Intento 5

La diferencia entre este intento y el anterior es que hemos reducido el batch_size de 32 a 20 imágenes por lote.

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

def crear_clasificador_intento5():
    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(Flatten())
    classifier.add(Dense(units = 128, activation = "relu"))
    classifier.add(Dropout(0.5))
    classifier.add(Dense(units = 1, activation = "sigmoid"))
    return classifier
```

2. Entrenamiento del intento 5

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]:
```python
classifier = crear_clasificador_intento5()
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
_________________________________________________________________
flatten (Flatten)            (None, 15488)             0
_________________________________________________________________
dense (Dense)                (None, 128)               1982592
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,992,865
Trainable params: 1,992,865
Non-trainable params: 0
_________________________________________________________________
```

En segundo lugar, generamos los datasets de entrenamiento y test. Emplearemos un tamaño de batch de 20

In [6]:

```
from keras.preprocessing.image import ImageDataGenerator

batch_size=20

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_size
                                                     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
400/400 [==============================] - 65s 163ms/step - loss: 0.6891 - accuracy: 0.5504 - val_loss: 0.6769 - val_accuracy: 0.5725
Epoch 2/100
400/400 [==============================] - 66s 165ms/step - loss: 0.6621 - accuracy: 0.6101 - val_loss: 0.6686 - val_accuracy: 0.6155
Epoch 3/100
400/400 [==============================] - 69s 173ms/step - loss: 0.6286 - accuracy: 0.6530 - val_loss: 0.6107 - val_accuracy: 0.6815
Epoch 4/100
400/400 [==============================] - 68s 169ms/step - loss: 0.5944 - accuracy: 0.6865 - val_loss: 0.5768 - val_accuracy: 0.7135
Epoch 5/100
400/400 [==============================] - 66s 165ms/step - loss: 0.5739 - accuracy: 0.7014 - val_loss: 0.5922 - val_accuracy: 0.6945
Epoch 6/100
400/400 [==============================] - 65s 163ms/step - loss: 0.5565 - accuracy: 0.7147 - val_loss: 0.5656 - val_accuracy: 0.7270
Epoch 7/100
400/400 [==============================] - 66s 164ms/step - loss: 0.5310 - accuracy: 0.7361 - val_loss: 0.5642 - val_accuracy: 0.7240
```

```
Epoch 8/100
400/400 [==============================] - 66s 164ms/step - loss: 0.5191 -
accuracy: 0.7475 - val_loss: 0.5380 - val_accuracy: 0.7330
Epoch 9/100
400/400 [==============================] - 67s 168ms/step - loss: 0.5003 -
accuracy: 0.7602 - val_loss: 0.4951 - val_accuracy: 0.7625
Epoch 10/100
400/400 [==============================] - 64s 161ms/step - loss: 0.4823 -
accuracy: 0.7661 - val_loss: 0.5526 - val_accuracy: 0.7370
Epoch 11/100
400/400 [==============================] - 66s 165ms/step - loss: 0.4839 -
accuracy: 0.7684 - val_loss: 0.5245 - val_accuracy: 0.7415
Epoch 12/100
400/400 [==============================] - 71s 177ms/step - loss: 0.4714 -
accuracy: 0.7747 - val_loss: 0.5040 - val_accuracy: 0.7620
Epoch 13/100
400/400 [==============================] - 71s 177ms/step - loss: 0.4546 -
accuracy: 0.7864 - val_loss: 0.4805 - val_accuracy: 0.7840
Epoch 14/100
400/400 [==============================] - 71s 177ms/step - loss: 0.4455 -
accuracy: 0.7894 - val_loss: 0.4879 - val_accuracy: 0.7770
Epoch 15/100
400/400 [==============================] - 66s 166ms/step - loss: 0.4422 -
accuracy: 0.7956 - val_loss: 0.4864 - val_accuracy: 0.7785
Epoch 16/100
400/400 [==============================] - 66s 165ms/step - loss: 0.4395 -
accuracy: 0.7946 - val_loss: 0.4869 - val_accuracy: 0.7810
Epoch 17/100
400/400 [==============================] - 66s 165ms/step - loss: 0.4238 -
accuracy: 0.8050 - val_loss: 0.4827 - val_accuracy: 0.7745
Epoch 18/100
400/400 [==============================] - 66s 166ms/step - loss: 0.4185 -
accuracy: 0.8046 - val_loss: 0.4856 - val_accuracy: 0.7800
Epoch 19/100
400/400 [==============================] - 66s 165ms/step - loss: 0.4115 -
accuracy: 0.8124 - val_loss: 0.4728 - val_accuracy: 0.7655
Epoch 20/100
400/400 [==============================] - 63s 157ms/step - loss: 0.4087 -
accuracy: 0.8138 - val_loss: 0.4641 - val_accuracy: 0.7785
Epoch 21/100
400/400 [==============================] - 64s 159ms/step - loss: 0.3924 -
accuracy: 0.8263 - val_loss: 0.4703 - val_accuracy: 0.7870
Epoch 22/100
400/400 [==============================] - 63s 158ms/step - loss: 0.3855 -
accuracy: 0.8259 - val_loss: 0.4679 - val_accuracy: 0.7840
Epoch 23/100
400/400 [==============================] - 63s 157ms/step - loss: 0.3748 -
accuracy: 0.8319 - val_loss: 0.4627 - val_accuracy: 0.7895
Epoch 24/100
400/400 [==============================] - 69s 173ms/step - loss: 0.3736 -
accuracy: 0.8311 - val_loss: 0.4907 - val_accuracy: 0.7965
Epoch 25/100
400/400 [==============================] - 68s 170ms/step - loss: 0.3658 -
accuracy: 0.8396 - val_loss: 0.4536 - val_accuracy: 0.8075
Epoch 26/100
400/400 [==============================] - 66s 166ms/step - loss: 0.3615 -
accuracy: 0.8384 - val_loss: 0.4898 - val_accuracy: 0.7895
Epoch 27/100
400/400 [==============================] - 67s 167ms/step - loss: 0.3400 -
accuracy: 0.8516 - val_loss: 0.5183 - val_accuracy: 0.7835
Epoch 28/100
400/400 [==============================] - 70s 174ms/step - loss: 0.3558 -
accuracy: 0.8435 - val_loss: 0.4958 - val_accuracy: 0.7805
Epoch 29/100
```

```
400/400 [==============================] - 68s 171ms/step - loss: 0.3407 -
accuracy: 0.8504 - val_loss: 0.5429 - val_accuracy: 0.7720
Epoch 30/100
400/400 [==============================] - 63s 158ms/step - loss: 0.3372 -
accuracy: 0.8520 - val_loss: 0.5438 - val_accuracy: 0.7790
Epoch 31/100
400/400 [==============================] - 63s 157ms/step - loss: 0.3417 -
accuracy: 0.8518 - val_loss: 0.4823 - val_accuracy: 0.7935
Epoch 32/100
400/400 [==============================] - 63s 157ms/step - loss: 0.3159 -
accuracy: 0.8634 - val_loss: 0.5004 - val_accuracy: 0.7855
Epoch 33/100
400/400 [==============================] - 62s 155ms/step - loss: 0.3286 -
accuracy: 0.8586 - val_loss: 0.4856 - val_accuracy: 0.7905
Epoch 34/100
400/400 [==============================] - 63s 158ms/step - loss: 0.3126 -
accuracy: 0.8669 - val_loss: 0.4897 - val_accuracy: 0.7940
Epoch 35/100
400/400 [==============================] - 62s 155ms/step - loss: 0.3092 -
accuracy: 0.8660 - val_loss: 0.5026 - val_accuracy: 0.7945
Epoch 36/100
400/400 [==============================] - 62s 155ms/step - loss: 0.3041 -
accuracy: 0.8677 - val_loss: 0.5205 - val_accuracy: 0.7865
Epoch 37/100
400/400 [==============================] - 62s 155ms/step - loss: 0.2952 -
accuracy: 0.8736 - val_loss: 0.4962 - val_accuracy: 0.7980
Epoch 38/100
400/400 [==============================] - 62s 155ms/step - loss: 0.3058 -
accuracy: 0.8690 - val_loss: 0.5053 - val_accuracy: 0.7900
Epoch 39/100
400/400 [==============================] - 62s 155ms/step - loss: 0.2920 -
accuracy: 0.8726 - val_loss: 0.5193 - val_accuracy: 0.8000
Epoch 40/100
400/400 [==============================] - 62s 156ms/step - loss: 0.2894 -
accuracy: 0.8755 - val_loss: 0.5133 - val_accuracy: 0.8030
Epoch 41/100
400/400 [==============================] - 63s 157ms/step - loss: 0.2772 -
accuracy: 0.8783 - val_loss: 0.5399 - val_accuracy: 0.7975
Epoch 42/100
400/400 [==============================] - 62s 154ms/step - loss: 0.2794 -
accuracy: 0.8850 - val_loss: 0.5444 - val_accuracy: 0.7925
Epoch 43/100
400/400 [==============================] - 63s 159ms/step - loss: 0.2748 -
accuracy: 0.8802 - val_loss: 0.5327 - val_accuracy: 0.7995
Epoch 44/100
400/400 [==============================] - 62s 156ms/step - loss: 0.2681 -
accuracy: 0.8839 - val_loss: 0.5396 - val_accuracy: 0.7890
Epoch 45/100
400/400 [==============================] - 63s 156ms/step - loss: 0.2691 -
accuracy: 0.8831 - val_loss: 0.5437 - val_accuracy: 0.8030
Epoch 46/100
400/400 [==============================] - 62s 156ms/step - loss: 0.2597 -
accuracy: 0.8901 - val_loss: 0.5166 - val_accuracy: 0.8040
Epoch 47/100
400/400 [==============================] - 63s 158ms/step - loss: 0.2591 -
accuracy: 0.8969 - val_loss: 0.5111 - val_accuracy: 0.7975
Epoch 48/100
400/400 [==============================] - 62s 155ms/step - loss: 0.2594 -
accuracy: 0.8934 - val_loss: 0.5454 - val_accuracy: 0.7945
Epoch 49/100
400/400 [==============================] - 63s 158ms/step - loss: 0.2585 -
accuracy: 0.8916 - val_loss: 0.5445 - val_accuracy: 0.7975
Epoch 50/100
400/400 [==============================] - 62s 154ms/step - loss: 0.2539 -
```

```
accuracy: 0.8974 - val_loss: 0.5336 - val_accuracy: 0.7845
Epoch 51/100
400/400 [==============================] - 62s 156ms/step - loss: 0.2547 -
accuracy: 0.8904 - val_loss: 0.5987 - val_accuracy: 0.8015
Epoch 52/100
400/400 [==============================] - 72s 180ms/step - loss: 0.2498 -
accuracy: 0.8972 - val_loss: 0.5794 - val_accuracy: 0.7970
Epoch 53/100
400/400 [==============================] - 70s 176ms/step - loss: 0.2373 -
accuracy: 0.9010 - val_loss: 0.5582 - val_accuracy: 0.8010
Epoch 54/100
400/400 [==============================] - 71s 177ms/step - loss: 0.2451 -
accuracy: 0.8970 - val_loss: 0.5773 - val_accuracy: 0.8020
Epoch 55/100
400/400 [==============================] - 68s 170ms/step - loss: 0.2380 -
accuracy: 0.9060 - val_loss: 0.5629 - val_accuracy: 0.7995
Epoch 56/100
400/400 [==============================] - 68s 170ms/step - loss: 0.2347 -
accuracy: 0.9034 - val_loss: 0.5397 - val_accuracy: 0.7940
Epoch 57/100
400/400 [==============================] - 66s 165ms/step - loss: 0.2287 -
accuracy: 0.9065 - val_loss: 0.5475 - val_accuracy: 0.8060
Epoch 58/100
400/400 [==============================] - 69s 173ms/step - loss: 0.2260 -
accuracy: 0.9076 - val_loss: 0.5976 - val_accuracy: 0.7885
Epoch 59/100
400/400 [==============================] - 68s 169ms/step - loss: 0.2221 -
accuracy: 0.9069 - val_loss: 0.5555 - val_accuracy: 0.7940
Epoch 60/100
400/400 [==============================] - 63s 157ms/step - loss: 0.2264 -
accuracy: 0.9064 - val_loss: 0.5553 - val_accuracy: 0.8035
Epoch 61/100
400/400 [==============================] - 62s 156ms/step - loss: 0.2195 -
accuracy: 0.9104 - val_loss: 0.5952 - val_accuracy: 0.8105
Epoch 62/100
400/400 [==============================] - 62s 156ms/step - loss: 0.2202 -
accuracy: 0.9115 - val_loss: 0.5896 - val_accuracy: 0.8030
Epoch 63/100
400/400 [==============================] - 62s 155ms/step - loss: 0.2020 -
accuracy: 0.9166 - val_loss: 0.6615 - val_accuracy: 0.8030
Epoch 64/100
400/400 [==============================] - 61s 152ms/step - loss: 0.2156 -
accuracy: 0.9097 - val_loss: 0.6891 - val_accuracy: 0.8000
Epoch 65/100
400/400 [==============================] - 62s 156ms/step - loss: 0.2139 -
accuracy: 0.9143 - val_loss: 0.5849 - val_accuracy: 0.8095
Epoch 66/100
400/400 [==============================] - 62s 155ms/step - loss: 0.2059 -
accuracy: 0.9143 - val_loss: 0.6366 - val_accuracy: 0.8025
Epoch 67/100
400/400 [==============================] - 61s 152ms/step - loss: 0.2001 -
accuracy: 0.9215 - val_loss: 0.5902 - val_accuracy: 0.8075
Epoch 68/100
400/400 [==============================] - 62s 155ms/step - loss: 0.1977 -
accuracy: 0.9168 - val_loss: 0.6109 - val_accuracy: 0.8015
Epoch 69/100
400/400 [==============================] - 63s 156ms/step - loss: 0.2147 -
accuracy: 0.9159 - val_loss: 0.6387 - val_accuracy: 0.7945
Epoch 70/100
400/400 [==============================] - 62s 154ms/step - loss: 0.2076 -
accuracy: 0.9120 - val_loss: 0.6369 - val_accuracy: 0.7985
Epoch 71/100
400/400 [==============================] - 61s 154ms/step - loss: 0.2030 -
accuracy: 0.9197 - val_loss: 0.6448 - val_accuracy: 0.8030
```

```
Epoch 72/100
400/400 [==============================] - 62s 156ms/step - loss: 0.1865 -
accuracy: 0.9279 - val_loss: 0.6212 - val_accuracy: 0.8065
Epoch 73/100
400/400 [==============================] - 64s 159ms/step - loss: 0.1964 -
accuracy: 0.9195 - val_loss: 0.6680 - val_accuracy: 0.8040
Epoch 74/100
400/400 [==============================] - 63s 157ms/step - loss: 0.1975 -
accuracy: 0.9212 - val_loss: 0.7121 - val_accuracy: 0.7865
Epoch 75/100
400/400 [==============================] - 63s 157ms/step - loss: 0.1908 -
accuracy: 0.9218 - val_loss: 0.6666 - val_accuracy: 0.7995
Epoch 76/100
400/400 [==============================] - 62s 155ms/step - loss: 0.1822 -
accuracy: 0.9276 - val_loss: 0.6559 - val_accuracy: 0.7880
Epoch 77/100
400/400 [==============================] - 62s 156ms/step - loss: 0.1919 -
accuracy: 0.9226 - val_loss: 0.6945 - val_accuracy: 0.7960
Epoch 78/100
400/400 [==============================] - 62s 155ms/step - loss: 0.1938 -
accuracy: 0.9214 - val_loss: 0.6047 - val_accuracy: 0.8000
Epoch 79/100
400/400 [==============================] - 62s 155ms/step - loss: 0.1853 -
accuracy: 0.9241 - val_loss: 0.6136 - val_accuracy: 0.8070
Epoch 80/100
400/400 [==============================] - 62s 155ms/step - loss: 0.1785 -
accuracy: 0.9281 - val_loss: 0.6507 - val_accuracy: 0.7925
Epoch 81/100
400/400 [==============================] - 61s 153ms/step - loss: 0.1836 -
accuracy: 0.9270 - val_loss: 0.6070 - val_accuracy: 0.7980
Epoch 82/100
400/400 [==============================] - 62s 155ms/step - loss: 0.1713 -
accuracy: 0.9296 - val_loss: 0.7638 - val_accuracy: 0.7960
Epoch 83/100
400/400 [==============================] - 62s 155ms/step - loss: 0.1836 -
accuracy: 0.9244 - val_loss: 0.6577 - val_accuracy: 0.7990
Epoch 84/100
400/400 [==============================] - 63s 157ms/step - loss: 0.1817 -
accuracy: 0.9250 - val_loss: 0.6616 - val_accuracy: 0.8115
Epoch 85/100
400/400 [==============================] - 62s 156ms/step - loss: 0.1775 -
accuracy: 0.9309 - val_loss: 0.6322 - val_accuracy: 0.7995
Epoch 86/100
400/400 [==============================] - 62s 156ms/step - loss: 0.1709 -
accuracy: 0.9293 - val_loss: 0.6730 - val_accuracy: 0.8000
Epoch 87/100
400/400 [==============================] - 62s 155ms/step - loss: 0.1829 -
accuracy: 0.9285 - val_loss: 0.6465 - val_accuracy: 0.8030
Epoch 88/100
400/400 [==============================] - 62s 155ms/step - loss: 0.1707 -
accuracy: 0.9306 - val_loss: 0.6676 - val_accuracy: 0.8105
Epoch 89/100
400/400 [==============================] - 64s 161ms/step - loss: 0.1661 -
accuracy: 0.9349 - val_loss: 0.6971 - val_accuracy: 0.8020
Epoch 90/100
400/400 [==============================] - 62s 155ms/step - loss: 0.1721 -
accuracy: 0.9326 - val_loss: 0.6851 - val_accuracy: 0.8090
Epoch 91/100
400/400 [==============================] - 62s 155ms/step - loss: 0.1680 -
accuracy: 0.9312 - val_loss: 0.6744 - val_accuracy: 0.8095
Epoch 92/100
400/400 [==============================] - 62s 155ms/step - loss: 0.1765 -
accuracy: 0.9281 - val_loss: 0.6401 - val_accuracy: 0.8130
Epoch 93/100
```

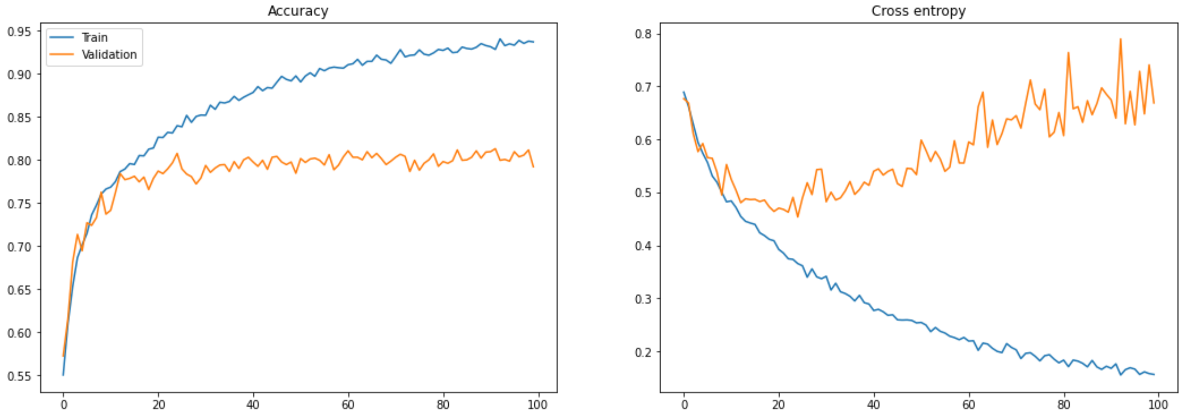
```
400/400 [==============================] - 63s 157ms/step - loss: 0.1555 - accuracy: 0.9404 - val_loss: 0.7896 - val_accuracy: 0.7995
Epoch 94/100
400/400 [==============================] - 63s 158ms/step - loss: 0.1656 - accuracy: 0.9325 - val_loss: 0.6293 - val_accuracy: 0.8005
Epoch 95/100
400/400 [==============================] - 64s 159ms/step - loss: 0.1692 - accuracy: 0.9346 - val_loss: 0.6907 - val_accuracy: 0.7985
Epoch 96/100
400/400 [==============================] - 70s 176ms/step - loss: 0.1666 - accuracy: 0.9330 - val_loss: 0.6273 - val_accuracy: 0.8095
Epoch 97/100
400/400 [==============================] - 71s 178ms/step - loss: 0.1566 - accuracy: 0.9386 - val_loss: 0.7284 - val_accuracy: 0.8035
Epoch 98/100
400/400 [==============================] - 71s 178ms/step - loss: 0.1615 - accuracy: 0.9351 - val_loss: 0.6480 - val_accuracy: 0.8055
Epoch 99/100
400/400 [==============================] - 71s 177ms/step - loss: 0.1582 - accuracy: 0.9377 - val_loss: 0.7406 - val_accuracy: 0.8115
Epoch 100/100
```

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/clasificador5')
```

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

La reducción del número de imágenes por lote no ha dado el resultado esperado. De hecho en esta ejecución ha arrojado precisiones un poco peores que en el anterior y se sigue produciendo sobre-entrenamiento, que despunta a partir de la epoch 20.

Propuesta de mejora

Un número más rico de *features* a la salida de la convolución también puede favorecer que la red neuronal valore otros patrones de las imagenes que ayuden a evitar el sobre-entrenamiento.

Porponemos la creacion de una tercera capa de convolucion mas rica en filtros (actualmente trabajamos con 2) y volver al tamaño de lote por defecto en keras (32).