TL09 KerasCV

Índice

1 Introducción

KerasCV: librería de visión por ordenador

Documentación: https://keras.io/keras_cv

Github: https://github.com/keras-team/keras-cv

Guías: https://keras.io/guides/keras_cv

API: https://keras.io/api/keras_cv

- **Layers:** capas de aumento de datos, preproceso y regularización
- **Models:** modelos backbone (preset y custom) para diversas tareas
- **Bounding box formats and utilities**: formatos y utilidades para cajas
- **Losses:** pérdidas específicas para tareas CV

2 API

In [ ]:
```
import keras_cv
```

2.1 Layers

Documentación: https://keras.io/api/keras_cv/layers

Augmentation layers: https://keras.io/api/keras_cv/layers/augmentation

Preprocessing layers: https://keras.io/api/keras_cv/layers/preprocessing

Regularization layers: https://keras.io/api/keras_cv/layers/regularization

2.2 Models

Documentación: https://keras.io/api/keras_cv/models

Tareas: https://keras.io/api/keras_cv/models/tasks

- **BASNet Segmentation:** https://keras.io/api/keras_cv/models/tasks/basnet_segmentation
- **DeepLabV3Plus Segmentation:** https://keras.io/api/keras_cv/models/tasks/deeplab_v3_segmentation
- **SegFormer Segmentation:** https://keras.io/api/keras_cv/models/tasks/segformer_segmentation
- **Segment Anything:** https://keras.io/api/keras_cv/models/tasks/segment_anything
- **CLIP Feature extractor:** https://keras.io/api/keras_cv/models/tasks/feature_extractor
- **ImageClassifier:** https://keras.io/api/keras_cv/models/tasks/image_classifier
- **RetinaNet:** https://keras.io/api/keras_cv/models/tasks/retinanet
- **StableDiffusion image generation:** https://keras.io/api/keras_cv/models/tasks/stable_diffusion
- **YOLOV8Detector:** https://keras.io/api/keras_cv/models/tasks/yolo_v8_detector

Backbones: https://keras.io/api/keras_cv/models/backbones

- **CSPDarkNet:** https://keras.io/api/keras_cv/models/backbones/csp_darknet
- **DenseNet:** https://keras.io/api/keras_cv/models/backbones/densenet
- **EfficientNetV1:** https://keras.io/api/keras_cv/models/backbones/efficientnet_v1
- **EfficientNetV2:** https://keras.io/api/keras_cv/models/backbones/efficientnet_v2
- **EfficientNet Lite:** https://keras.io/api/keras_cv/models/backbones/efficientnet_lite
- **MixTransformer:** https://keras.io/api/keras_cv/models/backbones/mix_transformer
- **MobileNetV3:** https://keras.io/api/keras_cv/models/backbones/mobilenet_v3
- **ResNetV1:** https://keras.io/api/keras_cv/models/backbones/resnet_v1
- **ResNetV2:** https://keras.io/api/keras_cv/models/backbones/resnet_v2
- **VGG16:** https://keras.io/api/keras_cv/models/backbones/vgg16
- **ViTDet:** https://keras.io/api/keras_cv/models/backbones/vitdet
- **YOLOV8:** https://keras.io/api/keras_cv/models/backbones/yolo_v8

2.3 Bounding box formats and utilities

Documentación: https://keras.io/api/keras_cv/bounding_box

Bounding box formats: https://keras.io/api/keras_cv/bounding_box/formats

- CENTER_XYWH
- XYWH
- REL_XYWH
- XYXY
- REL_XYXY
- YXYX
- REL_YXYX

Bounding box utilities: https://keras.io/api/keras_cv/bounding_box/utils

- Convert bounding box formats
- Compute intersection over union of bounding boxes
- Clip bounding boxes to be within the bounds of provided images
- Convert a bounding box dictionary to -1 padded Dense tensors
- Convert a bounding box dictionary batched Ragged tensors
- Ensure that your bounding boxes comply with the bounding box spec

2.4 Losses

Documentación: https://keras.io/api/keras_cv/losses

Focal: variante de entropía cruzada binaria

Binary Penalty Reduced Focal CrossEntropy: variante de entropía cruzada binaria

IoU: Intersection over Union

GIoU: Generalized IoU; modificación de IoU

CIoU: Complete IoU; extensión de GIoU

SimCLR: SimCLR Cosine Similarity para aprendizaje auto-supervisado contrastivo

SmoothL1Loss: función SmoothL1

3 TensorFlow datasets

TensorFlow datasets (TFDS): colección de conjuntos de datos

Web: https://www.tensorflow.org/datasets

Github: https://github.com/tensorflow/datasets

Librerías: tensorflow_datasets es una librería separada de tensorflow

In [ ]:
```
import numpy as np; import matplotlib.pyplot as plt; import os; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow as tf; import tensorflow_datasets as tfds
```

3.1 Datasets disponibles

Catálogo web: https://www.tensorflow.org/datasets/catalog/overview

`tfds.core.DatasetBuilder` : constructores de datasets

`tfds.list_builders()` : proporciona un listado

In [ ]:
```
list = tfds.list_builders()
print(f'Primeros 100 de {len(list)}:', list[:100])
```

```
Primeros 100 de 1291: ['abstract_reasoning', 'accentdb', 'aeslc', 'aflw2k3d', 'ag_news_subset', 'ai2_arc', 'ai2_arc_with_ir', 'aloha_mobile', 'amazon_us_reviews', 'anli', 'answer_equivalence', 'arc', 'asqa', 'asset', 'assin2', 'asu_table_top_converted_externally_to_rlds', 'austin_buds_dataset_converted_externally_to_rlds', 'austin_sailor_dataset_converted_externally_to_rlds', 'austin_sirius_dataset_converted_externally_to_rlds', 'bair_robot_pushing_small', 'bc_z', 'bccd', 'beans', 'bee_dataset', 'beir', 'berkeley_autolab_ur5', 'berkeley_cable_routing', 'berkeley_fanuc_manipulation', 'berkeley_gnm_cory_hall', 'berkeley_gnm_recon', 'berkeley_gnm_sac_son', 'berkeley_mvp_converted_externally_to_rlds', 'berkeley_rpt_converted_externally_to_rlds', 'big_patent', 'bigearthnet', 'billsum', 'binarized_mnist', 'binary_alpha_digits', 'ble_wind_field', 'blimp', 'booksum', 'bool_q', 'bot_adversarial_dialogue', 'bridge', 'bridge_data_msr', 'bucc', 'c4', 'c4_wsrs', 'caltech101', 'caltech_birds2010', 'caltech_birds2011', 'cardiotox', 'cars196', 'cassava', 'cats_vs_dogs', 'celeb_a', 'celeb_a_hq', 'cfq', 'cherry_blossoms', 'chexpert', 'cifar10', 'cifar100', 'cifar100_n', 'cifar10_1', 'cifar10_corrupted', 'cifar10_h', 'cifar10_n', 'citrus_leaves', 'cityscapes', 'civil_comments', 'clevr', 'clic', 'clinc_oos', 'cmaterdb', 'cmu_franka_exploration_dataset_converted_externally_to_rlds', 'cmu_play_fusion', 'cmu_stretch', 'cnn_dailymail', 'coco', 'coco_captions', 'coil100', 'colorectal_histology', 'colorectal_histology_large', 'columbia_cairlab_pusht_real', 'common_voice', 'conll2002', 'conll2003', 'conq_hose_manipulation', 'controlled_noisy_web_labels', 'coqa', 'corr2cause', 'cos_e', 'cosmos_qa', 'covid19', 'covid19sum', 'crema_d', 'criteo', 'cs_restaurants', 'curated_breast_imaging_ddsm', 'cycle_gan']
```

3.2 Lectura de un dataset

`tfds.load`: descarga datos y los guarda como ficheros `tfrecord`; luego los lee y crea un `tf.data.Dataset`

```python
ds = tfds.load('mnist', split='train', shuffle_files=True)
assert isinstance(ds, tf.data.Dataset)
print(ds)
```

```
<_PrefetchDataset element_spec={'image': TensorSpec(shape=(28, 28, 1), dtype=tf.uint8, name=None), 'label': TensorSp
ec(shape=(), dtype=tf.int64, name=None)}>
```

- `split`: partición a leer; por ejemplo, `'train'`, `['train','test']`, `'train[80%:]'`, etc.
- `shuffle_files`: para barajar o no los datos cada nueva época
- `data_dir`: directorio donde guardar el dataset, `~/tensorflow_datasets` por omisión
- `with_info=True`: devuelve metadatos en `tfds.core.DatasetInfo`
- `download=False`: inhabilita descarga

`tfds.builder`: `tfds.load` es un wrapper; `tfds.core.DatasetBuilder` es la clase base para todos los datasets

```python
builder = tfds.builder('mnist')
builder.download_and_prepare()
ds = builder.as_dataset(split='train', shuffle_files=True)
print(ds)
```

```
<_PrefetchDataset element_spec={'image': TensorSpec(shape=(28, 28, 1), dtype=tf.uint8, name=None), 'label': TensorSp
ec(shape=(), dtype=tf.int64, name=None)}>
```

3.3 Iterar sobre un dataset

Como dict: por omisión, `tf.data.Dataset` contiene un diccionario de `tf.Tensor`

```
In [ ]: ds = tfds.load('mnist', split='train'); ds = ds.take(1)  # Only take a single example
        for example in ds:  # example is `{'image': tf.Tensor, 'label': tf.Tensor}`
          print(example.keys()); print(example["image"].shape, example["label"])
```

```
dict_keys(['image', 'label'])
(28, 28, 1) tf.Tensor(4, shape=(), dtype=int64)
```

Como tupla: con `as_supervised=True` se obtiene una tupla `(features, label)`

```
In [ ]: ds = tfds.load('mnist', split='train', as_supervised=True); ds = ds.take(1)
        for image, label in ds:  # example is (image, label)
          print(image.shape, label)
```

```
(28, 28, 1) tf.Tensor(4, shape=(), dtype=int64)
```

Como numpy: `tfds.as_numpy` convierte `tf.Tensor` en `np.array` y `tf.data.Dataset` en iterador

```
In [ ]: ds = tfds.load('mnist', split='train', as_supervised=True); ds = ds.take(1)
        for image, label in tfds.as_numpy(ds):
          print(type(image), type(label), label)
```

```
<class 'numpy.ndarray'> <class 'numpy.int64'> 4
```

Como batched tf.Tensor: con `batch_size=-1` se lee el dataset completo en un único batch

```
In [ ]: image, label = tfds.as_numpy(tfds.load('mnist', split='test', batch_size=-1, as_supervised=True))
        print(type(image), image.shape)
```

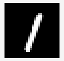
```
<class 'numpy.ndarray'> (10000, 28, 28, 1)
```

3.4 Visualización

`tfds.as_dataframe`: con `ds.take(num_ejemplos)` y `tfds.core.DatasetInfo` de argumentos

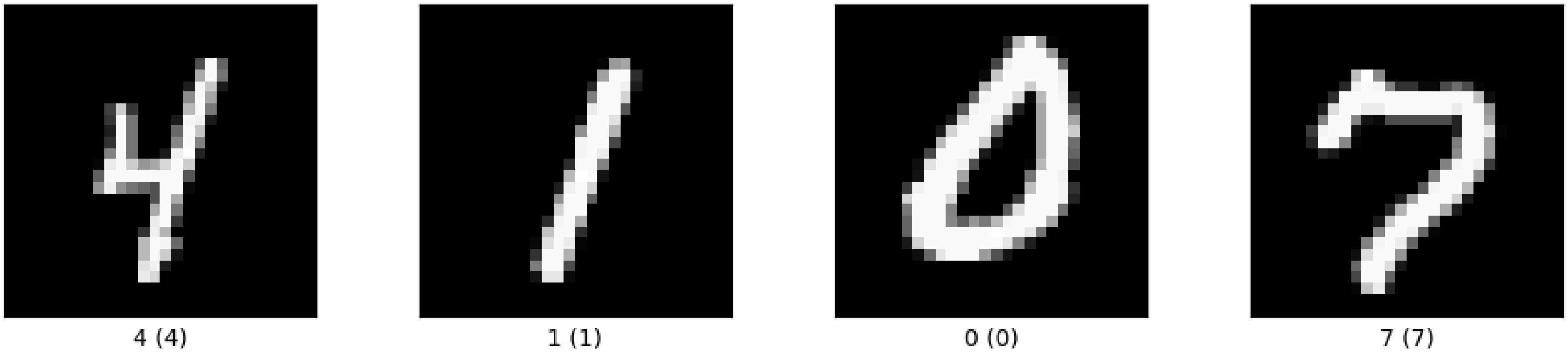
In [ ]:
```python
ds, info = tfds.load('mnist', split='train', with_info=True)
tfds.as_dataframe(ds.take(4), info)
```

Out[ ]:

	image	label
0		4
1		1
2		0
3		7

tfds.show_examples: devuelve una `matplotlib.figure.Figure` en el caso de imágenes

In [ ]:
```python
ds, info = tfds.load('mnist', split='train', with_info=True)
fig = tfds.show_examples(ds, info, rows=1, cols=4)
```



3.5 Metadatos

Obtención de metadatos: mediante la API de `tfds.load` o la de `tfds.core.DatasetBuilder`

In [ ]:
```
ds, info = tfds.load('mnist', with_info=True)
print(info)
```

```
tfds.core.DatasetInfo(
    name='mnist',
    full_name='mnist/3.0.1',
    description="""
    The MNIST database of handwritten digits.
    """,
    homepage='http://yann.lecun.com/exdb/mnist/',
    data_dir='/home/ajuan/tensorflow_datasets/mnist/3.0.1',
    file_format=tfrecord,
    download_size=11.06 MiB,
    dataset_size=21.00 MiB,
    features=FeaturesDict({
        'image': Image(shape=(28, 28, 1), dtype=uint8),
        'label': ClassLabel(shape=(), dtype=int64, num_classes=10),
    }),
    supervised_keys=('image', 'label'),
    disable_shuffling=False,
    splits={
        'test': <SplitInfo num_examples=10000, num_shards=1>,
        'train': <SplitInfo num_examples=60000, num_shards=1>,
    },
    citation="""@article{lecun2010mnist,
      title={MNIST handwritten digit database},
      author={LeCun, Yann and Cortes, Corinna and Burges, CJ},
      journal={ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist},
      volume={2},
      year={2010}
    }""",
)
```

3.6 MNIST con Keras

In [ ]:
```
import tensorflow as tf; import tensorflow_datasets as tfds; import keras
(ds_train, ds_test), ds_info = tfds.load(
    'mnist', split=['train', 'test'], shuffle_files=True, as_supervised=True, with_info=True)
```

Training pipeline: aplica las siguientes transformaciones

- `tf.data.Dataset.map` : para normalizar imágenes; por ejemplo, de `tf.uint8` a `tf.float32`
- `tf.data.Dataset.cache` : para guardar imágenes en memoria caché antes de barajar
- `tf.data.Dataset.shuffle` : para barajar los datos; cuantos más, mejor
- `tf.data.Dataset.batch:` para producir batches tras el barajado en cada época
- `tf.data.Dataset.prefetch:` al final del pipeline para paralelizar preproceso y entrenamiento

In [ ]:
```
def normalize_img(image, label):
  return tf.cast(image, tf.float32) / 255., label

ds_train = ds_train.map(normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_train = ds_train.cache()
ds_train = ds_train.shuffle(ds_info.splits['train'].num_examples)
ds_train = ds_train.batch(128)
ds_train = ds_train.prefetch(tf.data.AUTOTUNE)
```

Test pipeline: como el de training, pero sin shuffle y cache tras batch

In [ ]:
```
ds_test = ds_test.map(normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_test = ds_test.batch(128)
ds_test = ds_test.cache()
ds_test = ds_test.prefetch(tf.data.AUTOTUNE)
```

Keras: definición del modelo, compilación y ajuste

In [ ]:
```
model = keras.models.Sequential([
  keras.layers.Flatten(input_shape=(28, 28, 1)),
  keras.layers.Dense(128, activation='relu'),
  keras.layers.Dense(10)])
model.compile(
    optimizer=keras.optimizers.Adam(0.001),
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[keras.metrics.SparseCategoricalAccuracy()])
model.fit(ds_train, epochs=6, validation_data=ds_test);
```

```
Epoch 1/6
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - loss: 0.6069 - sparse_categorical_accuracy: 0.8318 - val_loss: 0.1980 - v
al_sparse_categorical_accuracy: 0.9448
Epoch 2/6
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.1798 - sparse_categorical_accuracy: 0.9498 - val_loss: 0.1362 - v
al_sparse_categorical_accuracy: 0.9605
Epoch 3/6
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.1240 - sparse_categorical_accuracy: 0.9647 - val_loss: 0.1138 - v
al_sparse_categorical_accuracy: 0.9671
Epoch 4/6
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.0965 - sparse_categorical_accuracy: 0.9720 - val_loss: 0.0946 - v
al_sparse_categorical_accuracy: 0.9716
Epoch 5/6
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.0759 - sparse_categorical_accuracy: 0.9776 - val_loss: 0.0862 - v
al_sparse_categorical_accuracy: 0.9740
Epoch 6/6
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.0615 - sparse_categorical_accuracy: 0.9825 - val_loss: 0.0818 - v
al_sparse_categorical_accuracy: 0.9758
```

4 CIFAR10

4.1 Dataset

In [ ]:

```python
import os; os.environ["KERAS_BACKEND"] = "tensorflow"; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import numpy as np; import matplotlib.pyplot as plt
import tensorflow as tf; import tensorflow_datasets as tfds; import keras; import keras_cv
keras.utils.set_random_seed(23)
(train, test), info = tfds.load("cifar10", split=['train', 'test'], with_info=True, as_supervised=True)
print(info.description, "\n\n", info.splits, "\n\n", info.features)
```

```
The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 5000
0 training images and 10000 test images.

 {'train': <SplitInfo num_examples=50000, num_shards=1>, 'test': <SplitInfo num_examples=10000, num_shards=1>}

 FeaturesDict({
    'id': Text(shape=(), dtype=string),
    'image': Image(shape=(32, 32, 3), dtype=uint8),
    'label': ClassLabel(shape=(), dtype=int64, num_classes=10),
})
```

In [ ]:

```python
fig = tfds.show_examples(train, info, rows=1, cols=5)
```


horse (7)


ship (8)


deer (4)


deer (4)


frog (6)

4.2 Transfer learning

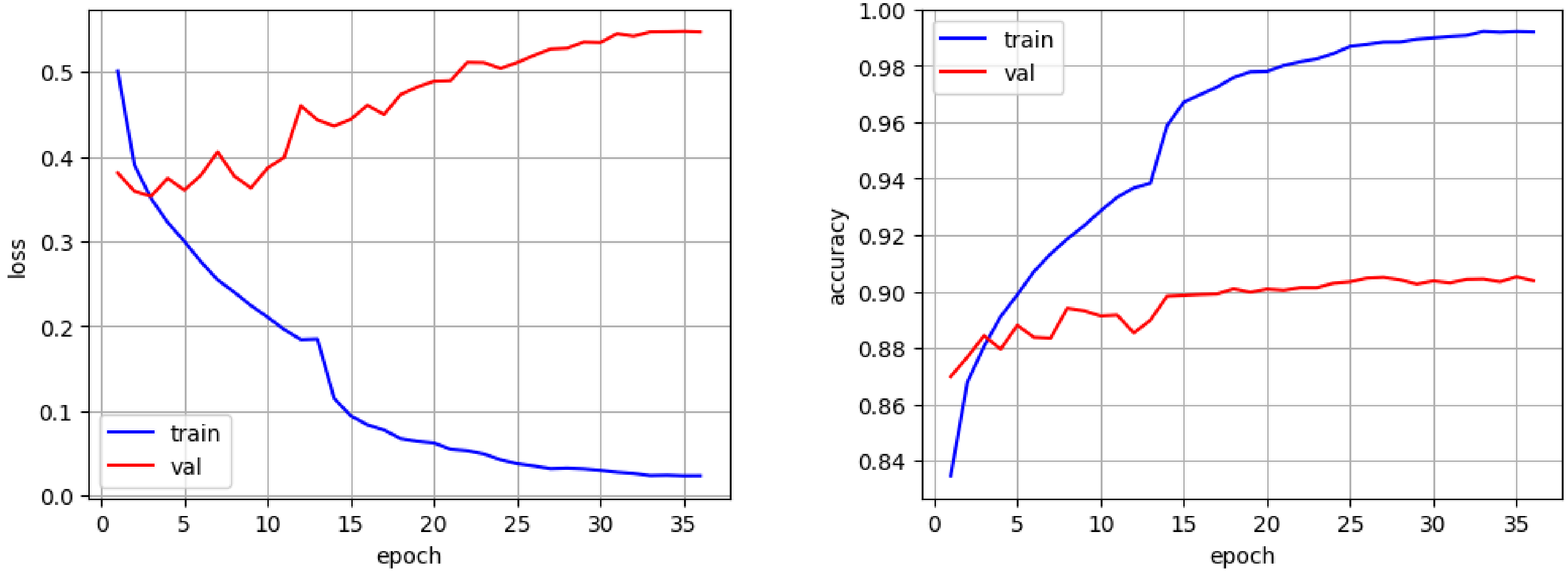
In [ ]:
```python
import os; os.environ["KERAS_BACKEND"] = "tensorflow"; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import numpy as np; import matplotlib.pyplot as plt
import tensorflow as tf; import tensorflow_datasets as tfds; import keras; import keras_cv
keras.utils.set_random_seed(23)
train, test = tfds.load("cifar10", split=['train', 'test'], as_supervised=True)
```

In [ ]:
```python
def normalize_images(images, labels):
    return tf.cast(images, tf.float32), tf.one_hot(labels, 10)
train = train.map(normalize_images, num_parallel_calls=tf.data.AUTOTUNE)
batch_size = 32; train = train.cache().shuffle(10 * batch_size).batch(batch_size).prefetch(tf.data.AUTOTUNE)
test = test.map(normalize_images, num_parallel_calls=tf.data.AUTOTUNE)
test = test.batch(batch_size).cache().prefetch(tf.data.AUTOTUNE)
```

In [ ]:
```python
import time; start = time.time()
inputs = keras.Input(shape=(32, 32, 3))
x = keras.layers.Rescaling(scale=1 / 127.5, offset=-1)(inputs)
x = keras.layers.Resizing(224, 224, interpolation="nearest")(x) # 224x224x3 en ResNet50V2
backbone = keras.applications.resnet_v2.ResNet50V2(include_top=False)
# backbone = keras_cv.models.ResNetV2Backbone.from_preset("resnet50_v2_imagenet", include_rescaling=False)
backbone.trainable = False
x = backbone(x, training=False)
x = keras.layers.GlobalAveragePooling2D()(x)
x = keras.layers.Dense(units=800, activation='relu')(x)
x = keras.layers.Dropout(0.5)(x)
predictions = keras.layers.Dense(10, activation='softmax')(x)
M = keras.models.Model(inputs=inputs, outputs=predictions)
opt = keras.optimizers.Adam(learning_rate=0.001)
M.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
check = keras.callbacks.ModelCheckpoint('cifar10.keras', monitor='val_accuracy', save_best_only=True, verbose=1)
reduce = keras.callbacks.ReduceLROnPlateau(
    monitor='val_accuracy', factor=0.3, patience=5, min_delta=0.0005, min_lr=0.0)
early = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=10, min_delta=0.0005)
H = M.fit(train, epochs=100, validation_data=test, verbose=1, callbacks=[check, early, reduce])
print('Tiempo (hh:mm:ss):', time.strftime('%H:%M:%S', time.gmtime(time.time() - start)))
```

```
Epoch 1: val_accuracy improved from -inf to 0.86980, saving model to cifar10.keras
Epoch 36: val_accuracy did not improve from 0.90520
Tiempo (hh:mm:ss): 00:39:36
```

```
In [ ]: fig, axes = plt.subplots(1, 2, figsize=(12, 4)); plt.subplots_adjust(wspace=0.3)
        xx = np.arange(1, len(H.history['loss'])+1)
        ax = axes[0]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('loss'); # ax.set_xticks(xx)
        ax.plot(xx, H.history['loss'], color='b', label='train')
        ax.plot(xx, H.history['val_loss'], color='r', label='val'); ax.legend()
        ax = axes[1]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('accuracy'); # ax.set_xticks(xx)
        ax.plot(xx, H.history['accuracy'], color='b', label='train')
        ax.plot(xx, H.history['val_accuracy'], color='r', label='val'); ax.legend();
```



4.3 Fine-tuning

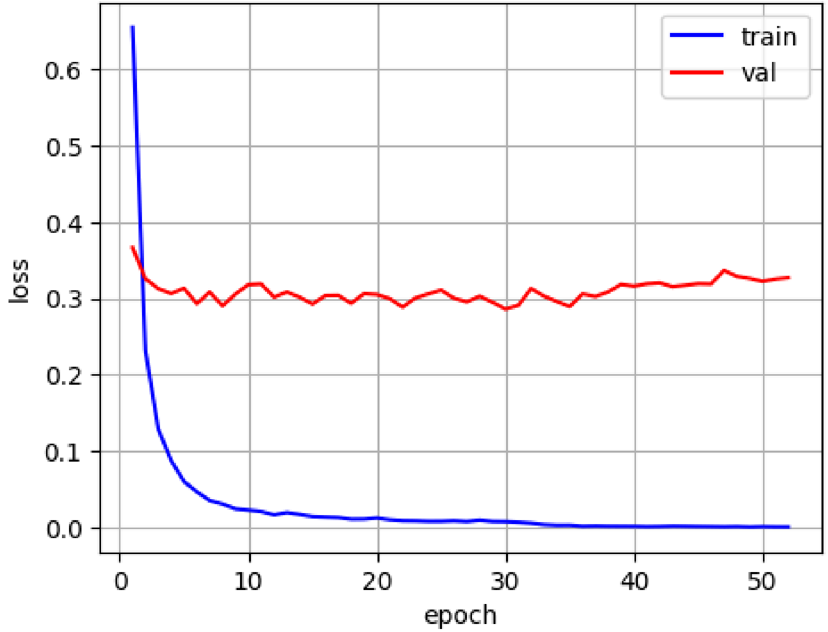
In [ ]:
```python
import os; os.environ["KERAS_BACKEND"] = "tensorflow"; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import numpy as np; import matplotlib.pyplot as plt
import tensorflow as tf; import tensorflow_datasets as tfds; import keras; import keras_cv
keras.utils.set_random_seed(23)
train, test = tfds.load("cifar10", split=['train', 'test'], as_supervised=True)
```

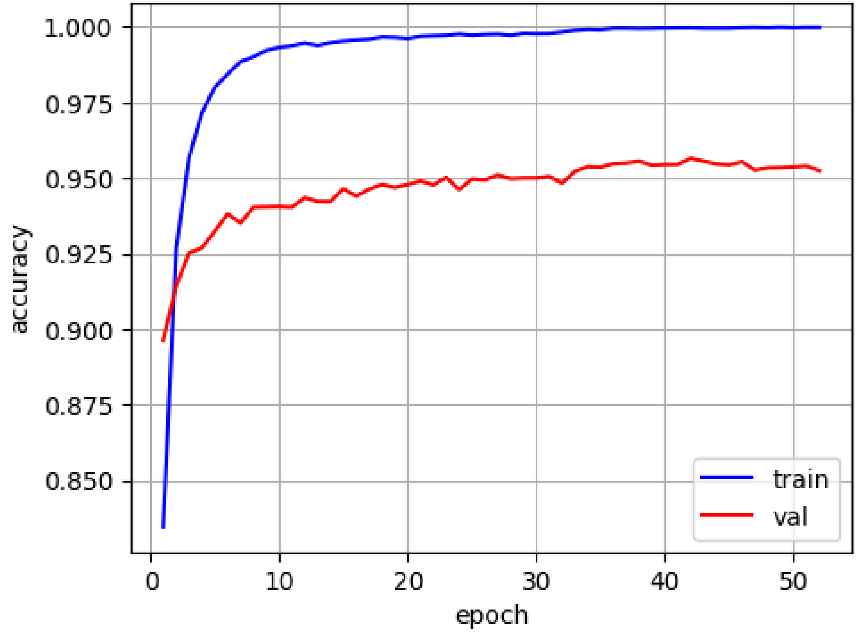
In [ ]:
```python
def normalize_images(images, labels):
    return tf.cast(images, tf.float32), tf.one_hot(labels, 10)
train = train.map(normalize_images, num_parallel_calls=tf.data.AUTOTUNE)
batch_size = 32; train = train.cache().shuffle(10 * batch_size).batch(batch_size).prefetch(tf.data.AUTOTUNE)
test = test.map(normalize_images, num_parallel_calls=tf.data.AUTOTUNE)
test = test.batch(batch_size).cache().prefetch(tf.data.AUTOTUNE)
```

In [ ]:
```python
import time; start = time.time()
M = keras.models.load_model('cifar10.keras')
for layer in M.layers:
    if not isinstance(layer, keras.layers.BatchNormalization):
        layer.trainable = True
opt = keras.optimizers.Adam(learning_rate=1e-5)
M.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
check = keras.callbacks.ModelCheckpoint('cifar10ft.keras', monitor='val_accuracy', save_best_only=True, verbose=1)
reduce = keras.callbacks.ReduceLROnPlateau(
    monitor='val_accuracy', factor=0.3, patience=5, min_delta=0.0005, min_lr=0.0)
early = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=10, min_delta=0.0005)
H = M.fit(train, epochs=100, validation_data=test, verbose=1, callbacks=[check, early, reduce])
print('Tiempo (hh:mm:ss):', time.strftime('%H:%M:%S', time.gmtime(time.time() - start)))
```

```
Epoch 1: val_accuracy improved from -inf to 0.89640, saving model to cifar10ft.keras
Epoch 52: val_accuracy did not improve from 0.95660
Tiempo (hh:mm:ss): 02:58:12
```

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(12, 4)); plt.subplots_adjust(wspace=0.3)
xx = np.arange(1, len(H.history['loss'])+1)
ax = axes[0]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('loss'); # ax.set_xticks(xx)
ax.plot(xx, H.history['loss'], color='b', label='train')
ax.plot(xx, H.history['val_loss'], color='r', label='val'); ax.legend()
ax = axes[1]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('accuracy'); # ax.set_xticks(xx)
ax.plot(xx, H.history['accuracy'], color='b', label='train')
ax.plot(xx, H.history['val_accuracy'], color='r', label='val'); ax.legend();
```





4.4 Aumento de datos

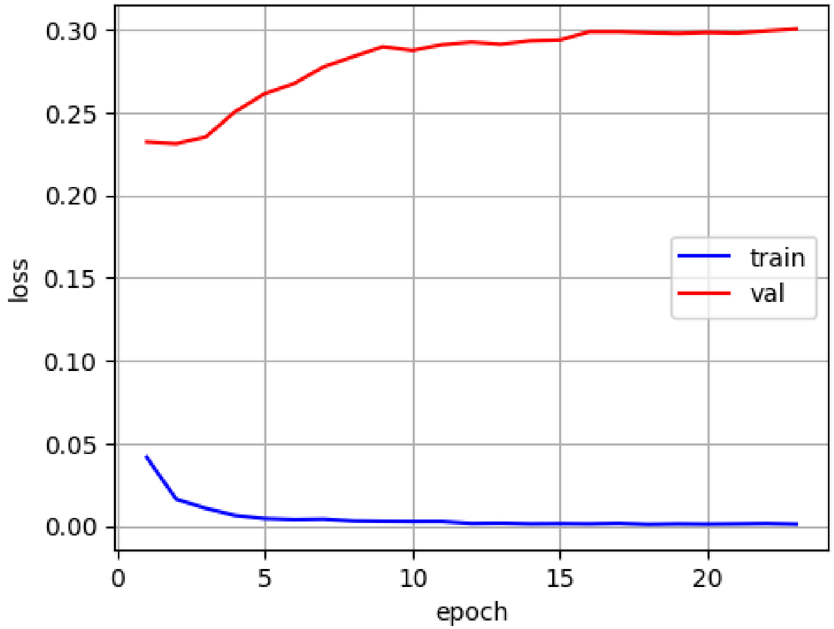
In [ ]:
```
import os; os.environ["KERAS_BACKEND"] = "tensorflow"; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import numpy as np; import matplotlib.pyplot as plt
import tensorflow as tf; import tensorflow_datasets as tfds; import keras; import keras_cv
keras.utils.set_random_seed(23)
train, test = tfds.load("cifar10", split=['train', 'test'], as_supervised=True)
```

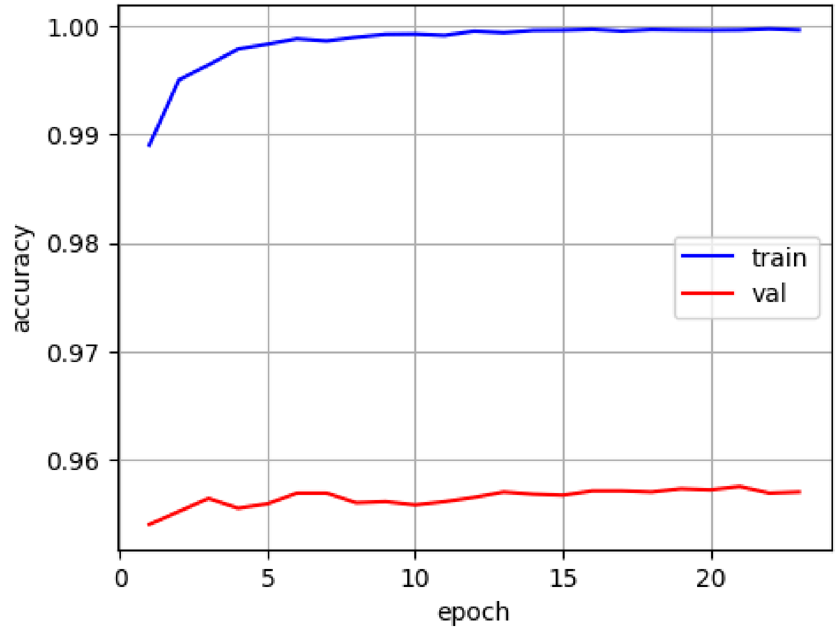
In [ ]:
```
def normalize_images(images, labels):
    return tf.cast(images, tf.float32), tf.one_hot(labels, 10)
random_flip = keras_cv.layers.RandomFlip("horizontal")
def augment_images(images, labels):
    images = random_flip(images)
    return normalize_images(images, labels)
train = train.map(augment_images, num_parallel_calls=tf.data.AUTOTUNE)
batch_size = 32; train = train.cache().shuffle(10 * batch_size).batch(batch_size).prefetch(tf.data.AUTOTUNE)
test = test.map(normalize_images, num_parallel_calls=tf.data.AUTOTUNE)
test = test.batch(batch_size).cache().prefetch(tf.data.AUTOTUNE)
```

In [ ]:
```
import time; start = time.time()
M = keras.models.load_model('cifar10ft.keras')
# M.summary(line_length=95, positions=[0.59, 0.82, 0.94, 1.], show_trainable=True)
for layer in M.layers:
    if not isinstance(layer, keras.layers.BatchNormalization):
        layer.trainable = True
#opt = keras.optimizers.Adam(learning_rate=1e-5)
#M.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
check = keras.callbacks.ModelCheckpoint('cifar10fta.keras', monitor='val_accuracy', save_best_only=True, verbose=1)
reduce = keras.callbacks.ReduceLROnPlateau(
    monitor='val_accuracy', factor=0.3, patience=5, min_delta=0.0005, min_lr=0.0)
early = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=10, min_delta=0.0005)
H = M.fit(train, epochs=100, validation_data=test, verbose=1, callbacks=[check, early, reduce])
print('Tiempo (hh:mm:ss):', time.strftime('%H:%M:%S', time.gmtime(time.time() - start)))
```

```
Epoch 1: val_accuracy improved from -inf to 0.95400, saving model to cifar10fta.keras
Epoch 23: val_accuracy did not improve from 0.95750
Tiempo (hh:mm:ss): 01:19:33
```

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(12, 4)); plt.subplots_adjust(wspace=0.3)
xx = np.arange(1, len(H.history['loss'])+1)
ax = axes[0]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('loss'); # ax.set_xticks(xx)
ax.plot(xx, H.history['loss'], color='b', label='train')
ax.plot(xx, H.history['val_loss'], color='r', label='val'); ax.legend()
ax = axes[1]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('accuracy'); # ax.set_xticks(xx)
ax.plot(xx, H.history['accuracy'], color='b', label='train')
ax.plot(xx, H.history['val_accuracy'], color='r', label='val'); ax.legend();
```





5 Ejercicio: Cats vs Dogs

Ejercicio: haz un experimento similar con Cats vs Dogs y `split=['train[:50%]', 'train[80%:]']`

5.1 Cats vs Dogs

In [ ]:
```
import os; os.environ["KERAS_BACKEND"] = "tensorflow"; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import numpy as np; import matplotlib.pyplot as plt
import tensorflow as tf; import tensorflow_datasets as tfds; import keras; import keras_cv
keras.utils.set_random_seed(23)
train, info = tfds.load("cats_vs_dogs", split='train', with_info=True, as_supervised=True)
print(info.description, "\n", info.splits, "\n", info.features)
```

```
A large set of images of cats and dogs. There are 1738 corrupted images that are dropped.
 {'train': <SplitInfo num_examples=23262, num_shards=16>}
 FeaturesDict({
    'image': Image(shape=(None, None, 3), dtype=uint8),
    'image/filename': Text(shape=(), dtype=string),
    'label': ClassLabel(shape=(), dtype=int64, num_classes=2),
})
```

In [ ]:
```
fig = tfds.show_examples(train, info, rows=1, cols=7)
```

5.2 Transfer learning

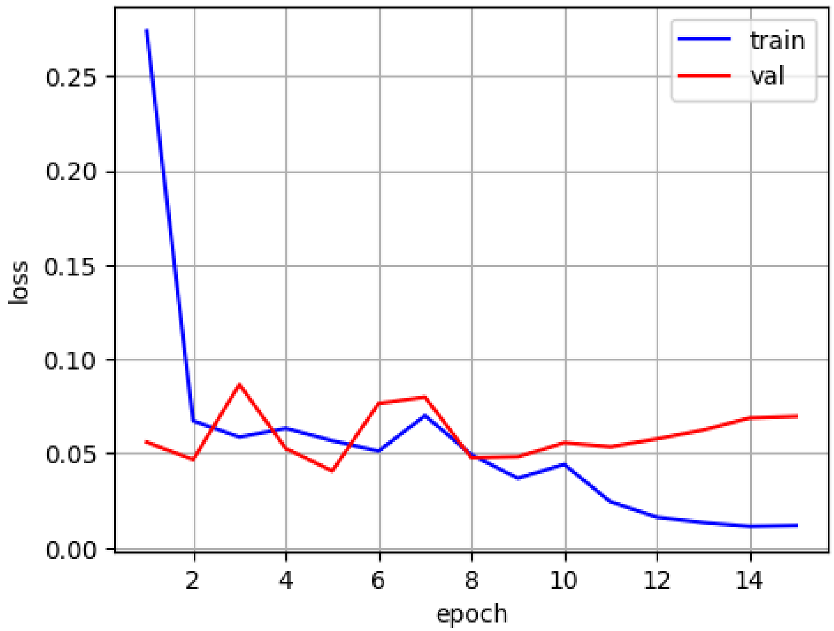
In [ ]:
```python
import os; os.environ["KERAS_BACKEND"] = "tensorflow"; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import numpy as np; import matplotlib.pyplot as plt
import tensorflow as tf; import tensorflow_datasets as tfds; import keras; import keras_cv
keras.utils.set_random_seed(23)
train, test = tfds.load("cats_vs_dogs", split=['train[:50%]', 'train[80%:]'], as_supervised=True)
```

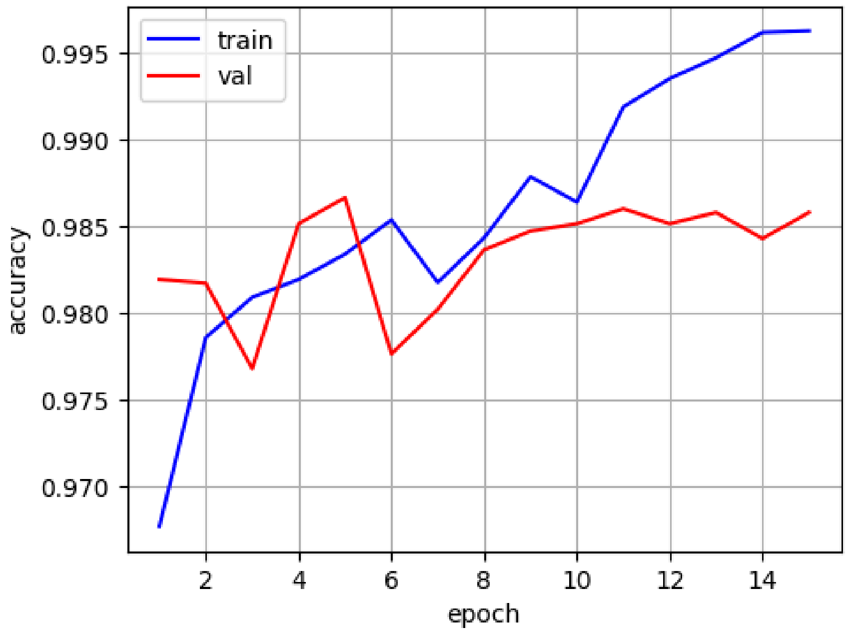
In [ ]:
```python
resize = keras_cv.layers.Resizing(224, 224, interpolation="nearest", pad_to_aspect_ratio=True)
def normalize_images(images, labels):
    return tf.cast(resize(images), tf.float32), labels
train = train.map(normalize_images, num_parallel_calls=tf.data.AUTOTUNE)
batch_size = 32; train = train.cache().shuffle(10 * batch_size).batch(batch_size).prefetch(tf.data.AUTOTUNE)
test = test.map(normalize_images, num_parallel_calls=tf.data.AUTOTUNE)
test = test.batch(batch_size).cache().prefetch(tf.data.AUTOTUNE)
```

In [ ]:
```python
import time; start = time.time()
inputs = keras.Input(shape=(None, None, 3))
x = keras.layers.Rescaling(scale=1 / 127.5, offset=-1)(inputs)
backbone = keras.applications.resnet_v2.ResNet50V2(include_top=False)
backbone.trainable = False
x = backbone(x, training=False)
x = keras.layers.GlobalAveragePooling2D()(x)
x = keras.layers.Dense(units=800, activation='relu')(x)
x = keras.layers.Dropout(0.5)(x)
predictions = keras.layers.Dense(1, activation='sigmoid')(x)
M = keras.models.Model(inputs=inputs, outputs=predictions)
opt = keras.optimizers.Adam(learning_rate=0.005) # la mitad del usado con CIFAR10
M.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
check = keras.callbacks.ModelCheckpoint('catdogs.keras', monitor='val_accuracy', save_best_only=True, verbose=1)
reduce = keras.callbacks.ReduceLROnPlateau(
    monitor='val_accuracy', factor=0.3, patience=5, min_delta=0.0005, min_lr=0.0)
early = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=10, min_delta=0.0005)
H = M.fit(train, epochs=100, validation_data=test, verbose=1, callbacks=[check, early, reduce])
print('Tiempo (hh:mm:ss):', time.strftime('%H:%M:%S', time.gmtime(time.time() - start)))
```

```
Epoch 1: val_accuracy improved from -inf to 0.98194, saving model to catdogs.keras
Epoch 15: val_accuracy did not improve from 0.98667
Tiempo (hh:mm:ss): 00:05:35
```

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(12, 4)); plt.subplots_adjust(wspace=0.3)
xx = np.arange(1, len(H.history['loss'])+1)
ax = axes[0]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('loss'); # ax.set_xticks(xx)
ax.plot(xx, H.history['loss'], color='b', label='train')
ax.plot(xx, H.history['val_loss'], color='r', label='val'); ax.legend()
ax = axes[1]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('accuracy'); # ax.set_xticks(xx)
ax.plot(xx, H.history['accuracy'], color='b', label='train')
ax.plot(xx, H.history['val_accuracy'], color='r', label='val'); ax.legend();
```





5.3 Fine-tuning

In [ ]:
```
import os; os.environ["KERAS_BACKEND"] = "tensorflow"; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import numpy as np; import matplotlib.pyplot as plt
import tensorflow as tf; import tensorflow_datasets as tfds; import keras; import keras_cv
keras.utils.set_random_seed(23)
train, test = tfds.load("cats_vs_dogs", split=['train[:50%]', 'train[80%:]'], as_supervised=True)
```

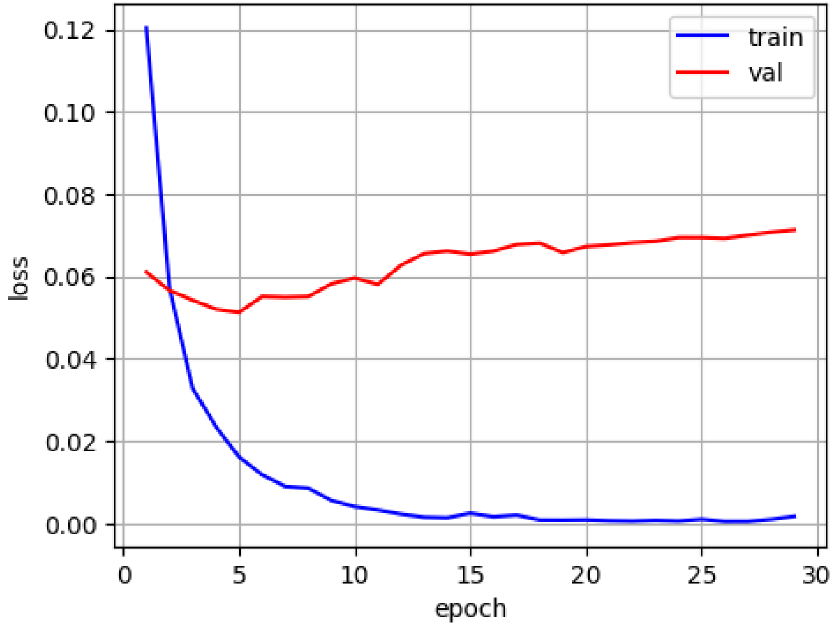
In [ ]:
```
resize = keras_cv.layers.Resizing(224, 224, interpolation="nearest", pad_to_aspect_ratio=True)
def normalize_images(images, labels):
    return tf.cast(resize(images), tf.float32), labels
train = train.map(normalize_images, num_parallel_calls=tf.data.AUTOTUNE)
batch_size = 32; train = train.cache().shuffle(10 * batch_size).batch(batch_size).prefetch(tf.data.AUTOTUNE)
test = test.map(normalize_images, num_parallel_calls=tf.data.AUTOTUNE)
test = test.batch(batch_size).cache().prefetch(tf.data.AUTOTUNE)
```

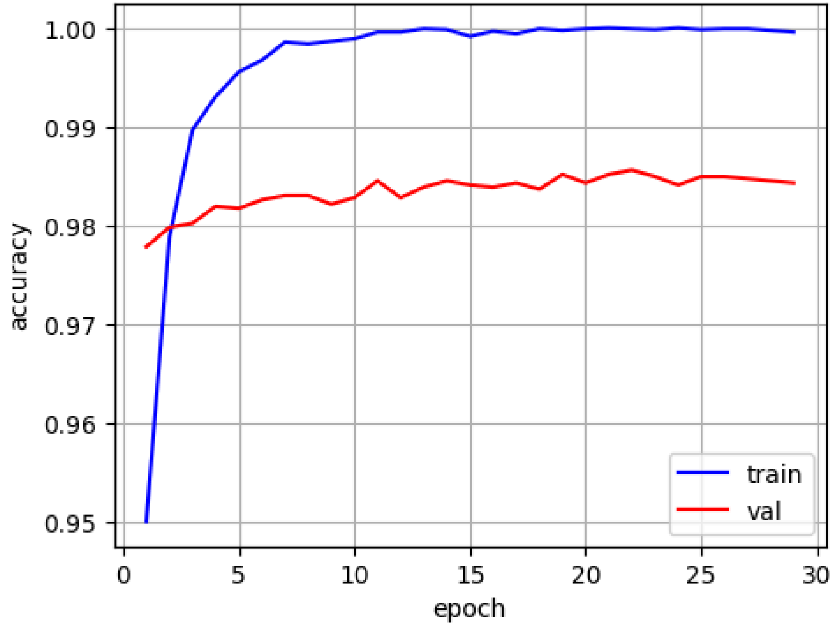
In [ ]:
```
import time; start = time.time()
M = keras.models.load_model('catdogs.keras')
for layer in M.layers:
    if not isinstance(layer, keras.layers.BatchNormalization):
        layer.trainable = True
opt = keras.optimizers.Adam(learning_rate=5e-6)
M.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
check = keras.callbacks.ModelCheckpoint('catdogsft.keras', monitor='val_accuracy', save_best_only=True, verbose=1)
reduce = keras.callbacks.ReduceLROnPlateau(
    monitor='val_accuracy', factor=0.3, patience=5, min_delta=0.0005, min_lr=0.0)
early = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=10, min_delta=0.0005)
H = M.fit(train, epochs=100, validation_data=test, verbose=1, callbacks=[check, early, reduce])
print('Tiempo (hh:mm:ss):', time.strftime('%H:%M:%S', time.gmtime(time.time() - start)))
```

```
Epoch 1: val_accuracy improved from -inf to 0.97786, saving model to catdogsft.keras
Epoch 29: val_accuracy did not improve from 0.98560
Tiempo (hh:mm:ss): 00:26:35
```

In [ ]:

```
fig, axes = plt.subplots(1, 2, figsize=(12, 4)); plt.subplots_adjust(wspace=0.3)
xx = np.arange(1, len(H.history['loss'])+1)
ax = axes[0]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('loss'); # ax.set_xticks(xx)
ax.plot(xx, H.history['loss'], color='b', label='train')
ax.plot(xx, H.history['val_loss'], color='r', label='val'); ax.legend()
ax = axes[1]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('accuracy'); # ax.set_xticks(xx)
ax.plot(xx, H.history['accuracy'], color='b', label='train')
ax.plot(xx, H.history['val_accuracy'], color='r', label='val'); ax.legend();
```





5.4 Aumento de datos

In [ ]:
```
import os; os.environ["KERAS_BACKEND"] = "tensorflow"; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import numpy as np; import matplotlib.pyplot as plt
import tensorflow as tf; import tensorflow_datasets as tfds; import keras; import keras_cv
keras.utils.set_random_seed(23)
train, test = tfds.load("cats_vs_dogs", split=['train[:50%]', 'train[80%:]'], as_supervised=True)
```

In [ ]:
```
resize = keras_cv.layers.Resizing(224, 224, interpolation="nearest", pad_to_aspect_ratio=True)
def normalize_images(images, labels):
    return tf.cast(resize(images), tf.float32), labels
random_flip = keras_cv.layers.RandomFlip("horizontal")
def augment_images(images, labels):
    return tf.cast(resize(random_flip(images)), tf.float32), labels
train = train.map(augment_images, num_parallel_calls=tf.data.AUTOTUNE)
batch_size = 32; train = train.cache().shuffle(10 * batch_size).batch(batch_size).prefetch(tf.data.AUTOTUNE)
test = test.map(normalize_images, num_parallel_calls=tf.data.AUTOTUNE)
test = test.batch(batch_size).cache().prefetch(tf.data.AUTOTUNE)
```

In [ ]:
```
import time; start = time.time()
M = keras.models.load_model('catdogsft.keras')
for layer in M.layers:
    if not isinstance(layer, keras.layers.BatchNormalization):
        layer.trainable = True
check = keras.callbacks.ModelCheckpoint('catdogsfta.keras', monitor='val_accuracy', save_best_only=True, verbose=1)
reduce = keras.callbacks.ReduceLROnPlateau(
    monitor='val_accuracy', factor=0.3, patience=5, min_delta=0.0005, min_lr=0.0)
early = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=10, min_delta=0.0005)
H = M.fit(train, epochs=100, validation_data=test, verbose=1, callbacks=[check, early, reduce])
print('Tiempo (hh:mm:ss):', time.strftime('%H:%M:%S', time.gmtime(time.time() - start)))
```

```
Epoch 1: val_accuracy improved from -inf to 0.98581, saving model to catdogsfta.keras
Epoch 11: val_accuracy did not improve from 0.98581
Tiempo (hh:mm:ss): 00:10:35
```

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(12, 4)); plt.subplots_adjust(wspace=0.3)
xx = np.arange(1, len(H.history['loss'])+1)
ax = axes[0]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('loss'); # ax.set_xticks(xx)
ax.plot(xx, H.history['loss'], color='b', label='train')
ax.plot(xx, H.history['val_loss'], color='r', label='val'); ax.legend()
ax = axes[1]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('accuracy'); # ax.set_xticks(xx)
ax.plot(xx, H.history['accuracy'], color='b', label='train')
ax.plot(xx, H.history['val_accuracy'], color='r', label='val'); ax.legend();
```