Intento 7 - Arquitectura VGG

In [1]:
```
train_ds_path  ='C:/Users/Usuario/Documents/Master/Aprendizaje Profundo/Udemy/deeple
test_ds_path   ='C:/Users/Usuario/Documents/Master/Aprendizaje Profundo/Udemy/deeple
cat_or_dog_path='C:/Users/Usuario/Documents/Master/Aprendizaje Profundo/Udemy/deeple
```

Parte 1 - Construir el modelo de CNN

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

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

"""
Necesitamos la capa de olvido para evitar el sobre-entrenamiento
"""
from keras.layers import Dropout
```

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)
```

Inicializar la CNN

In [4]:
```
classifier = Sequential()
```

Convolución

Esta arquitectura apila varias capas convolucionales seguidos de max-pooling. El número de filtros de cada capa aumenta conforme más profunda es la red, doblando el número en cada nivel.

En este intento se apilan varias capas convolucionales seguidas de max-pooling. El número de filtros de cada capa aumenta conforme más profunda es la red, doblando el número en cada nivel. Se añade en cada bloque un dropout para ayudar a reducir overfitting reduciendo la

dependencia entre neuronas. Además se usa batch normalization para normalizar cada lote de datos y mejorar el rendimiento de la red.

In [5]:
```
Image_Width = 128
Image_Height = 128
Image_Size = (Image_Width,Image_Height)
Image_Channels = 3

# Número de canales de la imagen
Image_Channels = 3

# En este intento añadimos 3 convoluciones, cada una con una normalización, un max p
classifier.add(Conv2D(32, (3, 3), activation='relu', input_shape = (*Image_Size, Ima
classifier.add(BatchNormalization())
classifier.add(MaxPooling2D(pool_size=(2,2)))
classifier.add(Dropout(0.25))

# Doblamos el número de filtros
classifier.add(Conv2D(64, (3, 3), activation = 'relu'))
classifier.add(BatchNormalization())
classifier.add(MaxPooling2D(pool_size = (2,2)))
classifier.add(Dropout(0.25))

# Doblamos el número de filtros
classifier.add(Conv2D(128, (3, 3), activation = 'relu'))
classifier.add(BatchNormalization())
classifier.add(MaxPooling2D(pool_size = (2,2)))
classifier.add(Dropout(0.25))

# Añadimos dos full-connected layer con 512 unidades y 1 respectivamente; con un dro
classifier.add(Flatten())
classifier.add(Dense(units = 512, activation = 'relu'))
classifier.add(BatchNormalization())
classifier.add(Dropout(0.5))
classifier.add(Dense(units = 1, activation = 'sigmoid'))

classifier.compile(loss = 'binary_crossentropy',
                   optimizer = 'adam',
                   metrics = ['accuracy'])

classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      896
_________________________________________________________________
batch_normalization (BatchNo (None, 126, 126, 32)      128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 64)        18496
_________________________________________________________________
batch_normalization_1 (Batch (None, 61, 61, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 28, 28, 128)       512
_________________________________________________________________
```

```
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 14, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 512)               12845568
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 12,942,273
Trainable params: 12,940,801
Non-trainable params: 1,472
_________________________________________________________________
```

Parte 2 - Ajustar la CNN a las imágenes para entrenar

In [6]:
```
from keras.preprocessing.image import ImageDataGenerator

"""
batch_size=32

32 es el valor por defecto que usaría
model.fit_generator, aunque aquí lo debemos
especificar en los generadores en lugar de
en la llamada al bucle de entrenamiento
"""
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=Image_Size,
                                                     batch_size=batch_size,
                                                     class_mode='binary')

testing_dataset = test_datagen.flow_from_directory(test_ds_path,
                                                   target_size=Image_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 para hacer early stopping 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, patience =
]

history = 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 rendimiento
```

```
Epoch 1/100
250/250 [==============================] - 168s 669ms/step - loss: 0.9507 - accurac
y: 0.6004 - val_loss: 0.7378 - val_accuracy: 0.5125
Epoch 2/100
250/250 [==============================] - 166s 664ms/step - loss: 0.6532 - accurac
y: 0.6697 - val_loss: 0.6930 - val_accuracy: 0.6260
Epoch 3/100
250/250 [==============================] - 166s 663ms/step - loss: 0.5770 - accurac
y: 0.7057 - val_loss: 0.5682 - val_accuracy: 0.7035
Epoch 4/100
250/250 [==============================] - 166s 665ms/step - loss: 0.5353 - accurac
y: 0.7275 - val_loss: 0.4795 - val_accuracy: 0.7705
Epoch 5/100
250/250 [==============================] - 167s 666ms/step - loss: 0.4817 - accurac
y: 0.7681 - val_loss: 0.7700 - val_accuracy: 0.6030
Epoch 6/100
250/250 [==============================] - 166s 661ms/step - loss: 0.4657 - accurac
y: 0.7855 - val_loss: 0.4898 - val_accuracy: 0.7655
Epoch 7/100
250/250 [==============================] - 165s 657ms/step - loss: 0.4435 - accurac
y: 0.7884 - val_loss: 0.4844 - val_accuracy: 0.7715
Epoch 8/100
250/250 [==============================] - 166s 665ms/step - loss: 0.4358 - accurac
y: 0.7937 - val_loss: 0.4750 - val_accuracy: 0.7830
Epoch 9/100
250/250 [==============================] - 170s 679ms/step - loss: 0.4024 - accurac
y: 0.8157 - val_loss: 0.5584 - val_accuracy: 0.7610
Epoch 10/100
250/250 [==============================] - 168s 673ms/step - loss: 0.3888 - accurac
y: 0.8157 - val_loss: 0.3950 - val_accuracy: 0.8150
Epoch 11/100
250/250 [==============================] - 166s 661ms/step - loss: 0.3669 - accurac
y: 0.8354 - val_loss: 0.3614 - val_accuracy: 0.8400
Epoch 12/100
250/250 [==============================] - 167s 669ms/step - loss: 0.3452 - accurac
y: 0.8449 - val_loss: 0.3894 - val_accuracy: 0.8355
Epoch 13/100
250/250 [==============================] - 167s 665ms/step - loss: 0.3419 - accurac
y: 0.8541 - val_loss: 0.3496 - val_accuracy: 0.8460
Epoch 14/100
250/250 [==============================] - 165s 658ms/step - loss: 0.3346 - accurac
y: 0.8543 - val_loss: 0.3716 - val_accuracy: 0.8505
Epoch 15/100
250/250 [==============================] - 165s 658ms/step - loss: 0.3298 - accurac
y: 0.8559 - val_loss: 1.3295 - val_accuracy: 0.5985
Epoch 16/100
250/250 [==============================] - 160s 640ms/step - loss: 0.3488 - accurac
y: 0.8372 - val_loss: 0.7945 - val_accuracy: 0.7180
Epoch 17/100
250/250 [==============================] - 160s 638ms/step - loss: 0.3013 - accurac
y: 0.8680 - val_loss: 0.3721 - val_accuracy: 0.8385
Epoch 18/100
250/250 [==============================] - 160s 639ms/step - loss: 0.2898 - accurac
y: 0.8811 - val_loss: 0.4090 - val_accuracy: 0.8300
Epoch 19/100
250/250 [==============================] - 159s 637ms/step - loss: 0.2916 - accurac
y: 0.8733 - val_loss: 0.3696 - val_accuracy: 0.8570
Epoch 20/100
250/250 [==============================] - 160s 639ms/step - loss: 0.2812 - accurac
y: 0.8770 - val_loss: 0.3899 - val_accuracy: 0.8390
Epoch 21/100
250/250 [==============================] - 159s 634ms/step - loss: 0.2967 - accurac
y: 0.8722 - val_loss: 0.4093 - val_accuracy: 0.8385
Epoch 22/100
```

```
250/250 [==============================] - 159s 637ms/step - loss: 0.2848 - accurac
y: 0.8800 - val_loss: 1.7105 - val_accuracy: 0.6525
Epoch 23/100
250/250 [==============================] - 159s 636ms/step - loss: 0.2873 - accurac
y: 0.8748 - val_loss: 0.3634 - val_accuracy: 0.8465
Epoch 24/100
250/250 [==============================] - 160s 639ms/step - loss: 0.2614 - accurac
y: 0.8945 - val_loss: 0.3451 - val_accuracy: 0.8550
Epoch 25/100
250/250 [==============================] - 159s 637ms/step - loss: 0.2555 - accurac
y: 0.8906 - val_loss: 0.3844 - val_accuracy: 0.8620
Epoch 26/100
250/250 [==============================] - 159s 636ms/step - loss: 0.2645 - accurac
y: 0.8913 - val_loss: 0.3652 - val_accuracy: 0.8660
Epoch 27/100
250/250 [==============================] - 160s 638ms/step - loss: 0.2910 - accurac
y: 0.8782 - val_loss: 0.8557 - val_accuracy: 0.7680
Epoch 28/100
250/250 [==============================] - 159s 637ms/step - loss: 0.2448 - accurac
y: 0.8997 - val_loss: 0.4310 - val_accuracy: 0.8425
Epoch 29/100
250/250 [==============================] - 159s 637ms/step - loss: 0.2317 - accurac
y: 0.9022 - val_loss: 0.5229 - val_accuracy: 0.8120
Epoch 30/100
250/250 [==============================] - 159s 637ms/step - loss: 0.2178 - accurac
y: 0.9099 - val_loss: 0.3100 - val_accuracy: 0.8805
Epoch 31/100
250/250 [==============================] - 160s 638ms/step - loss: 0.2058 - accurac
y: 0.9159 - val_loss: 0.3445 - val_accuracy: 0.8725
Epoch 32/100
250/250 [==============================] - 159s 637ms/step - loss: 0.2438 - accurac
y: 0.8974 - val_loss: 0.6356 - val_accuracy: 0.7845
Epoch 33/100
250/250 [==============================] - 159s 637ms/step - loss: 0.2136 - accurac
y: 0.9112 - val_loss: 0.3619 - val_accuracy: 0.8600
Epoch 34/100
250/250 [==============================] - 159s 634ms/step - loss: 0.2027 - accurac
y: 0.9152 - val_loss: 0.3490 - val_accuracy: 0.8815
Epoch 35/100
250/250 [==============================] - 160s 641ms/step - loss: 0.1806 - accurac
y: 0.9270 - val_loss: 0.3394 - val_accuracy: 0.8700
Epoch 36/100
250/250 [==============================] - 159s 636ms/step - loss: 0.1974 - accurac
y: 0.9179 - val_loss: 0.4025 - val_accuracy: 0.8600
Epoch 37/100
250/250 [==============================] - 160s 639ms/step - loss: 0.1758 - accurac
y: 0.9287 - val_loss: 0.3702 - val_accuracy: 0.8555
Epoch 38/100
250/250 [==============================] - 160s 639ms/step - loss: 0.2236 - accurac
y: 0.9119 - val_loss: 0.6570 - val_accuracy: 0.7715
Epoch 39/100
250/250 [==============================] - 160s 640ms/step - loss: 0.2433 - accurac
y: 0.9032 - val_loss: 0.5882 - val_accuracy: 0.7915
Epoch 40/100
250/250 [==============================] - 159s 634ms/step - loss: 0.2302 - accurac
y: 0.9069 - val_loss: 0.3426 - val_accuracy: 0.8755
Epoch 41/100
250/250 [==============================] - 160s 638ms/step - loss: 0.2084 - accurac
y: 0.9138 - val_loss: 0.3246 - val_accuracy: 0.8750
Epoch 42/100
250/250 [==============================] - 160s 639ms/step - loss: 0.1871 - accurac
y: 0.9280 - val_loss: 0.2830 - val_accuracy: 0.8890
Epoch 43/100
250/250 [==============================] - 160s 641ms/step - loss: 0.1858 - accurac
y: 0.9285 - val_loss: 0.3991 - val_accuracy: 0.8650
Epoch 44/100
250/250 [==============================] - 159s 635ms/step - loss: 0.1806 - accurac
y: 0.9256 - val_loss: 0.3075 - val_accuracy: 0.8830
Epoch 45/100
```

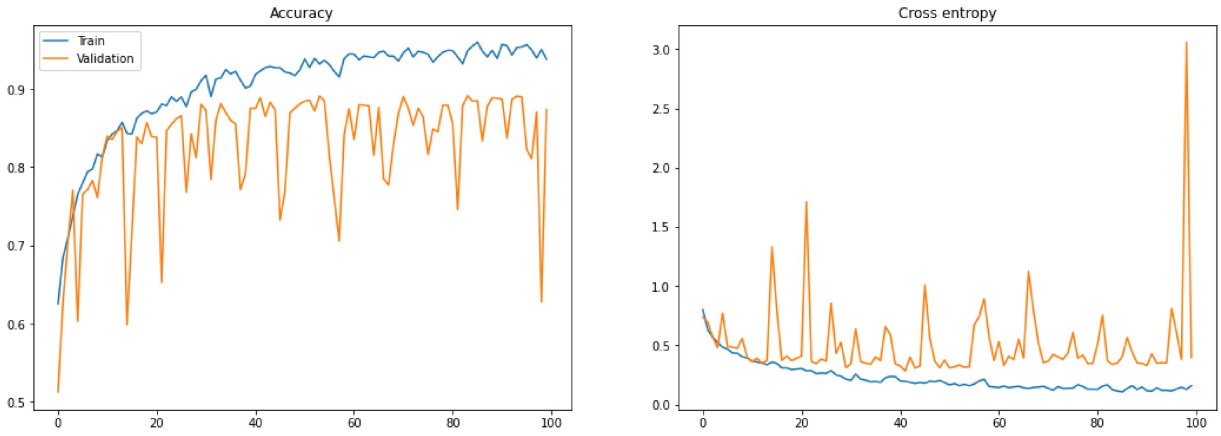
```
250/250 [==============================] - 159s 637ms/step - loss: 0.1824 - accurac
y: 0.9270 - val_loss: 0.3252 - val_accuracy: 0.8735
Epoch 46/100
250/250 [==============================] - 160s 638ms/step - loss: 0.1745 - accurac
y: 0.9292 - val_loss: 1.0096 - val_accuracy: 0.7325
Epoch 47/100
250/250 [==============================] - 159s 635ms/step - loss: 0.2157 - accurac
y: 0.9153 - val_loss: 0.5588 - val_accuracy: 0.7695
Epoch 48/100
250/250 [==============================] - 160s 639ms/step - loss: 0.2250 - accurac
y: 0.9030 - val_loss: 0.3673 - val_accuracy: 0.8695
Epoch 49/100
250/250 [==============================] - 160s 640ms/step - loss: 0.1937 - accurac
y: 0.9229 - val_loss: 0.3101 - val_accuracy: 0.8755
Epoch 50/100
250/250 [==============================] - 161s 643ms/step - loss: 0.1871 - accurac
y: 0.9242 - val_loss: 0.3748 - val_accuracy: 0.8810
Epoch 51/100
250/250 [==============================] - 159s 636ms/step - loss: 0.1573 - accurac
y: 0.9431 - val_loss: 0.3080 - val_accuracy: 0.8845
Epoch 52/100
250/250 [==============================] - 161s 642ms/step - loss: 0.1714 - accurac
y: 0.9293 - val_loss: 0.3183 - val_accuracy: 0.8855
Epoch 53/100
250/250 [==============================] - 159s 637ms/step - loss: 0.1565 - accurac
y: 0.9391 - val_loss: 0.3334 - val_accuracy: 0.8720
Epoch 54/100
250/250 [==============================] - 161s 644ms/step - loss: 0.1713 - accurac
y: 0.9300 - val_loss: 0.3139 - val_accuracy: 0.8910
Epoch 55/100
250/250 [==============================] - 160s 639ms/step - loss: 0.1341 - accurac
y: 0.9470 - val_loss: 0.3181 - val_accuracy: 0.8850
Epoch 56/100
250/250 [==============================] - 160s 639ms/step - loss: 0.1574 - accurac
y: 0.9372 - val_loss: 0.6729 - val_accuracy: 0.8145
Epoch 57/100
250/250 [==============================] - 159s 637ms/step - loss: 0.1621 - accurac
y: 0.9367 - val_loss: 0.7433 - val_accuracy: 0.7600
Epoch 58/100
250/250 [==============================] - 160s 639ms/step - loss: 0.2449 - accurac
y: 0.9003 - val_loss: 0.8917 - val_accuracy: 0.7055
Epoch 59/100
250/250 [==============================] - 160s 637ms/step - loss: 0.1639 - accurac
y: 0.9381 - val_loss: 0.5722 - val_accuracy: 0.8415
Epoch 60/100
250/250 [==============================] - 160s 637ms/step - loss: 0.1517 - accurac
y: 0.9433 - val_loss: 0.3710 - val_accuracy: 0.8750
Epoch 61/100
250/250 [==============================] - 160s 637ms/step - loss: 0.1336 - accurac
y: 0.9486 - val_loss: 0.5303 - val_accuracy: 0.8355
Epoch 62/100
250/250 [==============================] - 159s 635ms/step - loss: 0.1546 - accurac
y: 0.9392 - val_loss: 0.3287 - val_accuracy: 0.8800
Epoch 63/100
250/250 [==============================] - 159s 637ms/step - loss: 0.1415 - accurac
y: 0.9443 - val_loss: 0.4072 - val_accuracy: 0.8795
Epoch 64/100
250/250 [==============================] - 160s 638ms/step - loss: 0.1586 - accurac
y: 0.9359 - val_loss: 0.3801 - val_accuracy: 0.8785
Epoch 65/100
250/250 [==============================] - 160s 638ms/step - loss: 0.1419 - accurac
y: 0.9440 - val_loss: 0.5494 - val_accuracy: 0.8155
Epoch 66/100
250/250 [==============================] - 160s 640ms/step - loss: 0.1369 - accurac
y: 0.9501 - val_loss: 0.3934 - val_accuracy: 0.8765
Epoch 67/100
250/250 [==============================] - 160s 638ms/step - loss: 0.1315 - accurac
y: 0.9481 - val_loss: 1.1225 - val_accuracy: 0.7850
Epoch 68/100
```

```
250/250 [==============================] - 160s 639ms/step - loss: 0.1548 - accurac
y: 0.9360 - val_loss: 0.7960 - val_accuracy: 0.7775
Epoch 69/100
250/250 [==============================] - 160s 640ms/step - loss: 0.1417 - accurac
y: 0.9457 - val_loss: 0.5222 - val_accuracy: 0.8300
Epoch 70/100
250/250 [==============================] - 160s 640ms/step - loss: 0.1433 - accurac
y: 0.9399 - val_loss: 0.3535 - val_accuracy: 0.8690
Epoch 71/100
250/250 [==============================] - 160s 638ms/step - loss: 0.1332 - accurac
y: 0.9465 - val_loss: 0.3654 - val_accuracy: 0.8905
Epoch 72/100
250/250 [==============================] - 159s 635ms/step - loss: 0.1228 - accurac
y: 0.9509 - val_loss: 0.4230 - val_accuracy: 0.8755
Epoch 73/100
250/250 [==============================] - 161s 642ms/step - loss: 0.1457 - accurac
y: 0.9413 - val_loss: 0.4012 - val_accuracy: 0.8535
Epoch 74/100
250/250 [==============================] - 160s 639ms/step - loss: 0.1417 - accurac
y: 0.9447 - val_loss: 0.3811 - val_accuracy: 0.8755
Epoch 75/100
250/250 [==============================] - 159s 637ms/step - loss: 0.1354 - accurac
y: 0.9451 - val_loss: 0.4347 - val_accuracy: 0.8645
Epoch 76/100
250/250 [==============================] - 160s 641ms/step - loss: 0.1261 - accurac
y: 0.9476 - val_loss: 0.6064 - val_accuracy: 0.8160
Epoch 77/100
250/250 [==============================] - 160s 638ms/step - loss: 0.1419 - accurac
y: 0.9447 - val_loss: 0.3903 - val_accuracy: 0.8490
Epoch 78/100
250/250 [==============================] - 159s 635ms/step - loss: 0.1619 - accurac
y: 0.9377 - val_loss: 0.4163 - val_accuracy: 0.8455
Epoch 79/100
250/250 [==============================] - 159s 637ms/step - loss: 0.1291 - accurac
y: 0.9496 - val_loss: 0.3455 - val_accuracy: 0.8795
Epoch 80/100
250/250 [==============================] - 160s 640ms/step - loss: 0.1201 - accurac
y: 0.9518 - val_loss: 0.3459 - val_accuracy: 0.8795
Epoch 81/100
250/250 [==============================] - 160s 640ms/step - loss: 0.1304 - accurac
y: 0.9476 - val_loss: 0.5111 - val_accuracy: 0.8560
Epoch 82/100
250/250 [==============================] - 159s 637ms/step - loss: 0.1800 - accurac
y: 0.9329 - val_loss: 0.7523 - val_accuracy: 0.7460
Epoch 83/100
250/250 [==============================] - 160s 639ms/step - loss: 0.1784 - accurac
y: 0.9273 - val_loss: 0.3675 - val_accuracy: 0.8785
Epoch 84/100
250/250 [==============================] - 160s 638ms/step - loss: 0.1226 - accurac
y: 0.9511 - val_loss: 0.3372 - val_accuracy: 0.8915
Epoch 85/100
250/250 [==============================] - 160s 638ms/step - loss: 0.1145 - accurac
y: 0.9530 - val_loss: 0.3499 - val_accuracy: 0.8845
Epoch 86/100
250/250 [==============================] - 160s 639ms/step - loss: 0.1055 - accurac
y: 0.9603 - val_loss: 0.4047 - val_accuracy: 0.8850
Epoch 87/100
250/250 [==============================] - 159s 636ms/step - loss: 0.1230 - accurac
y: 0.9529 - val_loss: 0.5638 - val_accuracy: 0.8335
Epoch 88/100
250/250 [==============================] - 160s 640ms/step - loss: 0.1426 - accurac
y: 0.9475 - val_loss: 0.4423 - val_accuracy: 0.8780
Epoch 89/100
250/250 [==============================] - 160s 638ms/step - loss: 0.1349 - accurac
y: 0.9452 - val_loss: 0.3512 - val_accuracy: 0.8890
Epoch 90/100
250/250 [==============================] - 159s 636ms/step - loss: 0.1436 - accurac
y: 0.9348 - val_loss: 0.3468 - val_accuracy: 0.8885
Epoch 91/100
```

```
250/250 [==============================] - 163s 653ms/step - loss: 0.1209 - accurac
y: 0.9536 - val_loss: 0.3291 - val_accuracy: 0.8870
Epoch 92/100
250/250 [==============================] - 159s 636ms/step - loss: 0.1077 - accurac
y: 0.9566 - val_loss: 0.4283 - val_accuracy: 0.8370
Epoch 93/100
250/250 [==============================] - 159s 637ms/step - loss: 0.1492 - accurac
y: 0.9427 - val_loss: 0.3477 - val_accuracy: 0.8870
Epoch 94/100
250/250 [==============================] - 159s 635ms/step - loss: 0.1268 - accurac
y: 0.9487 - val_loss: 0.3529 - val_accuracy: 0.8910
Epoch 95/100
250/250 [==============================] - 161s 643ms/step - loss: 0.1213 - accurac
y: 0.9524 - val_loss: 0.3511 - val_accuracy: 0.8900
Epoch 96/100
250/250 [==============================] - 160s 639ms/step - loss: 0.1078 - accurac
y: 0.9618 - val_loss: 0.8103 - val_accuracy: 0.8225
Epoch 97/100
250/250 [==============================] - 160s 637ms/step - loss: 0.1309 - accurac
y: 0.9509 - val_loss: 0.6051 - val_accuracy: 0.8110
Epoch 98/100
250/250 [==============================] - 160s 640ms/step - loss: 0.1447 - accurac
y: 0.9390 - val_loss: 0.3812 - val_accuracy: 0.8705
Epoch 99/100
250/250 [==============================] - 159s 637ms/step - loss: 0.1279 - accurac
y: 0.9510 - val_loss: 3.0581 - val_accuracy: 0.6280
Epoch 100/100
250/250 [==============================] - 159s 636ms/step - loss: 0.1724 - accurac
y: 0.9335 - val_loss: 0.3962 - val_accuracy: 0.8735
```

Pintamos los resultados

In [8]:
```
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(history)
```



In [9]:
```
classifier.save('clasificador5')
```

```
INFO:tensorflow:Assets written to: clasificador5\assets
```

Parte 3 - Cómo hacer nuevas predicciones

In [10]:
```
import numpy as np
from keras.preprocessing import image
```

```
test_image = image.load_img(cat_or_dog_path, target_size = Image_Size)
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
training_dataset.class_indices
if result[0][0] == 1:
    prediction = 'dog'
else:
    prediction = 'cat'

print(prediction)
```

```
dog
```