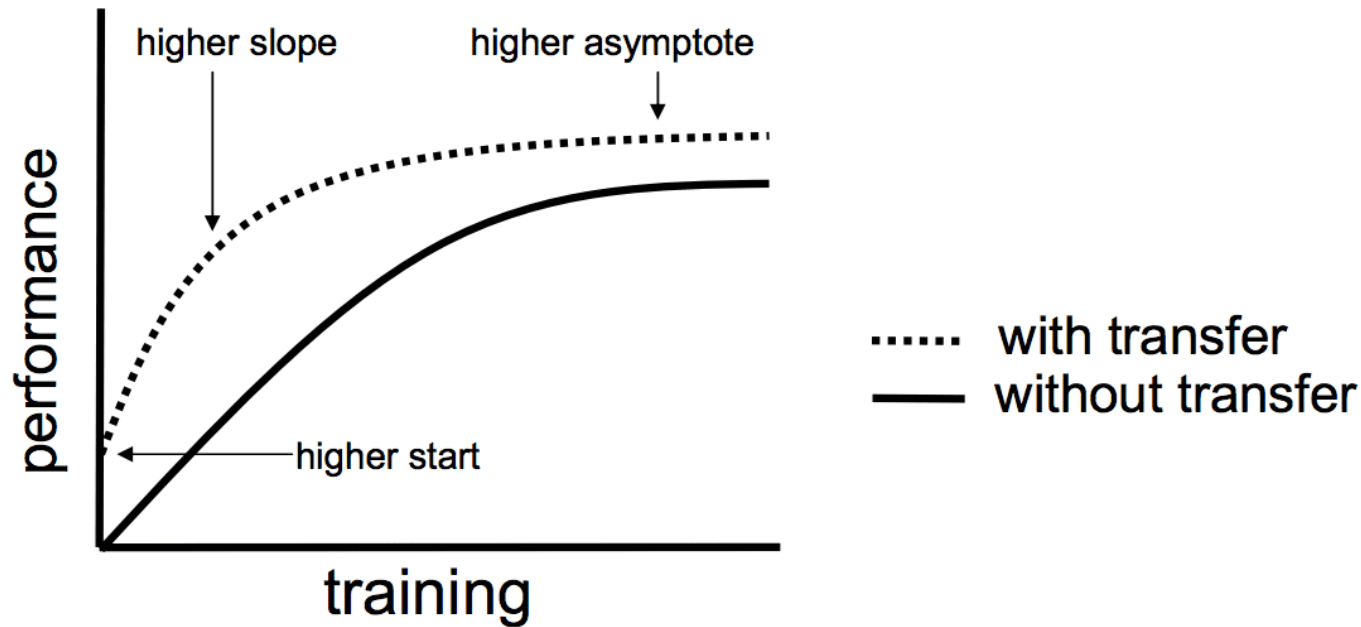


# **Tema 4. Transferencia de conocimiento**

Ana Jiménez Pastor  
anjipas@gmail.com

# Transferencia de conocimiento

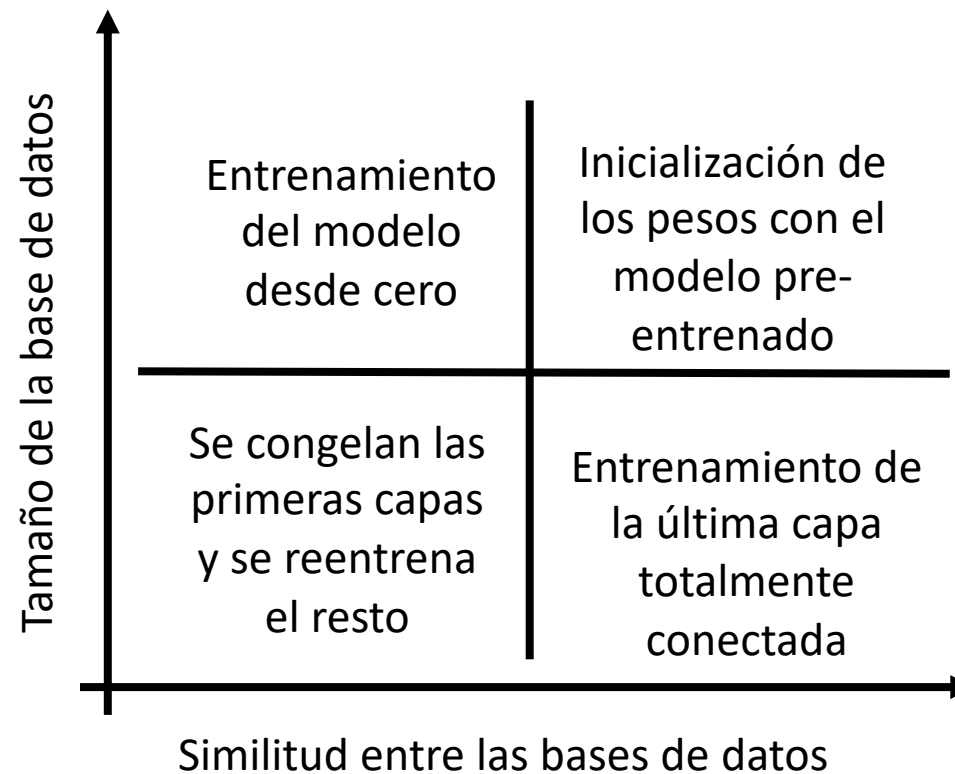
La transferencia de conocimiento (*transfer learning*) es la técnica según la cual un modelo entrenado para una tarea se emplea como **punto de partida** en la resolución de un problema diferente. Es una técnica de gran utilidad cuando no se cuenta de una gran base de datos de entrenamiento.



1. **Mayor punto de inicio:** el rendimiento del modelo al inicio de entrenamiento es mayor.
2. **Mayor pendiente:** convergencia del entrenamiento más rápida.
3. **Mayor asíntota:** mejor rendimiento del modelo al final del entrenamiento.

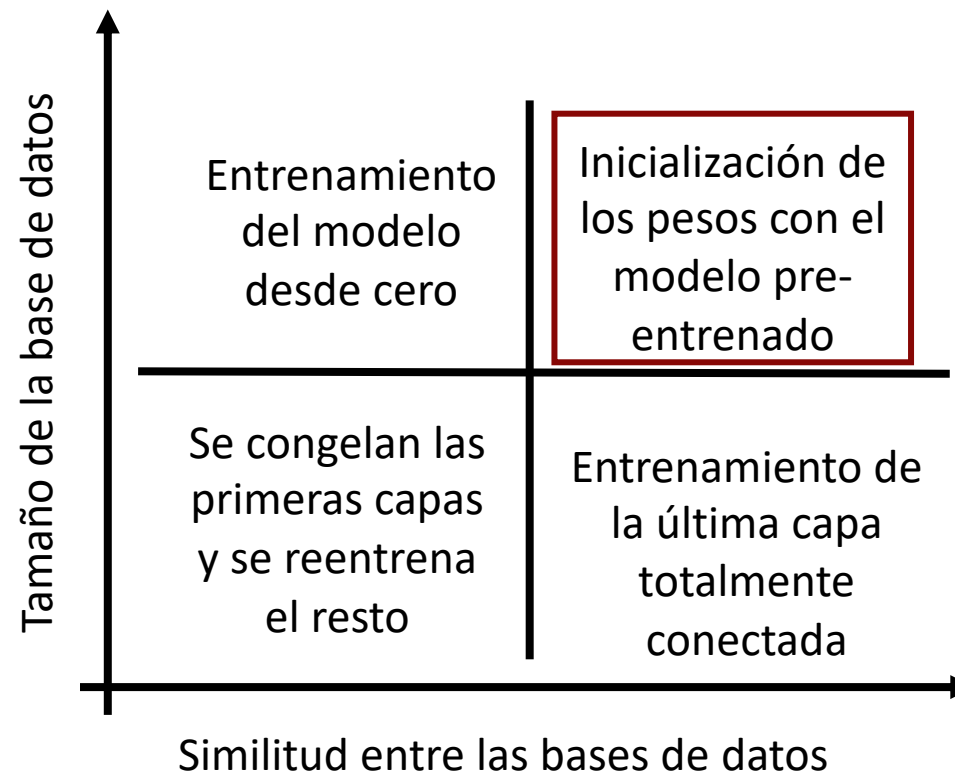
# Transferencia de conocimiento

La transferencia de conocimiento (*transfer learning*) es la técnica según la cual un modelo entrenado para una tarea se emplea como **punto de partida** en la resolución de un problema diferente. Es una técnica de gran utilidad cuando no se cuenta de una gran base de datos de entrenamiento.



# Transferencia de conocimiento

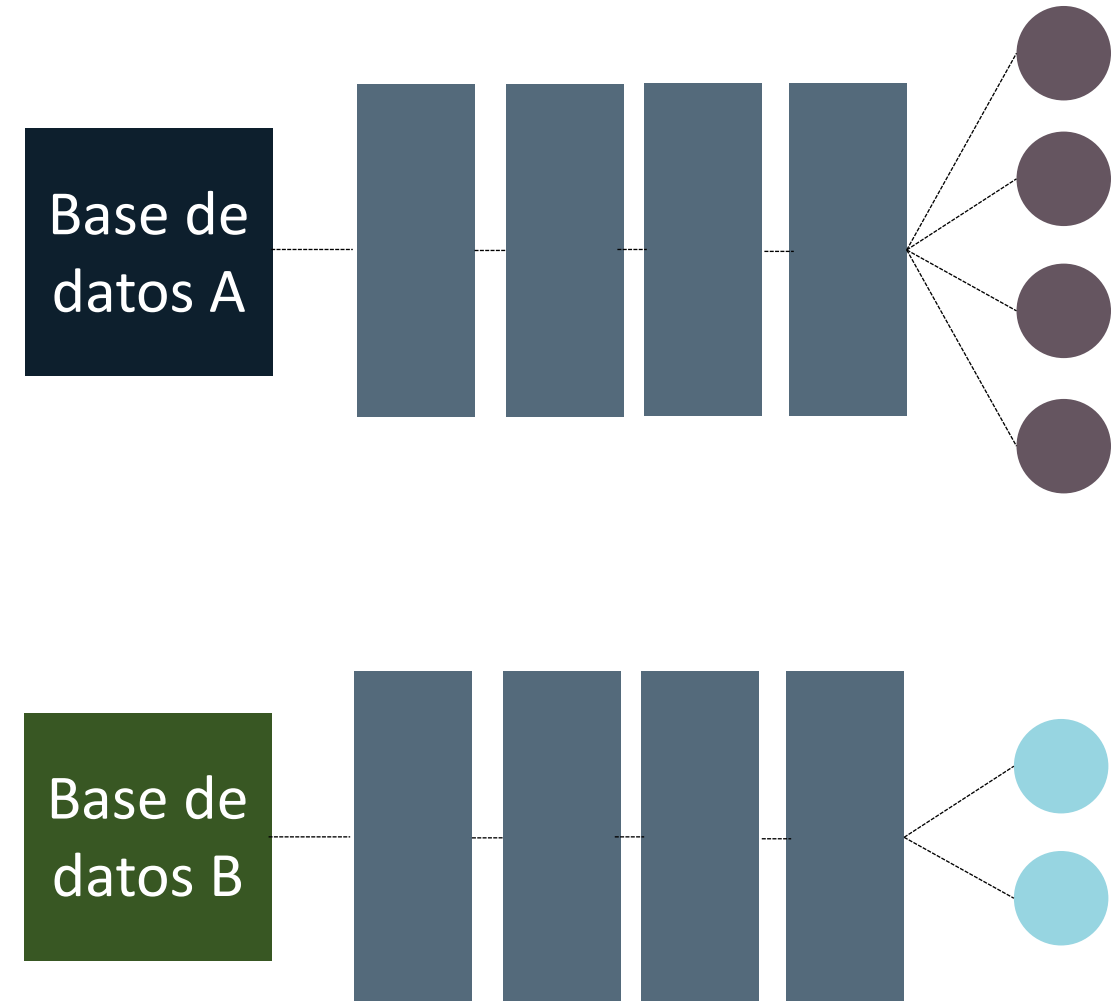
La transferencia de conocimiento (*transfer learning*) es la técnica según la cual un modelo entrenado para una tarea se emplea como **punto de partida** en la resolución de un problema diferente. Es una técnica de gran utilidad cuando no se cuenta de una gran base de datos de entrenamiento.



# Transferencia de conocimiento

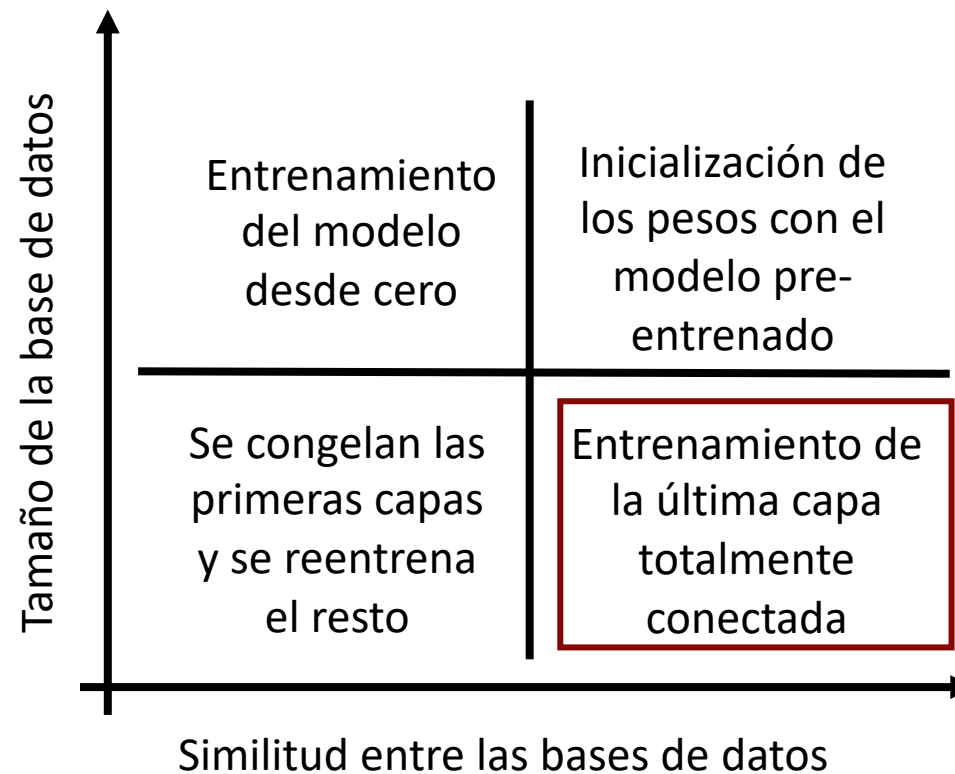
## Nueva base de datos grande y similar a la original

1. Se **elimina** la última capa totalmente conectada y se **sustituye** por una que tenga tantos nodos como clases tenga el nuevo problema.
2. Inicialización **aleatoria** de la nueva capa totalmente conectada.
3. Inicialización del resto de capas con los pesos del **modelo pre-entrenado**.
4. **Re-entrenamiento** de la CNN.



# Transferencia de conocimiento

La transferencia de conocimiento (*transfer learning*) es la técnica según la cual un modelo entrenado para una tarea se emplea como **punto de partida** en la resolución de un problema diferente. Es una técnica de gran utilidad cuando no se cuenta de una gran base de datos de entrenamiento.

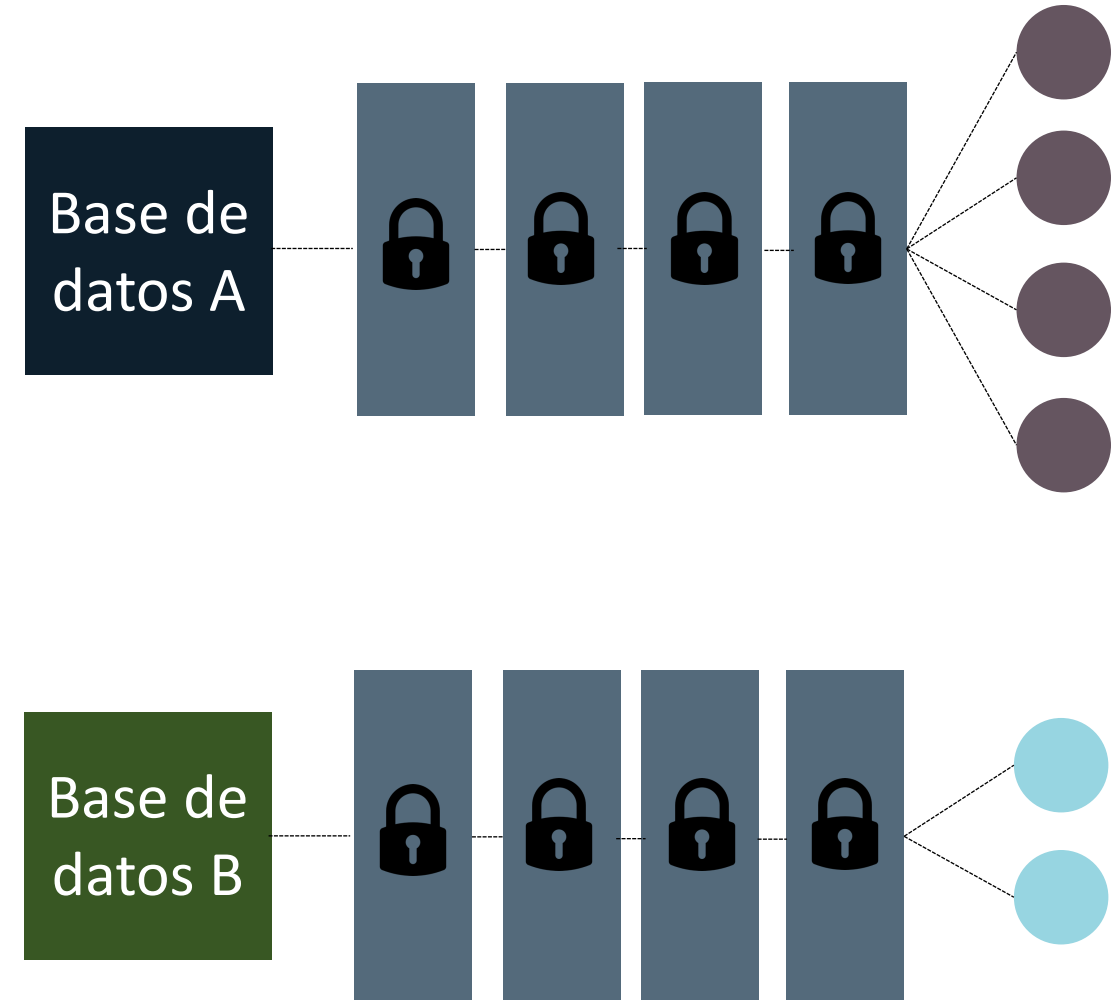


# Transferencia de conocimiento

## Nueva base de datos pequeña y similar a la original

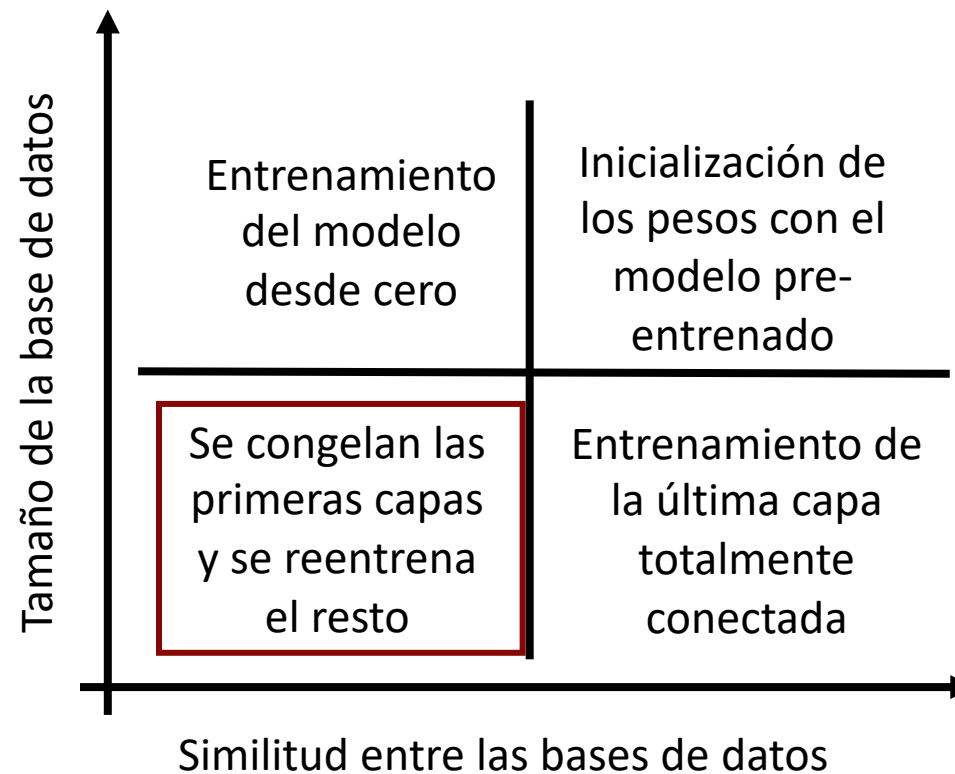
1. Se **eliminan** las capas totalmente conectadas y se **sustituye** por una que tenga tantos nodos como clases tenga el nuevo problema.
2. Inicialización **aleatoria** de la nueva capa totalmente conectada.
3. **Bloqueo** del valor de los pesos del resto de capas.
4. **Re-entrenamiento** de la nueva capa totalmente conectada.

## CNN como extractor de características



# Transferencia de conocimiento

La transferencia de conocimiento (*transfer learning*) es la técnica según la cual un modelo entrenado para una tarea se emplea como **punto de partida** en la resolución de un problema diferente. Es una técnica de gran utilidad cuando no se cuenta de una gran base de datos de entrenamiento.

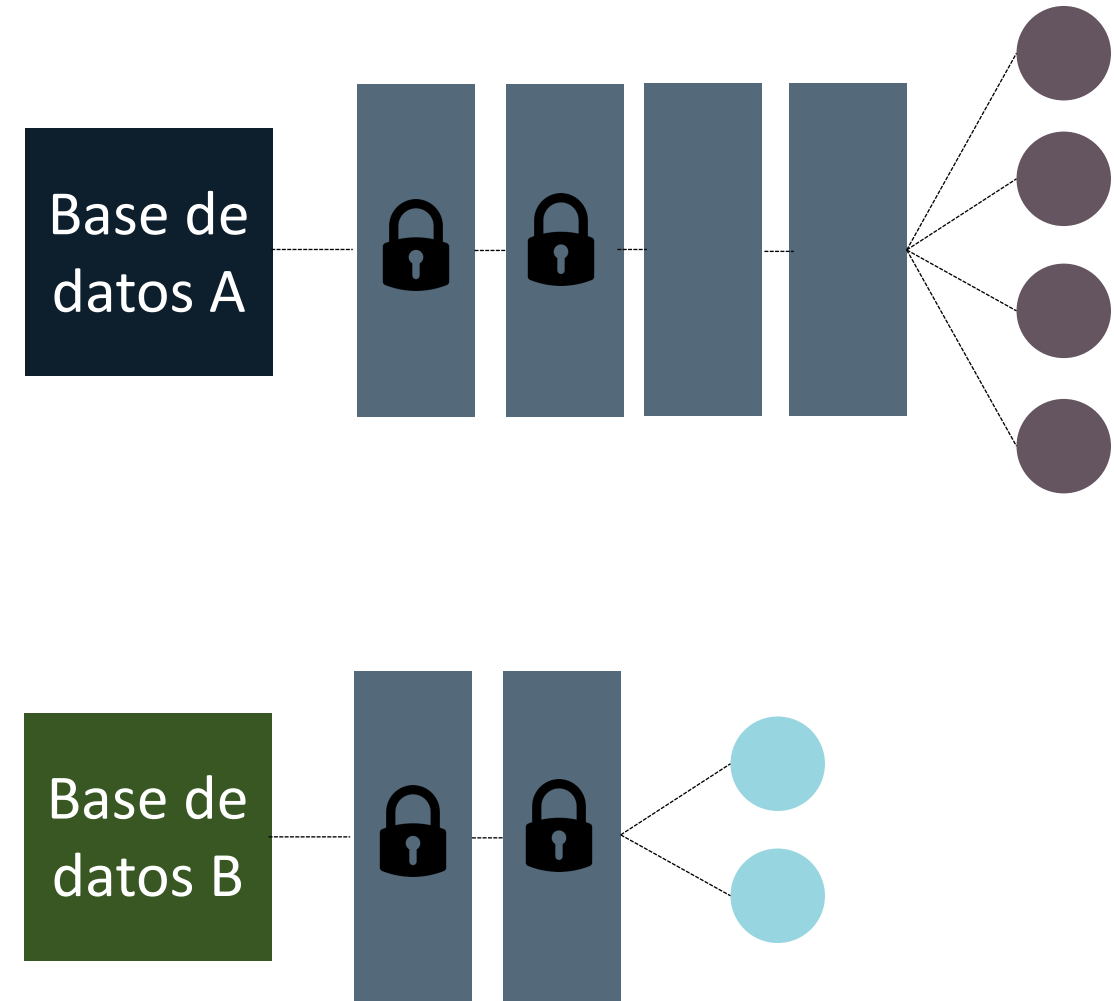




# Transferencia de conocimiento

## Nueva base de datos pequeña y diferente de la original

1. Se dejan únicamente las **primeras capas convolucionales** (características de bajo nivel) y se eliminan el resto.
2. Se **añaden** capas totalmente conectadas a las capas anteriores.
3. Se **mantienen** los pesos de las capas convolucionales.
4. Se inicializan **aleatoriamente** los pesos de las capas totalmente conectadas.
5. **Re-entrenamiento** de las capas totalmente conectadas.



# Transferencia de conocimiento

[www.image-net.org](http://www.image-net.org)



**14 M** imágenes organizadas en **22K** clases

# Transferencia de conocimiento



## Large Scale Visual Recognition Challenge

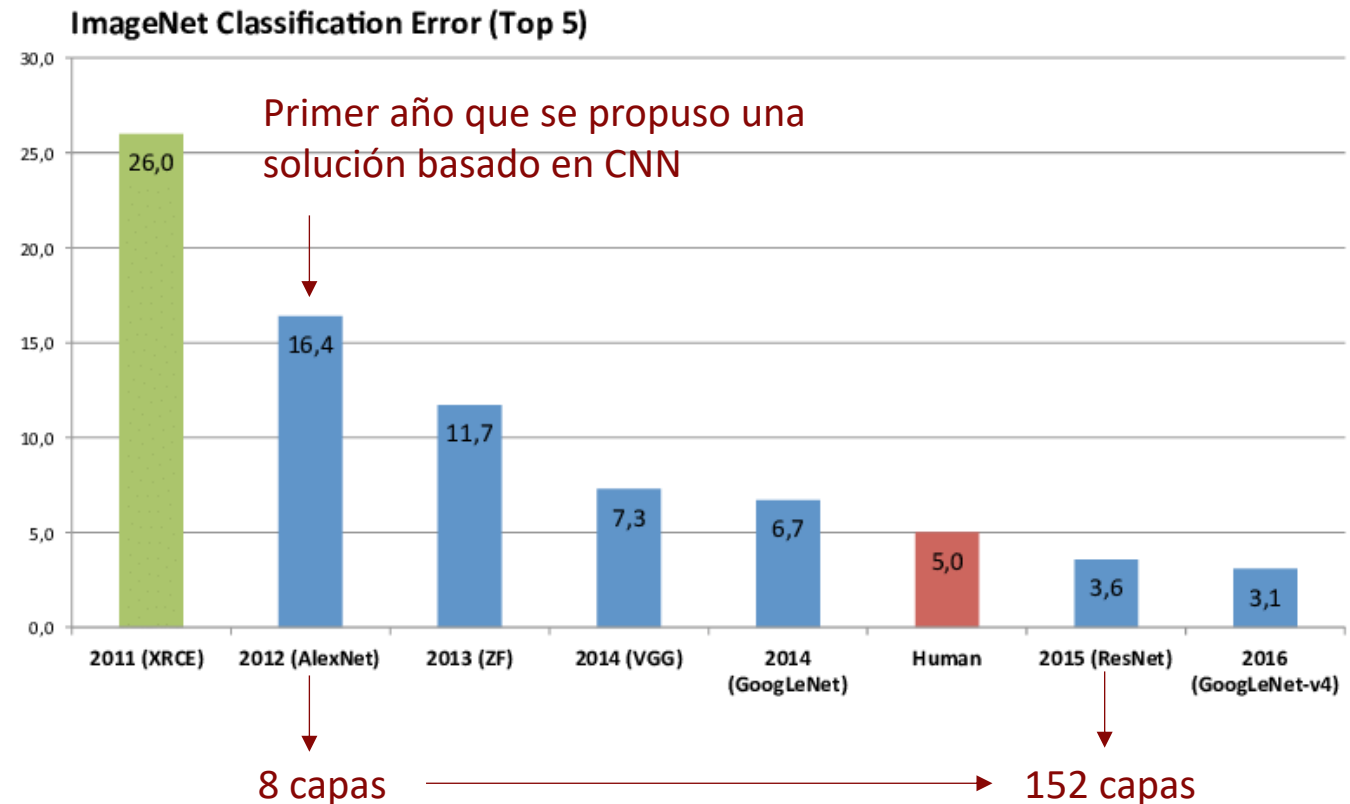


GPU



TPU

- Reto anual (desde 2009 a 2017) de clasificación de imágenes y detección de objetos.
- 1.28M de imágenes de entrenamiento.
- 50k imágenes de validación.
- 1k clases.
- Una imagen se da como correctamente clasificada si el algoritmo predice la etiqueta correcta entre las 5 de mayor probabilidad.



# Ejemplo práctico

# Transferencia de conocimiento

Vamos a emplear la misma base de datos que en el tema anterior: clasificación de imágenes de perros y gatos.  
Descargamos y preparamos la base de datos:

# Transferencia de conocimiento

```
# Descargamos la base de datos
_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'
path_to_zip = tf.keras.utils.get_file('cats_and_dogs.zip', origin=_URL, extract=True)

# Directorios con las imágenes de entrenamiento y validación
path_images = os.path.join(os.path.dirname(path_to_zip), 'cats_and_dogs_filtered')
path_train = os.path.join(path_images, 'train')
path_validation = os.path.join(path_images, 'validation')

# Vamos a emplear el método image_dataset_from_directory de tensorflow para crear los datasets
BATCH_SIZE = 32
IMG_SIZE = (160, 160)

train_dataset = image_dataset_from_directory(path_train,
                                             shuffle=True,
                                             batch_size=BATCH_SIZE,
                                             image_size=IMG_SIZE)

validation_dataset = image_dataset_from_directory(path_validation,
                                                  shuffle=True,
                                                  batch_size=BATCH_SIZE,
                                                  image_size=IMG_SIZE)

# Evitamos bloqueo E/S
AUTOTUNE = tf.data.AUTOTUNE
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
```



# Transferencia de conocimiento

## Ejercicio 1

Para, posteriormente, ver la comparativa de los resultados obtenidos al entrenar nosotros un modelo sencillo frente aplicar transferencia de conocimiento, vamos a implementar dicho modelo sencillo. Para ello sigue los siguientes pasos:

1. Aplica normalización (escalado entre 0-1) a la base de datos. No sobrescribas las variables “train\_dataset”, “validation\_dataset” y “test\_dataset” ya que las necesitaremos después, crea unas nuevas (e.g., train\_dataset\_norm, validation\_dataset\_norm, test\_dataset\_norm).
2. Implementa la arquitectura que se corresponda al summary de la siguiente figura donde la capa de salida hace uso de una función de activación sigmoide. Tras cada capa convolucional emplea una capa de normalización de lote “BatchNormalization”, de la forma: **`x = layers.BatchNormalization()(x)`** (esto le va a dar más estabilidad al entrenamiento).
3. Genera la carpeta donde vamos a almacenar los resultados de los experimentos.
4. Compila el modelo haciendo uso del optimizador Adam, función de activación entropía cruzada binaria, y sacando la exactitud (accuracy) como métrica.
5. Entrena el modelo durante 15 épocas haciendo uso del callback ModelCheckpoint de forma que se almacene el modelo correspondiente a la época en la que mejores métricas en validación se obtienen.
6. Visualiza las curvas de aprendizaje mostrando en una misma gráfica las métricas de entrenamiento y validación. ¿Hay sobreajuste?

## Ejercicio 1

```
Model: "model"
Layer (type)                 Output Shape                 Param #
-----
input_1 (InputLayer)         [(None, 160, 160, 3)]      0
conv2d (Conv2D)              (None, 160, 160, 128)      3584
batch_normalization (BatchN  (None, 160, 160, 128)      512
ormalization)
conv2d_1 (Conv2D)            (None, 160, 160, 128)      147584
batch_normalization_1 (Batc  (None, 160, 160, 128)      512
hNormalization)
max_pooling2d (MaxPooling2D  (None, 80, 80, 128)        0
)
conv2d_2 (Conv2D)            (None, 80, 80, 256)        295168
batch_normalization_2 (Batc  (None, 80, 80, 256)        1024
hNormalization)
conv2d_3 (Conv2D)            (None, 80, 80, 256)        590080
batch_normalization_3 (Batc  (None, 80, 80, 256)        1024
hNormalization)
max_pooling2d_1 (MaxPooling  (None, 40, 40, 256)        0
2D)
conv2d_4 (Conv2D)            (None, 40, 40, 512)        1180160
batch_normalization_4 (Batc  (None, 40, 40, 512)        2048
hNormalization)
conv2d_5 (Conv2D)            (None, 40, 40, 512)        2359808
batch_normalization_5 (Batc  (None, 40, 40, 512)        2048
hNormalization)
global_average_pooling2d (G  (None, 512)                0
lobalAveragePooling2D)
dense (Dense)                (None, 128)                65664
dense_1 (Dense)              (None, 1)                  129
=====
Total params: 4,649,345
Trainable params: 4,645,761
Non-trainable params: 3,584
```



# Transferencia de conocimiento

La transferencia de conocimiento se aplica, generalmente, en dos pasos:

1. **Extracción de características:** se emplea un modelo preentrenado al que se le sustituye la última capa densa (encargada de la clasificación final) por una nueva capa densa que se ajuste al problema a resolver. A continuación, se congelan todas las capas excepto la nueva capa densa, y se realiza el entrenamiento de dicha capa.
2. **Ajuste fino:** se descongelan las capas convolucionales encargadas de la extracción de características y se entrena toda la CNN durante unas pocas épocas para adaptar las características al nuevo problema.

# Transferencia de conocimiento

## 1. Extracción de características

Vamos a entrenar un modelo siguiendo una estrategia de transferencia de conocimiento empleando un modelo preentrenado como extractor de características, es decir, se van a **congelar todas las capas a excepción de la capa final de clasificación** que se sustituirá por un nuevo clasificador ajustado a nuestro problema.

Como arquitectura base vamos a emplear **MobileNet V2** (desarrollado por Google). Este modelo está preentrenado en ImageNet.

Dentro de la API de keras podemos encontrar el modelo preentrenado. Con el argumento **"include\_top"** a "False" le estamos indicando que no se incluya la capa correspondiente a la clasificación. Con el argumento **"weights"** le estamos indicando que inicialice la arquitectura con los pesos que se obtuvieron al entrenarla con ImageNet.

```
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                                include_top=False,
                                                weights='imagenet')

base_model.summary()
```

# Transferencia de conocimiento

## 1. Extracción de características

De todo el “summary” nos vamos a fijar en el final:

```
Conv_1 (Conv2D)                (None, 5, 5, 1280)  409600    ['block_16_project_BN[0][0]']
Conv_1_bn (BatchNormalization) (None, 5, 5, 1280)  5120      ['Conv_1[0][0]']
out_relu (ReLU)                (None, 5, 5, 1280)  0         ['Conv_1_bn[0][0]']

=====
Total params: 2,257,984
Trainable params: 2,223,872
Non-trainable params: 34,112
```

Esta arquitectura convierte una imagen de 160x160x3 en un bloque de características de 5x5x1280.

# Transferencia de conocimiento

## 1. Extracción de características

Para emplear el modelo anterior como extractor de características, tenemos que **congelar todo el bloque convolucional**, para ello, se establecen todas las capas pertenecientes al bloque convolucional (todas las que hemos cargado anteriormente en `base_model`) como **no entrenables**.

Esto se podría realizar capa a capa si únicamente se quisieran congelar algunas en concreto, pero dado que queremos congelar todo el bloque podemos hacerlo directamente como:

```
base_model.trainable = False
```

# Transferencia de conocimiento

## 1. Extracción de características

El modelo **MobileNetV2** se entrenó empleando un método de normalización basado en el **reescalado de los valores de la imagen en -1 y 1**. Podemos emplear el método incluido en la propia **API de keras** para preprocesar la imagen de la misma forma que se realizó originalmente. Lo siguiente implementa una capa que se encarga de aplicar el preprocesado a la imagen, por lo tanto, se incluirá como una capa más de la arquitectura.

```
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
```

# Transferencia de conocimiento

## 1. Extracción de características

Finalmente, vamos a implementar la arquitectura. A nuestro modelo base le vamos a añadir la capa final de clasificación.

Para convertir la matriz a vector vamos a hacer uso de la capa **GlobalAveragePooling**.

```
input = layers.Input(shape=(160, 160, 3))
# Capa preprocesado
x = preprocess_input(input)
# Añadimos modelo base
x = base_model(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
# Clasificación binaria
output = layers.Dense(1, activation="sigmoid")(x)

model_tl = Model(inputs=[input], outputs=[output])
model_tl.summary()
```

# Transferencia de conocimiento

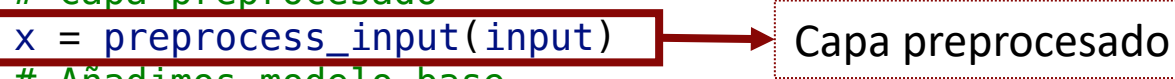
## 1. Extracción de características

Finalmente, vamos a implementar la arquitectura. A nuestro modelo base le vamos a añadir la capa final de clasificación.

Para convertir la matriz a vector vamos a hacer uso de la capa **GlobalAveragePooling**.

```
input = layers.Input(shape=(160, 160, 3))
# Capa preprocesado
x = preprocess_input(input)
# Añadimos modelo base
x = base_model(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
# Clasificación binaria
output = layers.Dense(1, activation="sigmoid")(x)

model_tl = Model(inputs=[input], outputs=[output])
model_tl.summary()
```



# Transferencia de conocimiento

## 1. Extracción de características

Model: "model\_1"

Layer (type)	Output Shape	Param #
=====		
input_4 (InputLayer)	[(None, 160, 160, 3)]	0
tf.math.truediv_1 (TFOpLambda)	(None, 160, 160, 3)	0
tf.math.subtract_1 (TFOpLambda)	(None, 160, 160, 3)	0
mobilenetv2_1.00_160 (Functional)	(None, 5, 5, 1280)	2257984
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 1280)	0
dense_2 (Dense)	(None, 1)	1281
=====		
Total params: 2,259,265		
Trainable params: 1,281		
Non-trainable params: 2,257,984		

Preprocesado de la imagen



# Transferencia de conocimiento

## 1. Extracción de características

Model: "model\_1"

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[ (None, 160, 160, 3) ]	0
tf.math.truediv_1 (TFOpLambda)	(None, 160, 160, 3)	0
tf.math.subtract_1 (TFOpLambda)	(None, 160, 160, 3)	0
mobilenetv2_1.00_160 (Functional)	(None, 5, 5, 1280)	2257984
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 1280)	0
dense_2 (Dense)	(None, 1)	1281
=====		
Total params: 2,259,265		
Trainable params: 1,281		
Non-trainable params: 2,257,984		

Correspondientes a la capa de “mobilenetv2” (modelo preentrenado) no se va a entrenar ningún parámetro. Todos los que se van a entrenar provienen de la capa totalmente conectada de clasificación.

# Transferencia de conocimiento

## 1. Extracción de características

Compilamos y entrenamos modelo:

```
# Compilamos
model_tl.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy'])

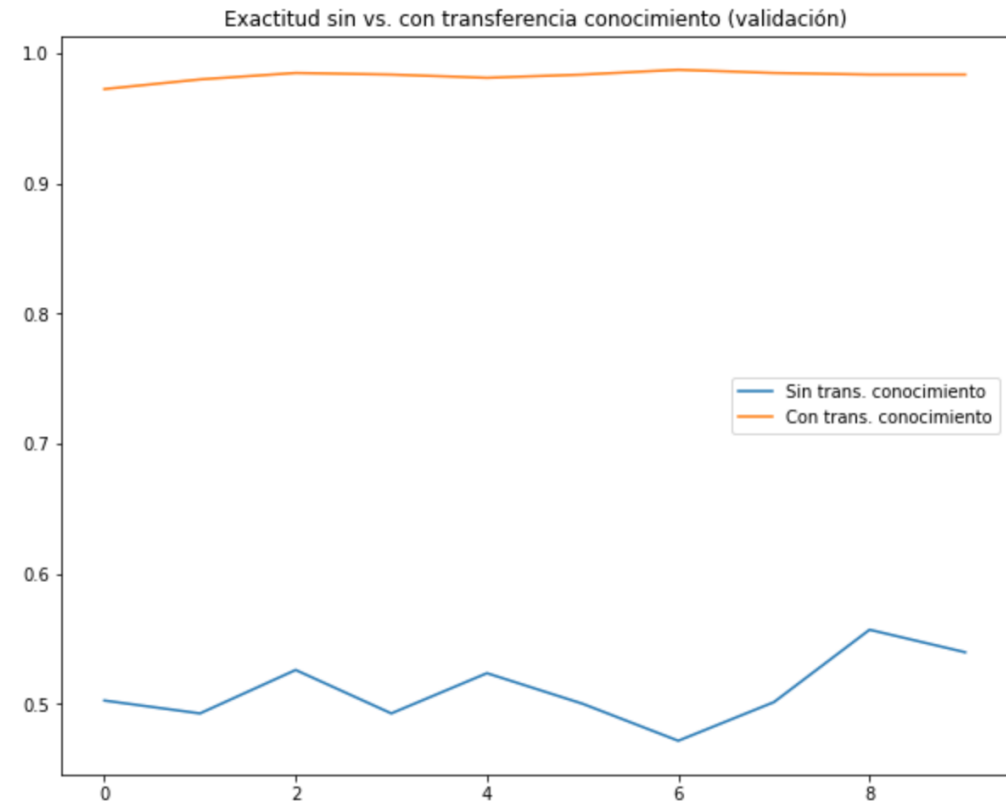
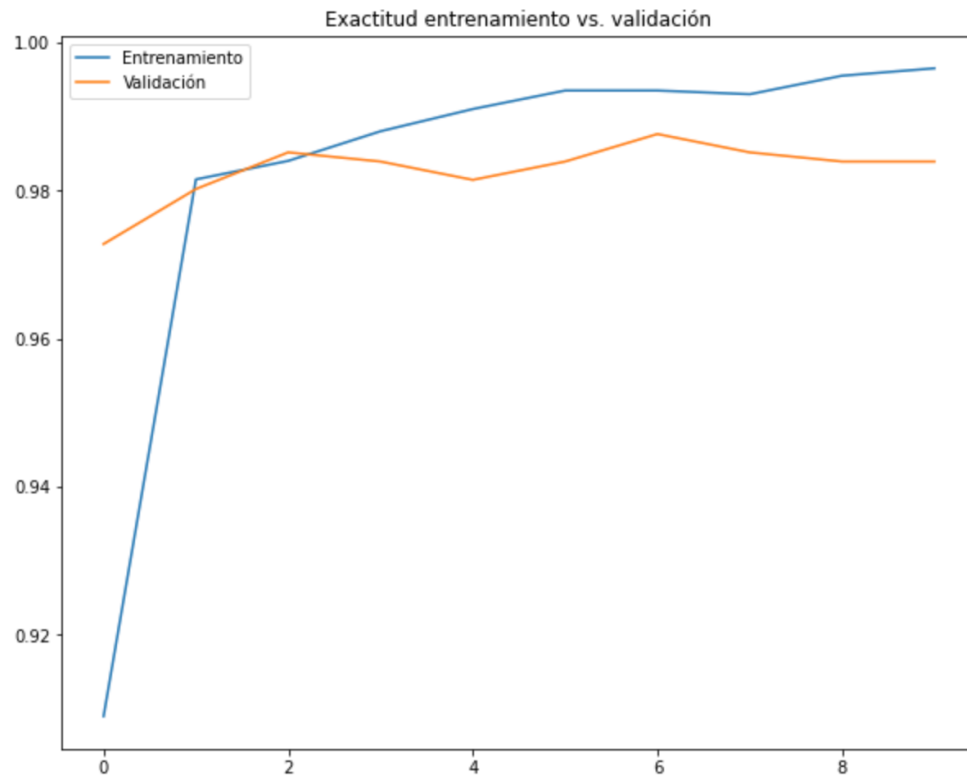
# Inicializamos el callback
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=os.path.join(path_experiment, 'dogs_and_cats.h5'),
    monitor='val_accuracy',
    mode='max',
    save_best_only=True,
    verbose=1)

# Entrenamos
history_tl = model_tl.fit(train_dataset,
                           epochs=10,
                           validation_data=validation_dataset,
                           callbacks=[model_checkpoint_callback],
                           verbose=1)
```

# Transferencia de conocimiento

## 1. Extracción de características

Evaluamos curvas aprendizaje:



# Transferencia de conocimiento

## 2. Ajuste fino

En el experimento anterior no se han entrenado las capas pertenecientes al modelo preentrenado. Una forma de aumentar todavía más el rendimiento del modelo es realizar un ajuste (fino) de los pesos de estas capas. De este modo, **ajustaremos los filtros de las capas convolucionales** a extraer mapas de características apropiados para la resolución del nuevo problema. Este paso se recomienda incluirlo cuando el conjunto de datos de entrenamiento es **grande y muy similar** al conjunto de datos original en el que se entrenó el modelo preentrenado.

**NOTA IMPORTANTE:** Este ajuste fino debe de realizarse **tras al ajuste de la capa de clasificación** con las capas preentrenadas congeladas. Si se agrega el clasificador inicializado aleatoriamente y no se congelan las capas preentrenadas (se ajustan todos los pesos de la red), la magnitud de las actualizaciones de gradiente será demasiado grande (debido a los pesos aleatorios del clasificador) y el modelo entrenado previamente **olvidará** lo que ya ha aprendido.

# Transferencia de conocimiento

## 2. Ajuste fino

No se va a realizar el ajuste de todas las capas preentrenadas, únicamente aquellas de **más alto nivel** que son las que están más especializadas en el problema para el que fueron entrenadas. Las primeras capas aprenden características más simples y genéricas que generalizan a casi todos los tipos de imágenes. De este modo, vamos a **descongelar** únicamente las **capas superiores**.

```
# Vemos el número de capas que tiene el modelo base
print("Número de capas en el modelo base: ", len(base_model.layers))
```

Salida:

Número de capas en el modelo base: 154

Vamos a ajustar las capas desde la capa 100

```
# Volvemos a descongelar el modelo
base_model.trainable = True

# Como ahora solo queremos entrenar capas específicas, vamos a congelar las capas concretas que
# no queremos reentrenar (hasta la 100)
for layer in base_model.layers[:100]:
    layer.trainable = False
```

# Transferencia de conocimiento

## 2. Ajuste fino

**IMPORTANTE:** Como solo queremos readaptar unos pesos ya entrenados, se va a emplear una **tasa de aprendizaje pequeña** para evitar el sobreajuste.

```
model_tl.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5),  
                 loss='binary_crossentropy',  
                 metrics=['accuracy'])  
  
model_tl.summary()
```

# Transferencia de conocimiento

## 2. Ajuste fino

**IMPORTANTE:** Como solo queremos readaptar unos pesos ya entrenados, se va a emplear una **tasa de aprendizaje pequeña** para evitar el sobreajuste.

```
model_tl.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5),  
                 loss='binary_crossentropy',  
                 metrics=['accuracy'])  
  
model_tl.summary()
```

Trabajamos sobre el mismo modelo,  
el cual vamos a seguir entrenando

# Transferencia de conocimiento

## 2. Ajuste fino

Model: "model\_1"

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 160, 160, 3)]	0
tf.math.truediv_1 (TFOpLamb da)	(None, 160, 160, 3)	0
tf.math.subtract_1 (TFOpLam bda)	(None, 160, 160, 3)	0
mobilenetv2_1.00_160 (Func tional)	(None, 5, 5, 1280)	2257984
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 1280)	0
dense_2 (Dense)	(None, 1)	1281

=====  
Total params: 2,259,265  
Trainable params: 1,862,721  
Non-trainable params: 396,544

Ahora vamos a ajustar algunos de los parámetros de la red preentrenada (mobilenetv2)



# Transferencia de conocimiento

## 2. Ajuste fino

Entrenamos durante 5 épocas más.

```
history_fine = model_tl.fit(train_dataset,  
                             epochs=5,  
                             validation_data=validation_dataset,  
                             callbacks=[model_checkpoint_callback],  
                             verbose=1)
```

Mismo callback para que  
tenga en cuenta la  
historia anterior

# Transferencia de conocimiento

## 2. Ajuste fino

Curvas aprendizaje

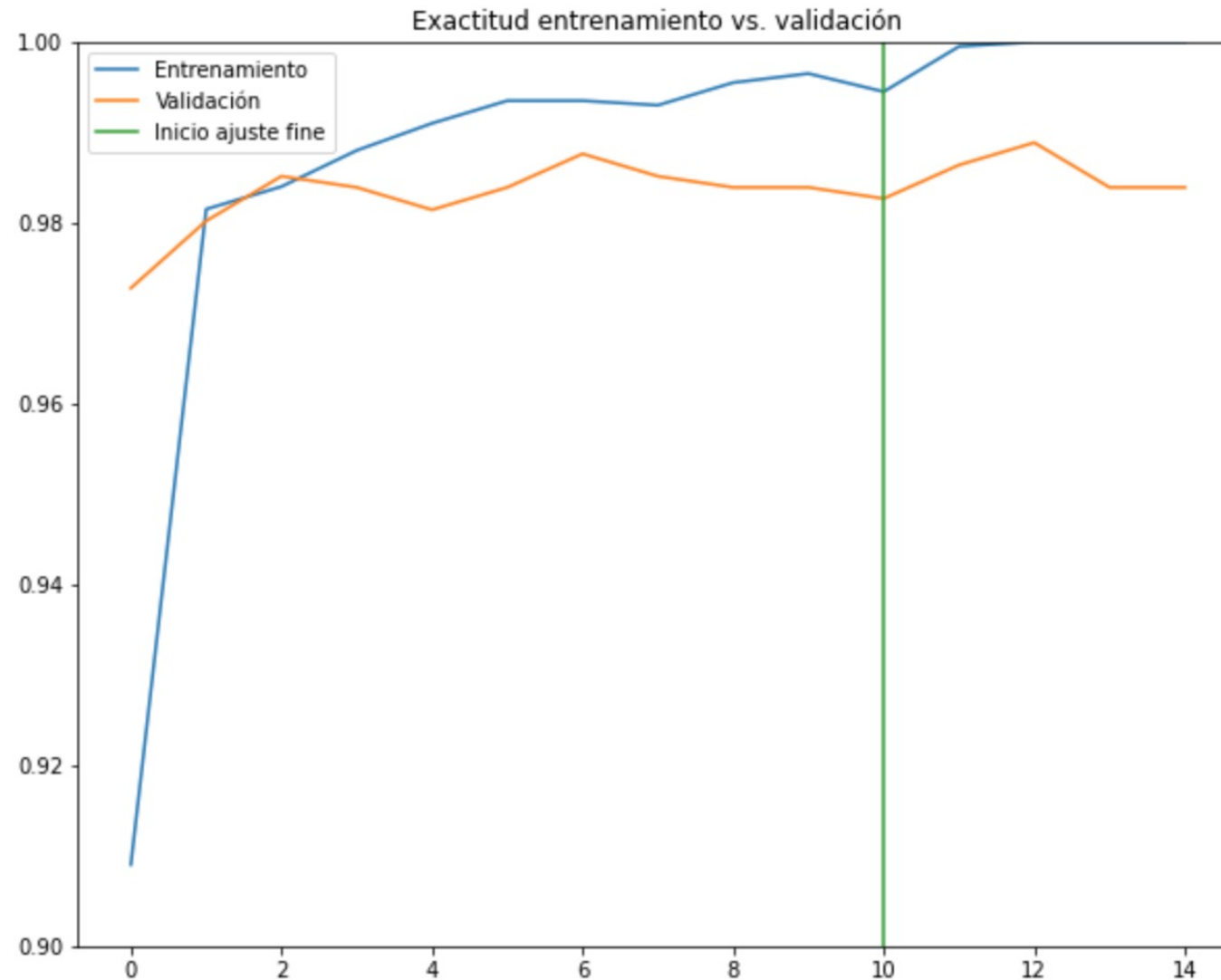
```
# Analizamos conjuntamente la precisión en entrenamiento y validación al congelar todas las
# capas y tras descongelar algunas de ellas

acc = history_tl.history['accuracy'] + history_fine.history['accuracy']
val_acc = history_tl.history['val_accuracy'] + history_fine.history['val_accuracy']

plt.plot(acc, label='Entrenamiento')
plt.plot(val_acc, label='Validación')
plt.ylim([0.9, 1])
plt.plot([10, 10],
plt.ylim(), label='Inicio ajuste fine')
plt.legend()
```

# Transferencia de conocimiento

## 2. Ajuste fino



# Transferencia de conocimiento

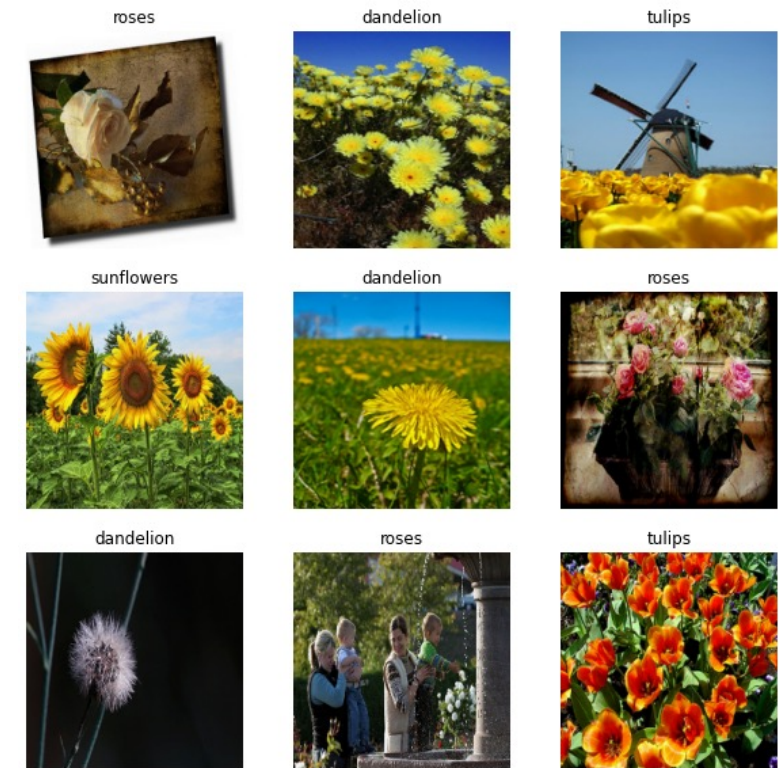
## Práctica con transferencia de conocimiento

*(Práctica. Clasificación flores)*

Ahora vais a implementar vosotros un modelo de clasificación con transferencia de conocimiento sobre una nueva base de datos.

Se trata de una base de datos con imágenes de flores pertenecientes a 5 clases diferentes. Vamos a emplear el modelo preentrenado Inception V3 para resolver este problema de clasificación.

Primero vamos a ver como cargar la base de datos:

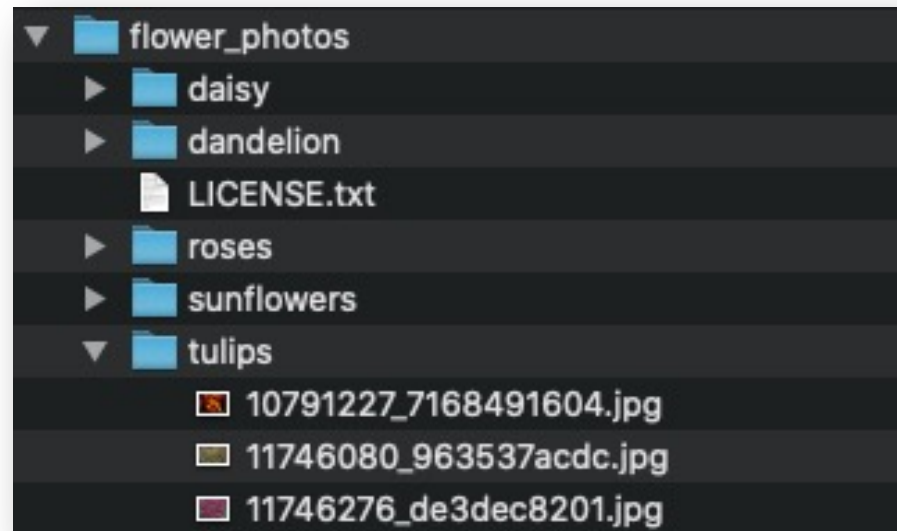


# Transferencia de conocimiento

## Práctica con transferencia de conocimiento

```
data_root = tf.keras.utils.get_file(  
    'flower_photos',  
    'https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz',  
    untar=True)
```

En este caso se trata de una base de datos en la que todas las imágenes se encuentran en una misma carpeta organizadas en subcarpetas pertenecientes a cada una de las clases.



# Transferencia de conocimiento

## Práctica con transferencia de conocimiento

```
IMG_SIZE = (224, 224)
BATCH_SIZE = 32

train_dataset = tf.keras.utils.image_dataset_from_directory(
    data_root,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE
)

validation_dataset = tf.keras.utils.image_dataset_from_directory(
    data_root,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE
)
```

# Transferencia de conocimiento

## Función de pérdidas en problemas de clasificación

### Entropía cruzada

En **Keras** existen 3 formas diferentes de implementar la entropía cruzada como función de pérdidas. En función del caso concreto emplearemos una u otra:

- Problemas de clasificación binaria (2 clases):
  - **Entropía cruzada binaria** (*binary\_crossentropy*). A la salida tenemos una capa totalmente conectada con 1 neurona.
- Problema de clasificación multiclase (>2 clases):
  - **Entropía cruzada categórica** (*categorical\_crossentropy*): Las clases se introducen al modelo en codificación One Hot.
  - **Entropía cruzada categórica dispersa** (*sparse\_categorical\_crossentropy*): Las clases se introducen como números enteros.
    - Por defecto tiene el argumento “from\_logits” a False, por lo que es necesario añadir la función de activación “softmax” en la última capa totalmente conectada.
    - Podemos especificar from\_logits=True (haciendo uso de la clase: `tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)`) pero, entonces, no hay que añadir función de activación a la capa de salida.

# **Tema 4. Transferencia de conocimiento**

Ana Jiménez Pastor  
anjipas@gmail.com