TL08 Arquitecturas comunes

Índice

1 Arquitecturas comunes

Stem-body-head: meta-arquitectura típica de las CNNs modernas, típicamente muy profundas

- **Stem (raíz):** dos o tres capas convolucionales que extraen características de bajo nivel
- **Body (cuerpo):** subred de **bloques convolucionales** repetidos muchas veces
- **Head (cabeza):** transforma la salida del cuerpo según la tarea a abordar (clasificación, segmentación, etc.)

Entrenamiento de redes muy profundas: difícil a causa de gradientes que explotan y, sobre todo, desaparecen

- **Gradient clipping:** solución sencilla para los que explotan
- **Funciones de activación:** que no favorezcan gradientes demasiado pequeños (o grandes)
- **Conexiones residuales:** conexiones aditivas que permiten saltarse capas bloqueantes
- **Normalización de salidas:** a nivel batch o capa

Keras Applications: modelos de arquitecturas comunes pre-entrenados en ImageNet

- **Documentación:** https://keras.io/api/applications
- **Modelos disponibles:** Xception, VGG16, VGG19, ResNet50, ResNet50V2, etc.
- **Usos directos:** predicción de clases ImageNet y extracción de características
- **Transfer learning:** congelamos un modelo pre-entrenado descabezado y entrenamos una nueva cabeza
- **Fine-tuning:** descongelamos (parte de) el modelo y lo re-entrenamos (suavemente)

2 CIFAR-10

CIFAR-10: $60\,000$ imágenes 32×32 a color de 10 clases; 6000 por clase

Clases: airplane (0), automobile (1), bird (2), cat (3), deer (4), dog (5), frog (6), horse (7), ship (8), truck (9)

Partición estándar: $50\,000$ muestras para training y $10\,000$ para test

Incluido en Keras: https://keras.io/api/datasets/cifar10

Fuente original: https://www.cs.toronto.edu/~kriz/cifar.html

Tarea muy popular: desde su introducción en 2009, CIFAR-10 ha sido muy usado para comparar técnicas de ML

SOTA por debajo del 0.5%: https://paperswithcode.com/sota/image-classification-on-cifar-10

Precisión del 94% en menos de 4 segundos A100: https://arxiv.org/pdf/2404.00498

Más info: https://en.wikipedia.org/wiki/CIFAR-10

CIFAR-100: versión de 100 clases (20 superclases de 5 clases cada una); 600 imágenes por clase

Lectura:

```
In [ ]: import numpy as np; import matplotlib.pyplot as plt
import os; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'; import keras; import keras_cv
(x_train_val, y_train_val), (x_test, y_test) = keras.datasets.cifar10.load_data()
```

Visualización:

```
In [ ]: rows = 1; cols = 10; scale = 2
print(y_train_val[:rows*cols].T)
keras_cv.visualization.plot_image_gallery(
    np.array(x_train_val[:rows*cols]), rows=rows, cols=cols, value_range=(0, 255), show=True, scale=scale)
```

```
[[6 9 9 4 1 1 2 7 8 3]]
```

Formato y partición de train_val:

```
In [ ]: x_train_val = x_train_val.astype("float32")
x_test = x_test.astype("float32")
y_train_val = keras.utils.to_categorical(y_train_val, 10)
y_test = keras.utils.to_categorical(y_test, 10)
x_train = x_train_val[:-10000]; x_val = x_train_val[-10000:]
y_train = y_train_val[:-10000]; y_val = y_train_val[-10000:]
print(x_train.shape, y_train.shape, x_val.shape, y_val.shape, x_test.shape, y_test.shape)
```

```
(40000, 32, 32, 3) (40000, 10) (10000, 32, 32, 3) (10000, 10) (10000, 32, 32, 3) (10000, 10)
```

3 Una CNN sencilla

Inicialización: librerías, semilla, lectura de CIFAR-10 y partición train-val-test

In [ ]:
```
import numpy as np; import matplotlib.pyplot as plt
import os; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras; from keras import layers; import keras_tuner
keras.utils.set_random_seed(23)
(x_train_val, y_train_val), (x_test, y_test) = keras.datasets.cifar10.load_data()
x_train_val = x_train_val.astype("float32")
x_test = x_test.astype("float32")
y_train_val = keras.utils.to_categorical(y_train_val, 10)
y_test = keras.utils.to_categorical(y_test, 10)
x_train = x_train_val[:-10000]; x_val = x_train_val[-10000:]
y_train = y_train_val[:-10000]; y_val = y_train_val[-10000:]
print(x_train.shape, y_train.shape, x_val.shape, y_val.shape, x_test.shape, y_test.shape)
```

```
(40000, 32, 32, 3) (40000, 10) (10000, 32, 32, 3) (10000, 10) (10000, 32, 32, 3) (10000, 10)
```

MyHyperModel: basada en la de (Fashion-)MNIST

In [ ]:
```
class MyHyperModel(keras_tuner.HyperModel):
    def build(self, hp):
        input_shape = (32, 32, 3)
        inputs = keras.Input(shape=input_shape)
        inputs = layers.Rescaling(1./255)(inputs)
        filters = 32
        conv = layers.Conv2D(filters, kernel_size=(3, 3), activation="relu")(inputs)
        pooling = layers.MaxPooling2D(pool_size=(2, 2))(conv)
        conv = layers.Conv2D(2*filters, kernel_size=(3, 3), activation="relu")(pooling)
        pooling = layers.MaxPooling2D(pool_size=(2, 2))(conv)
        dropout = 0.5
        x = layers.Flatten()(pooling)
        x = layers.Dense(units=800, activation='relu')(x)
        x = layers.Dropout(dropout)(x)
        predictions = layers.Dense(10, activation='softmax')(x)
        M = keras.models.Model(inputs=inputs, outputs=predictions)
        learning_rate = hp.Float("learning_rate", min_value=0.001, max_value=0.002)
        opt = keras.optimizers.Adam(learning_rate=learning_rate)
        M.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
        return M
    def fit(self, hp, M, x, y, xy_val, **kwargs):
        factor = 0.38; patience = 5
        reduce_cb = keras.callbacks.ReduceLROnPlateau(
            monitor='val_accuracy', factor=factor, patience=patience, min_delta=0.0, min_lr=0.0)
        early_cb = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=2*patience, min_delta=0)
        kwargs['callbacks'].extend([reduce_cb, early_cb])
        return M.fit(x, y, batch_size=256, epochs=100, validation_data=xy_val, **kwargs)
```

Experimento: exploración y resumen de resultados

In [ ]:
```
tuner = keras_tuner.BayesianOptimization(
    MyHyperModel(), objective="val_accuracy", max_trials=10, executions_per_trial=1,
    overwrite=True, directory="/tmp", project_name="CIFAR-10")
```

In [ ]:
```
tuner.search(x_train, y_train, (x_val, y_val))
```

```
Trial 10 Complete [00h 25m 57s]
val_accuracy: 0.6380000114440918

Best val_accuracy So Far: 0.659500002861023
Total elapsed time: 03h 42m 39s
```

In [ ]:
```
tuner.results_summary(num_trials=3)
```

```
Results summary
Results in /tmp/CIFAR-10
Showing 3 best trials
Objective(name="val_accuracy", direction="max")

Trial 02 summary
Hyperparameters:
learning rate: 0.001696793759981646
Score: 0.659500002861023

Trial 00 summary
Hyperparameters:
learning rate: 0.0015920633683537378
Score: 0.6413000226020813

Trial 07 summary
Hyperparameters:
learning rate: 0.0014018797565339002
Score: 0.6392999887466431
```

Experimento (cont.): evaluación en test de los mejores modelos en validación

In [ ]:
```
num_models = 10
best_hyperparameters = tuner.get_best_hyperparameters(num_trials=num_models)
best_models = tuner.get_best_models(num_models=num_models)
for m in range(num_models):
    values = best_hyperparameters[m].values
    score = best_models[m].evaluate(x_test, y_test, verbose=0)
    print(f'Model {m}: Hyperparameters: {values!s} Loss: {score[0]:.4} Precisión: {score[1]:.2%}')
```

```
Model 0: Hyperparameters: {'learning rate': 0.0016967937599816461} Loss: 2.525 Precisión: 64.62%
Model 1: Hyperparameters: {'learning rate': 0.0015920633683537378} Loss: 2.55 Precisión: 63.11%
Model 2: Hyperparameters: {'learning rate': 0.0014018797565339002} Loss: 2.038 Precisión: 64.07%
Model 3: Hyperparameters: {'learning rate': 0.0012163977280996831} Loss: 2.044 Precisión: 63.75%
Model 4: Hyperparameters: {'learning rate': 0.0017119106061321845} Loss: 2.33 Precisión: 62.75%
Model 5: Hyperparameters: {'learning rate': 0.0017575639764545671} Loss: 2.383 Precisión: 63.91%
Model 6: Hyperparameters: {'learning rate': 0.0018752588088493051} Loss: 2.775 Precisión: 61.84%
Model 7: Hyperparameters: {'learning rate': 0.0016551821566245157} Loss: 2.992 Precisión: 62.04%
Model 8: Hyperparameters: {'learning rate': 0.0014274725309133542} Loss: 2.872 Precisión: 62.55%
Model 9: Hyperparameters: {'learning rate': 0.0014468337750734056} Loss: 2.661 Precisión: 59.80%
```

Conclusión: muy lejos de las precisiones que se consiguen con redes profundas

4 ResNet

ResNet50V2: red residual profunda "pequeña" pre-entrenada en ImageNet

In [ ]:
```
import numpy as np; import matplotlib.pyplot as plt; import os; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras; from keras.applications.resnet_v2 import ResNet50V2
M = ResNet50V2(include_top=True, weights='imagenet')
print('Primeras capas de un total de', len(M.layers),'\n')
# lines = []; M.summary(line_length=120, print_fn=lambda x: lines.append(x)); summary = '\n'.join(lines)
M.summary(line_length=95, positions=[0.38, 0.66, 0.78, 1.], show_trainable=True, layer_range=('input', 'pool1'))
```

```
Primeras capas de un total de 192
```

Model: "resnet50v2"

Layer (type)	Output Shape	Param #	Connected to	Traina…
input_layer_14 (InputLayer)	(None, 224, 224, 3)	0	-	-
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	input_layer_14…	-
conv1_conv (Conv2D)	(None, 112, 112, 64)	9,472	conv1_pad[0][0]	Y
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	conv1_conv[0][…	-
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)	0	pool1_pad[0][0]	-

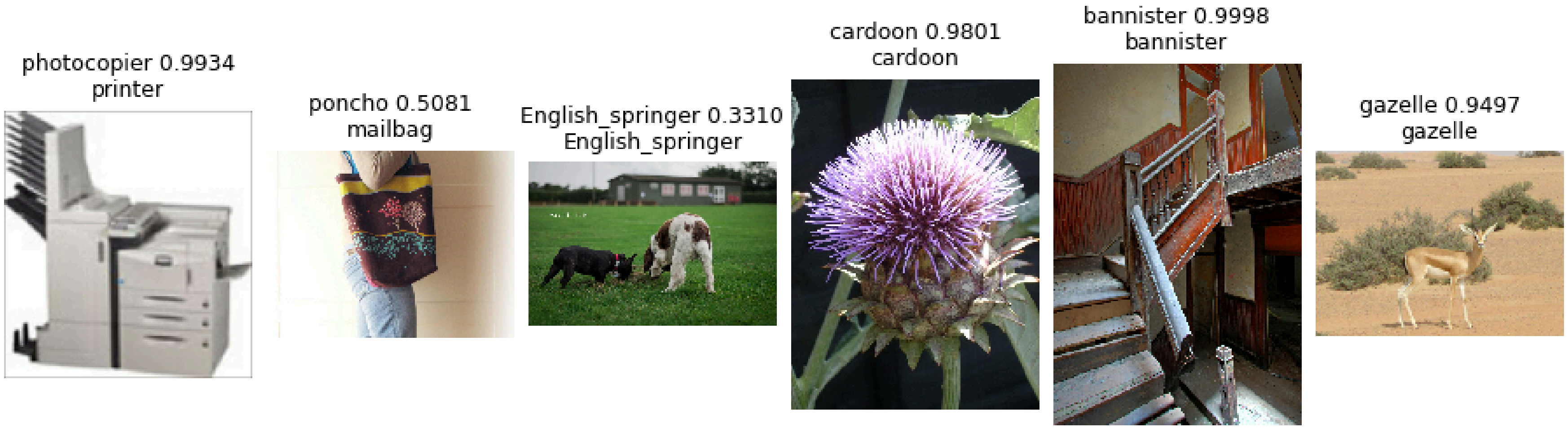
Total params: 25,613,800 (97.71 MB)

Trainable params: 25,568,360 (97.54 MB)

Non-trainable params: 45,440 (177.50 KB)

Predicción de clases ImageNet:

In [ ]:
```python
import tensorflow_datasets as tfds
from keras.preprocessing.image import smart_resize, img_to_array
from keras.applications.resnet_v2 import ResNet50V2, decode_predictions, preprocess_input
ds = tfds.load('imagenet_v2', split='test', as_supervised=True)
N = 16; ds = ds.take(N); nrows = 1; ncols = 6
_, axs = plt.subplots(nrows=nrows, ncols=ncols, figsize=(1.5*ncols, 1.5*ncols), constrained_layout=True)
for ax, img_label in zip(axs.flat, ds.as_numpy_iterator()):
    ax.set_axis_off(); ax.imshow(img_label[0], interpolation="none")
    img = np.expand_dims(img_to_array(smart_resize(img_label[0], M.input_shape[1:-1])), axis=0)
    pred = np.squeeze(decode_predictions(M.predict(preprocess_input(img), verbose=0), top=1))
    ref = np.squeeze(decode_predictions(np.eye(1, 1000, img_label[1]), top=1))
    ax.set_title(f'{pred[1]} {pred[2]:.6}\n{ref[1]}', fontsize=9)
```



5 Transfer learning

Inicialización: librerías, semilla, lectura de CIFAR-10 y partición train-val-test

In [ ]:
```
import numpy as np; import matplotlib.pyplot as plt
import os; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras; from keras import layers
keras.utils.set_random_seed(23)
(x_train_val, y_train_val), (x_test, y_test) = keras.datasets.cifar10.load_data()
x_train_val = x_train_val.astype("float32")
x_test = x_test.astype("float32")
y_train_val = keras.utils.to_categorical(y_train_val, 10)
y_test = keras.utils.to_categorical(y_test, 10)
x_train = x_train_val[:-10000]; x_val = x_train_val[-10000:]
y_train = y_train_val[:-10000]; y_val = y_train_val[-10000:]
print(x_train.shape, y_train.shape, x_val.shape, y_val.shape, x_test.shape, y_test.shape)
```

```
(40000, 32, 32, 3) (40000, 10) (10000, 32, 32, 3) (10000, 10) (10000, 32, 32, 3) (10000, 10)
```

Construcción del modelo: ResNet50V2 descabezada con entrada ajustada y nueva cabeza entrenable

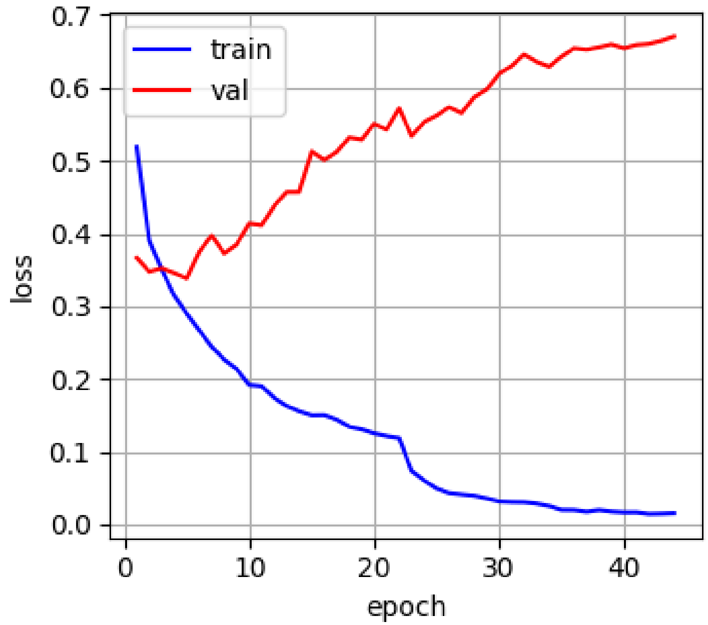
In [ ]:
```
inputs = keras.Input(shape=(32, 32, 3))
x = layers.Rescaling(scale=1 / 127.5, offset=-1)(inputs)
x = layers.Resizing(224, 224, interpolation="nearest")(x)
base_M = keras.applications.resnet_v2.ResNet50V2(include_top=False)
base_M.trainable = False
x = base_M(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(units=800, activation='relu')(x)
x = layers.Dropout(0.5)(x)
predictions = 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"])
```

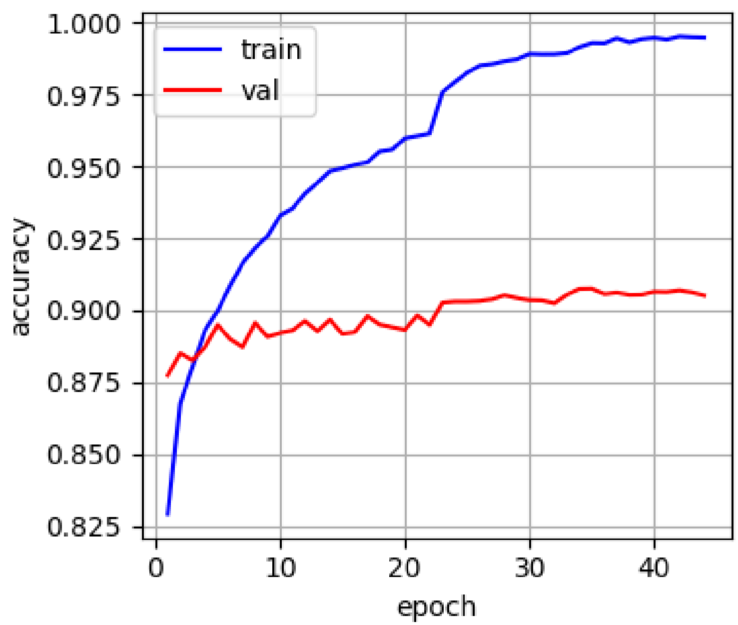
Entrenamiento del modelo: guardamos el mejor modelo con un callback checkpoint

In [ ]:
```
filename = 'CIFAR10_transfer_learning.keras'
checkpoint_cb = keras.callbacks.ModelCheckpoint(
    filepath=filename, monitor='val_accuracy', save_best_only=True, verbose=1)
reduce_cb = keras.callbacks.ReduceLROnPlateau(
    monitor='val_accuracy', factor=0.3, patience=5, min_delta=0.0005, min_lr=0.0)
early_cb = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=10, min_delta=0.0005)
H = M.fit(x_train, y_train, batch_size=32, epochs=500, validation_data=(x_val, y_val), verbose=0,
    callbacks=[checkpoint_cb, early_cb, reduce_cb])
```

```
Epoch 1: val_accuracy improved from -inf to 0.87740, saving model to CIFAR10_transfer_learning.keras
Epoch 2: val_accuracy improved from 0.87740 to 0.88500, saving model to CIFAR10_transfer_learning.keras
Epoch 3: val_accuracy did not improve from 0.88500
Epoch 4: val_accuracy improved from 0.88500 to 0.88730, saving model to CIFAR10_transfer_learning.keras
Epoch 5: val_accuracy improved from 0.88730 to 0.89490, saving model to CIFAR10_transfer_learning.keras
Epoch 6: val_accuracy did not improve from 0.89490 ...
Epoch 8: val_accuracy improved from 0.89490 to 0.89560, saving model to CIFAR10_transfer_learning.keras
Epoch 9: val_accuracy did not improve from 0.89560 ...
Epoch 12: val_accuracy improved from 0.89560 to 0.89620, saving model to CIFAR10_transfer_learning.keras
Epoch 13: val_accuracy did not improve from 0.89620
Epoch 14: val_accuracy improved from 0.89620 to 0.89670, saving model to CIFAR10_transfer_learning.keras
Epoch 15: val_accuracy did not improve from 0.89670 ...
Epoch 17: val_accuracy improved from 0.89670 to 0.89790, saving model to CIFAR10_transfer_learning.keras
Epoch 18: val_accuracy did not improve from 0.89790 ...
Epoch 21: val_accuracy improved from 0.89790 to 0.89810, saving model to CIFAR10_transfer_learning.keras
Epoch 22: val_accuracy did not improve from 0.89810
Epoch 23: val_accuracy improved from 0.89810 to 0.90260, saving model to CIFAR10_transfer_learning.keras
Epoch 24: val_accuracy improved from 0.90260 to 0.90300, saving model to CIFAR10_transfer_learning.keras
Epoch 25: val_accuracy did not improve from 0.90300
Epoch 26: val_accuracy improved from 0.90300 to 0.90320, saving model to CIFAR10_transfer_learning.keras
Epoch 27: val_accuracy improved from 0.90320 to 0.90380, saving model to CIFAR10_transfer_learning.keras
Epoch 28: val_accuracy improved from 0.90380 to 0.90520, saving model to CIFAR10_transfer_learning.keras
Epoch 29: val_accuracy did not improve from 0.90520 ...
Epoch 33: val_accuracy improved from 0.90520 to 0.90540, saving model to CIFAR10_transfer_learning.keras
Epoch 34: val_accuracy improved from 0.90540 to 0.90740, saving model to CIFAR10_transfer_learning.keras
Epoch 35: val_accuracy improved from 0.90740 to 0.90750, saving model to CIFAR10_transfer_learning.keras
Epoch 36: val_accuracy did not improve from 0.90750
...
Epoch 44: val_accuracy did not improve from 0.90750
```

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(9, 3.5)); plt.subplots_adjust(wspace=0.3)
xx = np.arange(1, len(H.history['loss'])+1)
ax = axes[0]; ax.grid();  # ax.set_xticks(xx)
ax.set_xlabel('epoch'); ax.set_ylabel('loss')
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_xticks(xx)
ax.set_xlabel('epoch'); ax.set_ylabel('accuracy')
ax.plot(xx, H.history['accuracy'], color='b', label='train')
ax.plot(xx, H.history['val_accuracy'], color='r', label='val'); ax.legend();
```





Evaluación en test:

In [ ]:
```
score = keras.models.load_model(filename).evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4} Precisión: {score[1]:.2%}')
```

```
Loss: 0.6359 Precisión: 90.15%
```

Conclusión: precisión en test mucho mejor que con una CNN sencilla

6 Fine-tuning

Inicialización: librerías, semilla, lectura de CIFAR-10 y partición train-val-test

In [ ]:
```
import numpy as np; import matplotlib.pyplot as plt
import os; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras; from keras import layers
keras.utils.set_random_seed(23)
(x_train_val, y_train_val), (x_test, y_test) = keras.datasets.cifar10.load_data()
x_train_val = x_train_val.astype("float32")
x_test = x_test.astype("float32")
y_train_val = keras.utils.to_categorical(y_train_val, 10)
y_test = keras.utils.to_categorical(y_test, 10)
x_train = x_train_val[:-10000]; x_val = x_train_val[-10000:]
y_train = y_train_val[:-10000]; y_val = y_train_val[-10000:]
print(x_train.shape, y_train.shape, x_val.shape, y_val.shape, x_test.shape, y_test.shape)
```

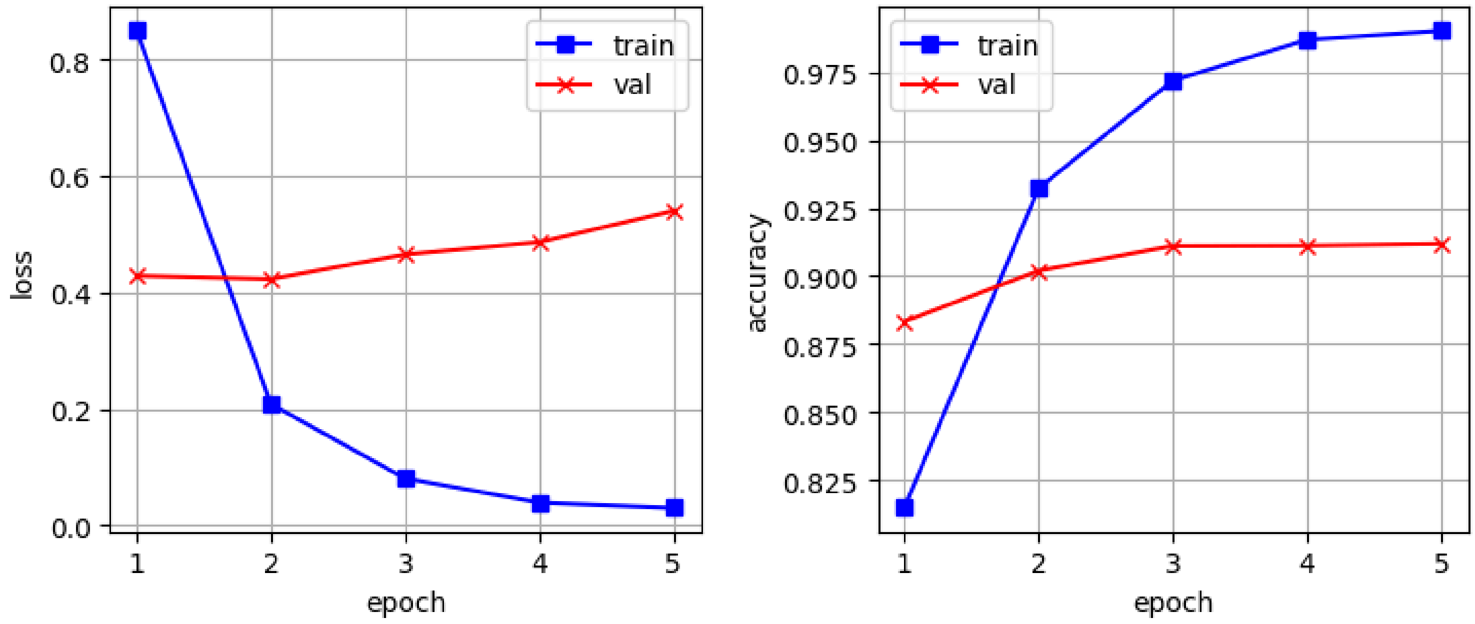
```
(40000, 32, 32, 3) (40000, 10) (10000, 32, 32, 3) (10000, 10) (10000, 32, 32, 3) (10000, 10)
```

Fine-tuning: re-entrenamos muy suavemente todo el modelo salvo capas BatchNorm

In [ ]:
```python
M = keras.models.load_model('CIFAR10_transfer_learning.keras')
M.trainable = True
for layer in M.layers:
    if not isinstance(layer, layers.BatchNormalization):
        layer.trainable = True
opt = keras.optimizers.Adam(learning_rate=1e-5)
M.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
filename = 'CIFAR10_fine-tuning.keras'
checkpoint_cb = keras.callbacks.ModelCheckpoint(
    filepath=filename, monitor='val_accuracy', save_best_only=True, verbose=1)
reduce_cb = keras.callbacks.ReduceLROnPlateau(
    monitor='val_accuracy', factor=0.3, patience=5, min_delta=0.0005, min_lr=0.0)
early_cb = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=10, min_delta=0.0005)
H = M.fit(x_train, y_train, batch_size=32, epochs=5, validation_data=(x_val, y_val), verbose=0,
    callbacks=[checkpoint_cb, early_cb, reduce_cb])
```

```
Epoch 1: val_accuracy improved from -inf to 0.88340, saving model to CIFAR10_fine-tuning.keras

Epoch 2: val_accuracy improved from 0.88340 to 0.90200, saving model to CIFAR10_fine-tuning.keras

Epoch 3: val_accuracy improved from 0.90200 to 0.91110, saving model to CIFAR10_fine-tuning.keras

Epoch 4: val_accuracy improved from 0.91110 to 0.91120, saving model to CIFAR10_fine-tuning.keras

Epoch 5: val_accuracy improved from 0.91120 to 0.91190, saving model to CIFAR10_fine-tuning.keras
```

```
fig, axes = plt.subplots(1, 2, figsize=(9, 3.5)); 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', marker='s', label='train')
ax.plot(xx, H.history['val_loss'], color='r', marker='x', 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', marker='s', label='train')
ax.plot(xx, H.history['val_accuracy'], color='r', marker='x', label='val'); ax.legend();
```



Evaluación en test:

```
score = keras.models.load_model(filename).evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4} Precisión: {score[1]:.2%}')
```

```
Loss: 0.5317 Precisión: 91.12%
```

Conclusión: precisión en test un poco mejor que la obtenida sin fine-tuning

7 Ejercicio

Ejercicio: realiza un experimento similar al de transfer learning con aumento de datos

Inicialización: librerías, semilla, lectura de CIFAR-10 y partición train-val-test

In [ ]:
```
import numpy as np; import matplotlib.pyplot as plt
import os; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras; from keras import layers
keras.utils.set_random_seed(23)
(x_train_val, y_train_val), (x_test, y_test) = keras.datasets.cifar10.load_data()
x_train_val = x_train_val.astype("float32")
x_test = x_test.astype("float32")
y_train_val = keras.utils.to_categorical(y_train_val, 10)
y_test = keras.utils.to_categorical(y_test, 10)
x_train = x_train_val[:-10000]; x_val = x_train_val[-10000:]
y_train = y_train_val[:-10000]; y_val = y_train_val[-10000:]
print(x_train.shape, y_train.shape, x_val.shape, y_val.shape, x_test.shape, y_test.shape)
```

```
(40000, 32, 32, 3) (40000, 10) (10000, 32, 32, 3) (10000, 10) (10000, 32, 32, 3) (10000, 10)
```

Construcción del modelo: añadimos reflexiones horizontales y translaciones

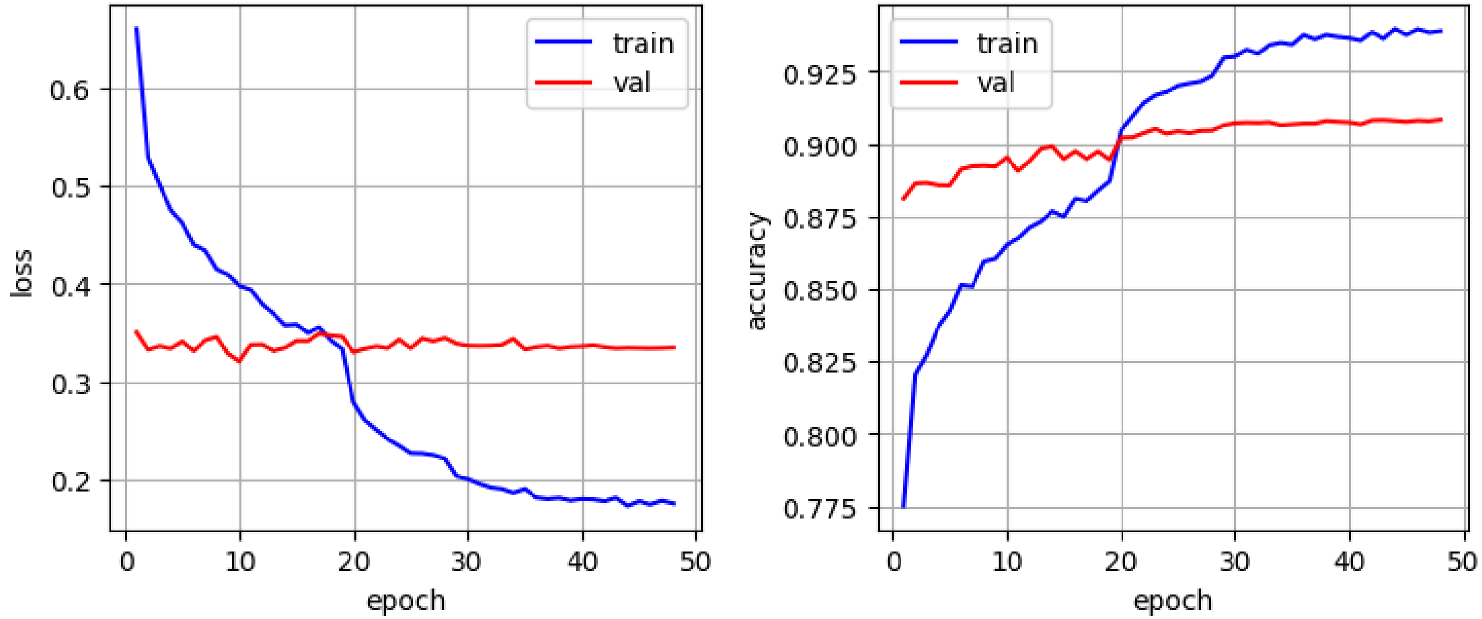
In [ ]:
```
inputs = keras.Input(shape=(32, 32, 3))
x = layers.Rescaling(scale=1 / 127.5, offset=-1)(inputs)
x = layers.RandomFlip(mode="horizontal")(x)
factor = 2.0 / 32.0
x = layers.RandomTranslation(factor, factor, fill_mode="nearest")(x)
x = layers.Resizing(224, 224, interpolation="nearest")(x)
base_M = keras.applications.resnet_v2.ResNet50V2(include_top=False)
base_M.trainable = False
x = base_M(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(units=800, activation='relu')(x)
x = layers.Dropout(0.5)(x)
predictions = 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"])
```

Entrenamiento del modelo: guardamos el mejor modelo con un callback checkpoint

In [ ]:
```
filename = 'CIFAR10_transfer_learning.keras'
checkpoint_cb = keras.callbacks.ModelCheckpoint(
    filepath=filename, monitor='val_accuracy', save_best_only=True, verbose=1)
reduce_cb = keras.callbacks.ReduceLROnPlateau(
    monitor='val_accuracy', factor=0.3, patience=5, min_delta=0.0005, min_lr=0.0)
early_cb = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=10, min_delta=0.0005)
H = M.fit(x_train, y_train, batch_size=32, epochs=500, validation_data=(x_val, y_val), verbose=0,
    callbacks=[checkpoint_cb, early_cb, reduce_cb])
```

```
Epoch 1: val_accuracy improved from -inf to 0.88110, saving model to CIFAR10_transfer_learning.keras
Epoch 2: val_accuracy improved from 0.88110 to 0.88630, saving model to CIFAR10_transfer_learning.keras
Epoch 3: val_accuracy improved from 0.88630 to 0.88650, saving model to CIFAR10_transfer_learning.keras
Epoch 6: val_accuracy improved from 0.88650 to 0.89130, saving model to CIFAR10_transfer_learning.keras
Epoch 7: val_accuracy improved from 0.89130 to 0.89230, saving model to CIFAR10_transfer_learning.keras
Epoch 8: val_accuracy improved from 0.89230 to 0.89250, saving model to CIFAR10_transfer_learning.keras
Epoch 10: val_accuracy improved from 0.89250 to 0.89530, saving model to CIFAR10_transfer_learning.keras
Epoch 13: val_accuracy improved from 0.89530 to 0.89830, saving model to CIFAR10_transfer_learning.keras
Epoch 14: val_accuracy improved from 0.89830 to 0.89910, saving model to CIFAR10_transfer_learning.keras
Epoch 20: val_accuracy improved from 0.89910 to 0.90210, saving model to CIFAR10_transfer_learning.keras
Epoch 21: val_accuracy improved from 0.90210 to 0.90220, saving model to CIFAR10_transfer_learning.keras
Epoch 22: val_accuracy improved from 0.90220 to 0.90370, saving model to CIFAR10_transfer_learning.keras
Epoch 23: val_accuracy improved from 0.90370 to 0.90510, saving model to CIFAR10_transfer_learning.keras
Epoch 29: val_accuracy improved from 0.90510 to 0.90640, saving model to CIFAR10_transfer_learning.keras
Epoch 30: val_accuracy improved from 0.90640 to 0.90710, saving model to CIFAR10_transfer_learning.keras
Epoch 31: val_accuracy improved from 0.90710 to 0.90730, saving model to CIFAR10_transfer_learning.keras
Epoch 33: val_accuracy improved from 0.90730 to 0.90740, saving model to CIFAR10_transfer_learning.keras
Epoch 38: val_accuracy improved from 0.90740 to 0.90790, saving model to CIFAR10_transfer_learning.keras
Epoch 42: val_accuracy improved from 0.90790 to 0.90810, saving model to CIFAR10_transfer_learning.keras
Epoch 43: val_accuracy improved from 0.90810 to 0.90820, saving model to CIFAR10_transfer_learning.keras
Epoch 48: val_accuracy improved from 0.90820 to 0.90830, saving model to CIFAR10_transfer_learning.keras
```

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(9, 3.5)); plt.subplots_adjust(wspace=0.3)
xx = np.arange(1, len(H.history['loss'])+1)
ax = axes[0]; ax.grid(); # ax.set_xticks(xx)
ax.set_xlabel('epoch'); ax.set_ylabel('loss')
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_xticks(xx)
ax.set_xlabel('epoch'); ax.set_ylabel('accuracy')
ax.plot(xx, H.history['accuracy'], color='b', label='train')
ax.plot(xx, H.history['val_accuracy'], color='r', label='val'); ax.legend();
```



Evaluación en test:

In [ ]:
```
score = keras.models.load_model(filename).evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4} Precisión: {score[1]:.2%}')
```

```
Loss: 0.3332 Precisión: 90.79%
```