más de un proceso el ren
```

```
Epoch 1/100
250/250 [==============================] - 64s 257ms/step - loss: 0.6726 -
accuracy: 0.5761 - val_loss: 0.6280 - val_accuracy: 0.6505
Epoch 2/100
250/250 [==============================] - 64s 258ms/step - loss: 0.6050 -
accuracy: 0.6625 - val_loss: 0.5728 - val_accuracy: 0.7005
Epoch 3/100
250/250 [==============================] - 65s 259ms/step - loss: 0.5668 -
accuracy: 0.7040 - val_loss: 0.5317 - val_accuracy: 0.7295
Epoch 4/100
250/250 [==============================] - 64s 256ms/step - loss: 0.5340 -
accuracy: 0.7343 - val_loss: 0.5045 - val_accuracy: 0.7520
Epoch 5/100
250/250 [==============================] - 67s 268ms/step - loss: 0.5000 -
accuracy: 0.7555 - val_loss: 0.5228 - val_accuracy: 0.7430
```

```
Epoch 6/100
250/250 [==============================] - 63s 253ms/step - loss: 0.4728 -
accuracy: 0.7690 - val_loss: 0.4564 - val_accuracy: 0.7880
Epoch 7/100
250/250 [==============================] - 61s 244ms/step - loss: 0.4607 -
accuracy: 0.7843 - val_loss: 0.4446 - val_accuracy: 0.7940
Epoch 8/100
250/250 [==============================] - 61s 245ms/step - loss: 0.4375 -
accuracy: 0.7983 - val_loss: 0.4250 - val_accuracy: 0.7970
Epoch 9/100
250/250 [==============================] - 61s 245ms/step - loss: 0.4145 -
accuracy: 0.8090 - val_loss: 0.4376 - val_accuracy: 0.7995
Epoch 10/100
250/250 [==============================] - 61s 246ms/step - loss: 0.4024 -
accuracy: 0.8139 - val_loss: 0.4393 - val_accuracy: 0.7955
Epoch 11/100
250/250 [==============================] - 62s 248ms/step - loss: 0.3884 -
accuracy: 0.8231 - val_loss: 0.4128 - val_accuracy: 0.8160
Epoch 12/100
250/250 [==============================] - 59s 237ms/step - loss: 0.3870 -
accuracy: 0.8267 - val_loss: 0.4154 - val_accuracy: 0.8145
Epoch 13/100
250/250 [==============================] - 62s 247ms/step - loss: 0.3690 -
accuracy: 0.8353 - val_loss: 0.3837 - val_accuracy: 0.8400
Epoch 14/100
250/250 [==============================] - 59s 234ms/step - loss: 0.3609 -
accuracy: 0.8356 - val_loss: 0.4149 - val_accuracy: 0.8250
Epoch 15/100
250/250 [==============================] - 59s 237ms/step - loss: 0.3565 -
accuracy: 0.8397 - val_loss: 0.3835 - val_accuracy: 0.8280
Epoch 16/100
250/250 [==============================] - 59s 236ms/step - loss: 0.3358 -
accuracy: 0.8506 - val_loss: 0.3798 - val_accuracy: 0.8485
Epoch 17/100
250/250 [==============================] - 59s 236ms/step - loss: 0.3426 -
accuracy: 0.8489 - val_loss: 0.3912 - val_accuracy: 0.8290
Epoch 18/100
250/250 [==============================] - 59s 237ms/step - loss: 0.3145 -
accuracy: 0.8602 - val_loss: 0.4399 - val_accuracy: 0.8220
Epoch 19/100
250/250 [==============================] - 59s 236ms/step - loss: 0.3087 -
accuracy: 0.8680 - val_loss: 0.3955 - val_accuracy: 0.8365
Epoch 20/100
250/250 [==============================] - 59s 236ms/step - loss: 0.2976 -
accuracy: 0.8716 - val_loss: 0.4297 - val_accuracy: 0.8375
Epoch 21/100
250/250 [==============================] - 59s 236ms/step - loss: 0.2965 -
accuracy: 0.8706 - val_loss: 0.3784 - val_accuracy: 0.8370
Epoch 22/100
250/250 [==============================] - 59s 237ms/step - loss: 0.2806 -
accuracy: 0.8798 - val_loss: 0.4103 - val_accuracy: 0.8395
Epoch 23/100
250/250 [==============================] - 59s 236ms/step - loss: 0.2738 -
accuracy: 0.8821 - val_loss: 0.3818 - val_accuracy: 0.8495
Epoch 24/100
250/250 [==============================] - 66s 264ms/step - loss: 0.2850 -
accuracy: 0.8727 - val_loss: 0.3671 - val_accuracy: 0.8525
Epoch 25/100
250/250 [==============================] - 66s 262ms/step - loss: 0.2749 -
accuracy: 0.8855 - val_loss: 0.4001 - val_accuracy: 0.8485
Epoch 26/100
250/250 [==============================] - 64s 255ms/step - loss: 0.2626 -
accuracy: 0.8884 - val_loss: 0.4052 - val_accuracy: 0.8395
Epoch 27/100
```

```
250/250 [==============================] - 64s 257ms/step - loss: 0.2379 -
accuracy: 0.8994 - val_loss: 0.4071 - val_accuracy: 0.8440
Epoch 28/100
250/250 [==============================] - 65s 258ms/step - loss: 0.2402 -
accuracy: 0.9007 - val_loss: 0.3818 - val_accuracy: 0.8530
Epoch 29/100
250/250 [==============================] - 67s 269ms/step - loss: 0.2338 -
accuracy: 0.8976 - val_loss: 0.4517 - val_accuracy: 0.8505
Epoch 30/100
250/250 [==============================] - 66s 264ms/step - loss: 0.2381 -
accuracy: 0.8992 - val_loss: 0.4283 - val_accuracy: 0.8435
Epoch 31/100
250/250 [==============================] - 60s 238ms/step - loss: 0.2375 -
accuracy: 0.9004 - val_loss: 0.3870 - val_accuracy: 0.8475
Epoch 32/100
250/250 [==============================] - 60s 238ms/step - loss: 0.2179 -
accuracy: 0.9078 - val_loss: 0.4980 - val_accuracy: 0.8360
Epoch 33/100
250/250 [==============================] - 59s 236ms/step - loss: 0.2247 -
accuracy: 0.9061 - val_loss: 0.4475 - val_accuracy: 0.8440
Epoch 34/100
250/250 [==============================] - 59s 235ms/step - loss: 0.2084 -
accuracy: 0.9151 - val_loss: 0.4367 - val_accuracy: 0.8530
Epoch 35/100
250/250 [==============================] - 59s 236ms/step - loss: 0.1989 -
accuracy: 0.9186 - val_loss: 0.4722 - val_accuracy: 0.8380
Epoch 36/100
250/250 [==============================] - 59s 234ms/step - loss: 0.1930 -
accuracy: 0.9178 - val_loss: 0.4657 - val_accuracy: 0.8495
Epoch 37/100
250/250 [==============================] - 58s 233ms/step - loss: 0.1915 -
accuracy: 0.9194 - val_loss: 0.4857 - val_accuracy: 0.8500
Epoch 38/100
250/250 [==============================] - 59s 235ms/step - loss: 0.1960 -
accuracy: 0.9202 - val_loss: 0.4880 - val_accuracy: 0.8325
Epoch 39/100
250/250 [==============================] - 59s 237ms/step - loss: 0.1823 -
accuracy: 0.9265 - val_loss: 0.4757 - val_accuracy: 0.8545
Epoch 40/100
250/250 [==============================] - 60s 238ms/step - loss: 0.1893 -
accuracy: 0.9235 - val_loss: 0.4632 - val_accuracy: 0.8435
Epoch 41/100
250/250 [==============================] - 59s 237ms/step - loss: 0.1824 -
accuracy: 0.9270 - val_loss: 0.4595 - val_accuracy: 0.8390
Epoch 42/100
250/250 [==============================] - 58s 234ms/step - loss: 0.1756 -
accuracy: 0.9261 - val_loss: 0.5226 - val_accuracy: 0.8505
Epoch 43/100
250/250 [==============================] - 62s 249ms/step - loss: 0.1782 -
accuracy: 0.9293 - val_loss: 0.5695 - val_accuracy: 0.8255
Epoch 44/100
250/250 [==============================] - 66s 265ms/step - loss: 0.1670 -
accuracy: 0.9324 - val_loss: 0.5274 - val_accuracy: 0.8505
Epoch 45/100
250/250 [==============================] - 65s 258ms/step - loss: 0.1611 -
accuracy: 0.9345 - val_loss: 0.4886 - val_accuracy: 0.8420
Epoch 46/100
250/250 [==============================] - 63s 253ms/step - loss: 0.1521 -
accuracy: 0.9392 - val_loss: 0.5189 - val_accuracy: 0.8485
Epoch 47/100
250/250 [==============================] - 64s 257ms/step - loss: 0.1504 -
accuracy: 0.9396 - val_loss: 0.4718 - val_accuracy: 0.8540
Epoch 48/100
250/250 [==============================] - 64s 257ms/step - loss: 0.1504 -
```

```
accuracy: 0.9415 - val_loss: 0.5719 - val_accuracy: 0.8245
Epoch 49/100
250/250 [==============================] - 66s 263ms/step - loss: 0.1517 -
accuracy: 0.9341 - val_loss: 0.5151 - val_accuracy: 0.8525
Epoch 50/100
250/250 [==============================] - 59s 236ms/step - loss: 0.1418 -
accuracy: 0.9436 - val_loss: 0.4707 - val_accuracy: 0.8520
Epoch 51/100
250/250 [==============================] - 58s 232ms/step - loss: 0.1426 -
accuracy: 0.9416 - val_loss: 0.4788 - val_accuracy: 0.8500
Epoch 52/100
250/250 [==============================] - 59s 235ms/step - loss: 0.1475 -
accuracy: 0.9435 - val_loss: 0.5568 - val_accuracy: 0.8475
Epoch 53/100
250/250 [==============================] - 58s 233ms/step - loss: 0.1481 -
accuracy: 0.9419 - val_loss: 0.5339 - val_accuracy: 0.8435
Epoch 54/100
250/250 [==============================] - 59s 234ms/step - loss: 0.1330 -
accuracy: 0.9439 - val_loss: 0.5265 - val_accuracy: 0.8515
Epoch 55/100
250/250 [==============================] - 58s 234ms/step - loss: 0.1432 -
accuracy: 0.9450 - val_loss: 0.5515 - val_accuracy: 0.8490
Epoch 56/100
250/250 [==============================] - 59s 236ms/step - loss: 0.1310 -
accuracy: 0.9491 - val_loss: 0.5198 - val_accuracy: 0.8515
Epoch 57/100
250/250 [==============================] - 60s 238ms/step - loss: 0.1255 -
accuracy: 0.9499 - val_loss: 0.6195 - val_accuracy: 0.8260
Epoch 58/100
250/250 [==============================] - 58s 233ms/step - loss: 0.1248 -
accuracy: 0.9520 - val_loss: 0.6331 - val_accuracy: 0.8405
Epoch 59/100
250/250 [==============================] - 59s 236ms/step - loss: 0.1308 -
accuracy: 0.9471 - val_loss: 0.5211 - val_accuracy: 0.8525
Epoch 60/100
250/250 [==============================] - 61s 243ms/step - loss: 0.1333 -
accuracy: 0.9473 - val_loss: 0.5735 - val_accuracy: 0.8425
Epoch 61/100
250/250 [==============================] - 59s 235ms/step - loss: 0.1357 -
accuracy: 0.9450 - val_loss: 0.7643 - val_accuracy: 0.8125
Epoch 62/100
250/250 [==============================] - 59s 236ms/step - loss: 0.1362 -
accuracy: 0.9461 - val_loss: 0.5254 - val_accuracy: 0.8395
Epoch 63/100
250/250 [==============================] - 58s 234ms/step - loss: 0.1142 -
accuracy: 0.9565 - val_loss: 0.5917 - val_accuracy: 0.8385
Epoch 64/100
250/250 [==============================] - 59s 235ms/step - loss: 0.1157 -
accuracy: 0.9551 - val_loss: 0.6431 - val_accuracy: 0.8510
Epoch 65/100
250/250 [==============================] - 59s 235ms/step - loss: 0.1279 -
accuracy: 0.9480 - val_loss: 0.5674 - val_accuracy: 0.8500
Epoch 66/100
250/250 [==============================] - 59s 236ms/step - loss: 0.1116 -
accuracy: 0.9550 - val_loss: 0.5678 - val_accuracy: 0.8450
Epoch 67/100
250/250 [==============================] - 59s 235ms/step - loss: 0.1267 -
accuracy: 0.9491 - val_loss: 0.6206 - val_accuracy: 0.8580
Epoch 68/100
250/250 [==============================] - 58s 232ms/step - loss: 0.1144 -
accuracy: 0.9540 - val_loss: 0.5905 - val_accuracy: 0.8615
Epoch 69/100
250/250 [==============================] - 59s 236ms/step - loss: 0.1096 -
accuracy: 0.9576 - val_loss: 0.6223 - val_accuracy: 0.8590
```

```
Epoch 70/100
250/250 [==============================] - 60s 238ms/step - loss: 0.1210 -
accuracy: 0.9523 - val_loss: 0.6188 - val_accuracy: 0.8510
Epoch 71/100
250/250 [==============================] - 59s 235ms/step - loss: 0.1178 -
accuracy: 0.9531 - val_loss: 0.5838 - val_accuracy: 0.8450
Epoch 72/100
250/250 [==============================] - 59s 236ms/step - loss: 0.1053 -
accuracy: 0.9588 - val_loss: 0.7065 - val_accuracy: 0.8585
Epoch 73/100
250/250 [==============================] - 59s 234ms/step - loss: 0.1044 -
accuracy: 0.9580 - val_loss: 0.7754 - val_accuracy: 0.8545
Epoch 74/100
250/250 [==============================] - 59s 238ms/step - loss: 0.1078 -
accuracy: 0.9561 - val_loss: 0.6626 - val_accuracy: 0.8455
Epoch 75/100
250/250 [==============================] - 59s 235ms/step - loss: 0.0988 -
accuracy: 0.9604 - val_loss: 0.6386 - val_accuracy: 0.8570
Epoch 76/100
250/250 [==============================] - 61s 242ms/step - loss: 0.1037 -
accuracy: 0.9591 - val_loss: 0.6246 - val_accuracy: 0.8565
Epoch 77/100
250/250 [==============================] - 59s 235ms/step - loss: 0.1116 -
accuracy: 0.9565 - val_loss: 0.5792 - val_accuracy: 0.8620
Epoch 78/100
250/250 [==============================] - 66s 262ms/step - loss: 0.1072 -
accuracy: 0.9585 - val_loss: 0.6018 - val_accuracy: 0.8610
Epoch 79/100
250/250 [==============================] - 64s 257ms/step - loss: 0.0979 -
accuracy: 0.9609 - val_loss: 0.6525 - val_accuracy: 0.8490
Epoch 80/100
250/250 [==============================] - 64s 258ms/step - loss: 0.0979 -
accuracy: 0.9621 - val_loss: 0.6262 - val_accuracy: 0.8580
Epoch 81/100
250/250 [==============================] - 62s 249ms/step - loss: 0.1030 -
accuracy: 0.9616 - val_loss: 0.5836 - val_accuracy: 0.8505
Epoch 82/100
250/250 [==============================] - 62s 247ms/step - loss: 0.0958 -
accuracy: 0.9628 - val_loss: 0.6047 - val_accuracy: 0.8545
Epoch 83/100
250/250 [==============================] - 61s 245ms/step - loss: 0.0969 -
accuracy: 0.9628 - val_loss: 0.7115 - val_accuracy: 0.8325
Epoch 84/100
250/250 [==============================] - 61s 245ms/step - loss: 0.1136 -
accuracy: 0.9575 - val_loss: 0.5949 - val_accuracy: 0.8520
Epoch 85/100
250/250 [==============================] - 62s 248ms/step - loss: 0.0949 -
accuracy: 0.9632 - val_loss: 0.7848 - val_accuracy: 0.8175
Epoch 86/100
250/250 [==============================] - 63s 253ms/step - loss: 0.1024 -
accuracy: 0.9620 - val_loss: 0.5941 - val_accuracy: 0.8510
Epoch 87/100
250/250 [==============================] - 59s 238ms/step - loss: 0.1054 -
accuracy: 0.9617 - val_loss: 0.6411 - val_accuracy: 0.8535
Epoch 88/100
250/250 [==============================] - 59s 237ms/step - loss: 0.0851 -
accuracy: 0.9678 - val_loss: 0.6140 - val_accuracy: 0.8570
Epoch 89/100
250/250 [==============================] - 59s 236ms/step - loss: 0.0902 -
accuracy: 0.9647 - val_loss: 0.5954 - val_accuracy: 0.8585
Epoch 90/100
250/250 [==============================] - 58s 233ms/step - loss: 0.0922 -
accuracy: 0.9639 - val_loss: 0.5895 - val_accuracy: 0.8625
Epoch 91/100
```

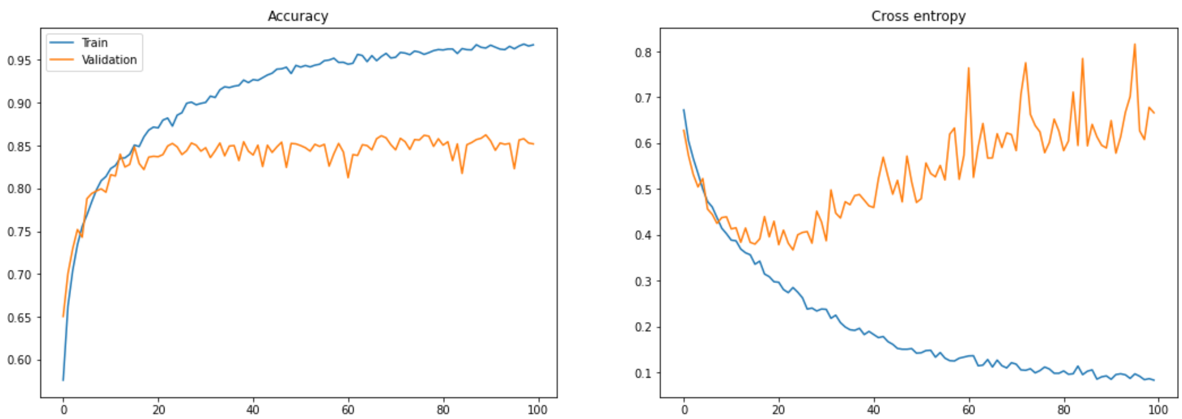
```
250/250 [==============================] - 59s 235ms/step - loss: 0.0848 -
accuracy: 0.9672 - val_loss: 0.6492 - val_accuracy: 0.8555
Epoch 92/100
250/250 [==============================] - 61s 242ms/step - loss: 0.0951 -
accuracy: 0.9649 - val_loss: 0.5782 - val_accuracy: 0.8445
Epoch 93/100
250/250 [==============================] - 59s 235ms/step - loss: 0.0968 -
accuracy: 0.9626 - val_loss: 0.6141 - val_accuracy: 0.8530
Epoch 94/100
250/250 [==============================] - 58s 234ms/step - loss: 0.0945 -
accuracy: 0.9620 - val_loss: 0.6674 - val_accuracy: 0.8515
Epoch 95/100
250/250 [==============================] - 59s 235ms/step - loss: 0.0867 -
accuracy: 0.9659 - val_loss: 0.7019 - val_accuracy: 0.8525
Epoch 96/100
250/250 [==============================] - 58s 233ms/step - loss: 0.0966 -
accuracy: 0.9630 - val_loss: 0.8162 - val_accuracy: 0.8230
Epoch 97/100
250/250 [==============================] - 59s 235ms/step - loss: 0.0915 -
accuracy: 0.9663 - val_loss: 0.6270 - val_accuracy: 0.8565
Epoch 98/100
250/250 [==============================] - 59s 236ms/step - loss: 0.0839 -
accuracy: 0.9685 - val_loss: 0.6079 - val_accuracy: 0.8580
Epoch 99/100
250/250 [==============================] - 60s 240ms/step - loss: 0.0862 -
accuracy: 0.9663 - val_loss: 0.6783 - val_accuracy: 0.8530
Epoch 100/100
250/250 [==============================] - 58s 234ms/step - loss: 0.0830
```

Ploteamos el resultado

In [8]:

```python
def plot_resultados_training(history):
    fig, axes = plt.subplots(1,2, figsize=(18,6))
    axes[0].plot(history.history['accuracy'], label='Train')
    axes[0].plot(history.history['val_accuracy'], label='Validation')
    axes[0].legend()
    axes[0].set_title('Accuracy')
    axes[1].plot(history.history['loss'], label='Train')
    axes[1].plot(history.history['val_loss'], label='Validation')
    axes[1].set_title('Cross entropy')

plot_resultados_training(history1)
```



In [9]:

```python
classifier.save('./models/clasificador6')
```

```
WARNING:tensorflow:From /home/llbernat/anaconda3/envs/musi-ap/lib/python3.7
/site-packages/tensorflow/python/training/tracking/tracking.py:111: Model.s
tate_updates (from tensorflow.python.keras.engine.training) is deprecated a
```

```
nd will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied
automatically.
WARNING:tensorflow:From /home/llbernat/anaconda3/envs/musi-ap/lib/python3.7
/site-packages/tensorflow/python/training/tracking/tracking.py:111: Layer.u
pdates (from tensorflow.python.keras.engine.base_layer) is deprecated and w
ill be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied
automatically.
```

Comentario

En este intento hemos ampliado a tres capas de convolución con 32, 32 y 64 filtros de 3x3. El resultado ha sido esperanzador, pues hemos llegado a una precisón del 85.2% en validación.

Propuesta de mejora

Los mejora de resultados en cuanto hemos *complicado* la arquitectura de las capas convolucionales nos hace pensar que quizás ese sea el camino.