

Visión por Computador (1782)

Herramientas de programación de aplicaciones OpenCV – Python

DNN Transfer-Learning with Keras

Luis M. Jiménez



Lab. Automática, Robótica y Visión por Computador
Universidad Miguel Hernández
<http://arvc.umh.es>



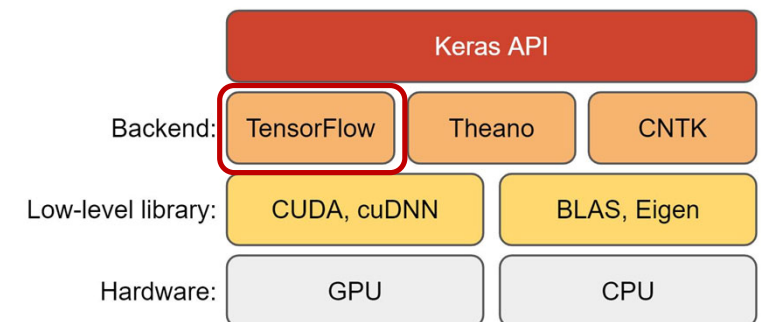
REDES NEURONALES CONVOLUCIONALES (CNN)

Transfer Learning ([keras](https://keras.io))

<https://keras.io>

- Importar redes pre-entrenadas (tensorflow)
- Clasificación: : VGG16, DenseNet121, Inceptionv3
- Modificar capas de salida y reentrenar
- Librerías: **keras-tensorflow**

```
import keras
import tensorflow as tf
```



Disponible como paquete independiente o como parte de tensorflow: **tf.keras**

```
import tensorflow.keras as keras
```

Deep Learning (CNN)

- Paquete **keras** <https://keras.io/api/>

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                 loss='binary_crossentropy',
                 metrics=['accuracy'])
>>> model.fit(data, labels, epochs=10, batch_size=32)
>>> predictions = model.predict(data)
```

Model Training

```
>>> model3.fit(x_train4,
              y_train4,
              batch_size=32,
              epochs=15,
              verbose=1,
              validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Conv2D(64,(3,3),padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Deep Learning (CNN)

- Redes CNN pre-entrenadas Clasificación, formatos (frameworks):
 - Caffe, **Tensorflow**, PyTorch, Darknet, ONNX
- Modelos Tensorflow: tf.keras.applications
 - Xception
 - EfficientNet B0 to B7
 - EfficientNetV2 B0 to B3 and S, M, L
 - VGG16 and VGG19
 - ResNet and ResNetV2
 - MobileNet and MobileNetV2
 - DenseNet
 - NasNetLarge and NasNetMobile
 - InceptionV3
 - InceptionResNetV2
- Datasets:
 - **ImageNet** Large Scale Visual Recognition Challenge (ILSVRC):
 - <https://image-net.org/challenges/LSVRC/>

se descargan desde el mismo código

Transfer Learning (CNN)

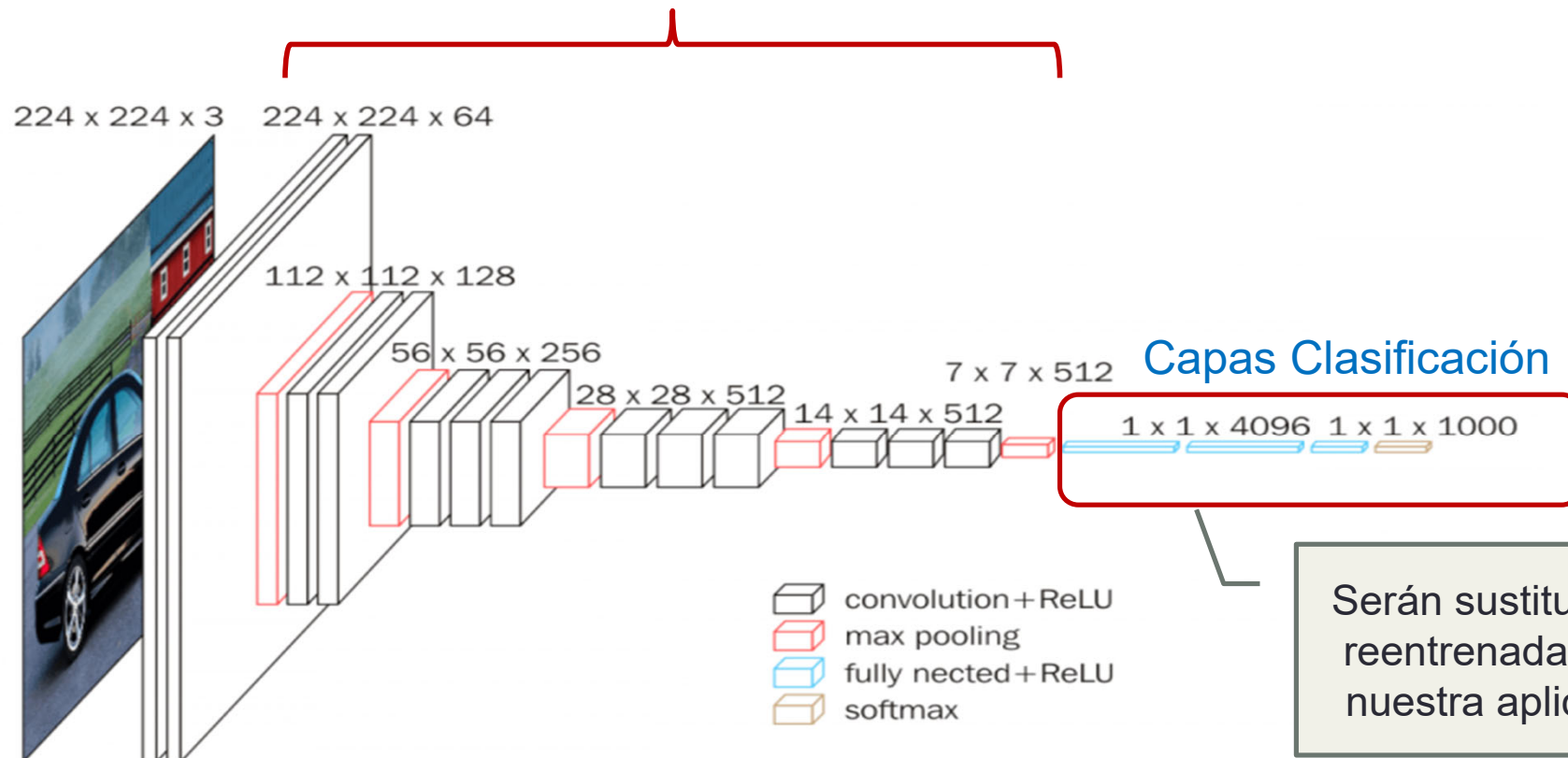
VGG16: (16 capas-138Millones parámetros) - input 224x224x3

“Very Deep Convolutional Networks for Large-Scale Image Recognition”

Karen Simonyan and Andrew Zisserman Univ. Oxford

arXiv 1409.1556 ICLR 2015

Capas Convolucionales: Descriptores

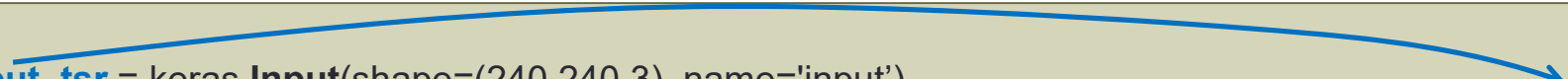


Clases: `keras.models`, `keras.layers`, `keras.Input`

- Tipos de Modelos: `keras.models`

- `keras.models.Model`(inputs, outputs, name) → model
 - Modelo general (Funcional), permite cualquier conexión (bifurcación, múltiples salidas o entradas) o topologías no lineales (Residual-realimentación, multi-rama, ...)
 - La idea básica de `keras` es crear la red como un grafo que va conectando diferentes capas.
 - A cada capa le pasamos como parámetro el tensor de salida de la capa previa usando el operador de **llamada al constructor ()** que nos devolverá el nuevo tensor de salida

```
input_tsr = keras.Input(shape=(240,240,3), name='input')
output_tsr = keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', name='conv1')(input_tsr)
```



- Los datos que se pasan entre capas se denomina **tensores** (*arrays multidimensionales*)
- Un **modelo keras** se crea indicando los tensores de la primera y la última capa

```
model = keras.models.Model( inputs=input_tsr, outputs=output_tsr, name='MyNet')
```

- `keras.models.Sequential`(layers=None) → model
 - Modelo simplificado para redes feed-forward. Cada capa con un solo **tensor** de entrada y de salida
 - Se utiliza el **método add()** para conectar capas o bien se pasan como una lista al constructor
 - `keras.models.Sequential.add`(layer) añade la capa al final
 - `keras.models.Sequential.pop`() elimina la última capa

Keras: Layers

<https://keras.io/api/layers/>

- Capas: [keras.layers](#)

- Clase base: [keras.layers.Layer](#)

- Clase base, el resto de capas deriva de esta
- Componentes:
 - **trainable** → (bool) indica si los pesos de la capa deben ser entrenados
lo pondremos a False para las capas ya entrenadas a reutilizar.
 - **dtype** → tipo de datos de los pesos ('float32')
 - **name** → nombre de la capa
 - **input_shape** → dimensiones tensor entrada
 - **output_shape** → dimensiones tensor de salida
 - **input** → tensor de entrada de la capa
 - **output** → tensor de salida de la capa

Métodos:

- [Layer.get_weights\(\)](#) → numpy array
- [Layer.set_weights\(weights\)](#)

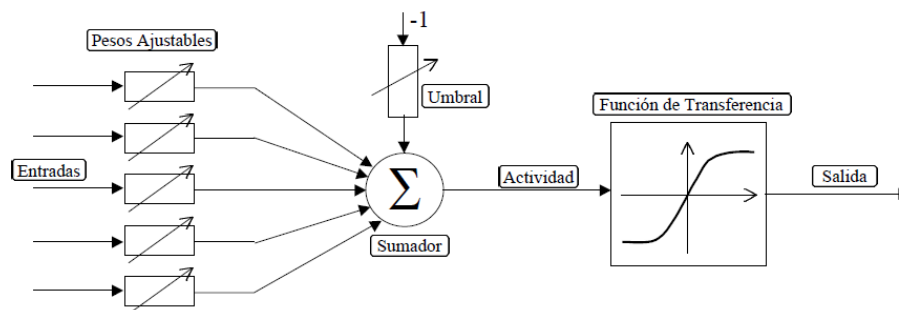
- Capas base:

- [keras.Input\(shape, name, dtype\)](#) → crea un tensor de entrada
shape: una tupla (integers), sin incluir el tamaño del lote/batch: shape=(32,)
Si **data_format**='channels_last' (defecto) (h,w,chan) / 'channels_first' (chan,h,w)
dtype: tipo de datos de la entrada input, como un string ('float32', 'float64', 'int32'...)
- [keras.layers.Activation\(activation\)](#) → Capa solo con función de activación (sin pesos)
- [keras.layers.Dense\(units, activation, name\)](#) → Perceptrón estándar (totalmente conectado)
units: número de neuronas de la capa

Keras: models

<https://keras.io/api/layers/activations/>

- Activación de capas:
 - Como parámetro de la capa:
 - `keras.layers.Conv2D(activation='relu')` → layer
 - Capa Independiente:
 - `keras.layers.Activation(activation)` → layer

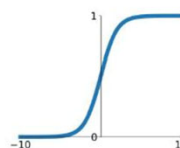


Layer activations

- relu function
- sigmoid function
- softmax function
- softplus function
- softsign function
- tanh function
- selu function
- elu function
- exponential function
- leaky_relu function
- relu6 function
- silu function
- hard_silu function
- gelu function
- hard_sigmoid function
- linear function
- mish function
- log_softmax function

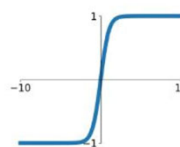
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



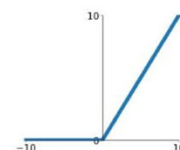
tanh

$$\tanh(x)$$



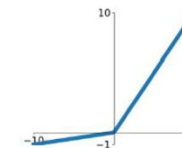
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

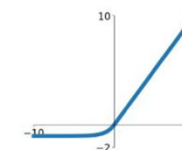


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



$$\text{softmax: } p(i) = \frac{e^{z(i)}}{\sum_j e^{z(j)}}$$

$$\text{softplus} = \log(e^x + 1)$$

$$\text{softsign} = \frac{x}{|x| + 1}$$

$$\text{exponential} = e^x$$

$$\text{selu}(z) = \begin{cases} s * x & \text{si } x \geq 0 \\ s * \alpha(e^x - 1) & \text{si } x < 0 \end{cases}$$

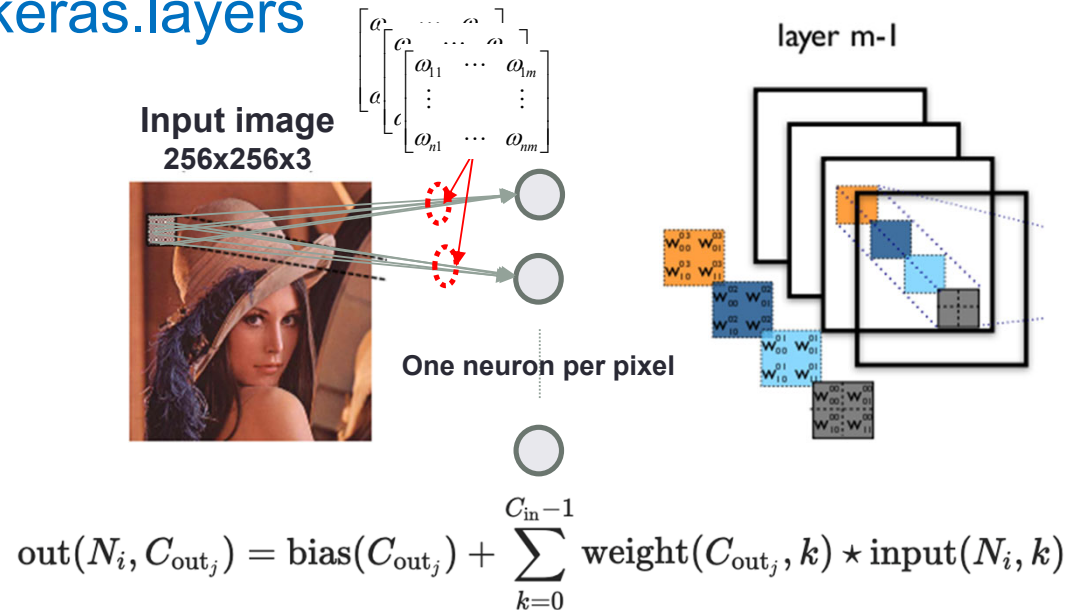
$$\text{sgn}(z) = \begin{cases} +1 & \text{si } x > 0 \\ 0 & \text{si } x = 0 \\ -1 & \text{si } x < 0 \end{cases}$$

Keras: Layers

https://keras.io/api/layers/convolution_layers/

• Capas convolucionales : `keras.layers`

- `keras.layers.Conv1D`
- `keras.layers.Conv2D`
- `keras.layers.Conv3D`
- `keras.layers.SeparableConv1D`
- `keras.layers.SeparableConv2D`
- `keras.layers.DepthwiseConv1D`
- `keras.layers.DepthwiseConv2D`
- `keras.layers.Conv1DTranspose`
- `keras.layers.Conv2DTranspose`
- `keras.layers.Conv3DTranspose`



`keras.layers.Conv2D`(`filters`, `kernel_size`, `strides`=(1, 1), `padding`="valid", `dilation_rate`=(1, 1), `activation`=None, `data_format`=None, `use_bias`=True, `groups`=1)

filters: numero de conjunto de pesos distintos: niveles de la capa (dimensión de la salida)

kernel_size: (h,w) tamaño de la matriz de pesos

strides=(1, 1): paso desplazamiento de la convolución en altura y anchura por la imagen

padding: ('valid', 'same') *valid*: elimina pixels de borde (mascara fuera de la imagen)

same: rellena con ceros los pixels de la máscara fuera de la imagen

dilation_rate=(1, 1): espacios entre pixels a los que se aplica la convolución

activation = None

groups=1: grupos del canales de la imagen de la entrada que se procesan separadamente

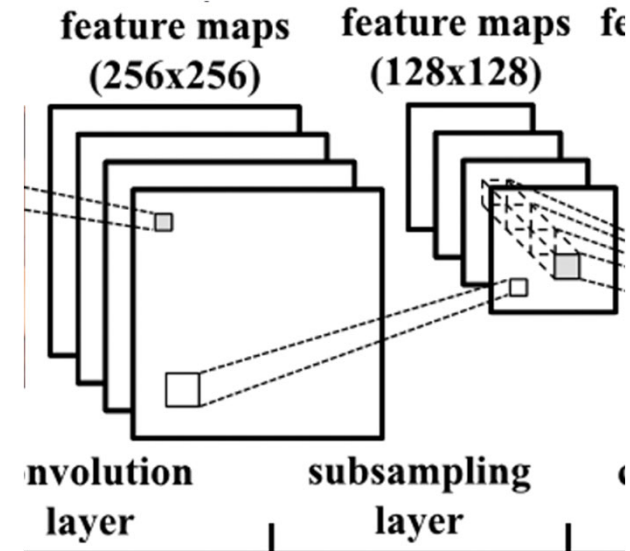
data_format: string 'channels_last' (default) (h,w,chan) or 'channels_first' (chan,h,w)

Orden de las dimensiones de la entrada

Keras: Layers

https://keras.io/api/layers/pooling_layers/

- Capas: `keras.layers`
 - Capas muestreo:
 - `keras.layers.MaxPooling1D`
 - `keras.layers.MaxPooling2D`
 - `keras.layers.MaxPooling3D`
 - `keras.layers.AveragePooling1D`
 - `keras.layers.AveragePooling2D`
 - `keras.layers.AveragePooling3D`
 - `keras.layers.GlobalMaxPooling1D`
 - `keras.layers.GlobalMaxPooling2D`
 - `keras.layers.GlobalMaxPooling3D`
 - `keras.layers.GlobalAveragePooling1D`
 - `keras.layers.GlobalAveragePooling2D`
 - `keras.layers.GlobalAveragePooling3D`



- `keras.layers.MaxPooling2D`(`pool_size=(2, 2)`, `strides=None`, `padding="valid"`, `data_format=None`)

pool_size=(2, 2): ventana en la que calcular el máximo (muestreo vertical y horizontal)

strides=(1, 1): paso desplazamiento de la convolución en altura y anchura por la imagen

padding: ('valid', 'same') valid: elimina pixels de borde (mascara fuera de la imagen)

same: rellena con ceros los pixels de la máscara fuera de la imagen

data_format: string '`channels_last`' (default) (h,w,chan) or '`channels_first`' (chan,h,w)

Orden de las dimensiones de la entrada

Keras: Layers

https://keras.io/api/layers/regularization_layers/

- Capas: [keras.layers](#)

- Capas Realimentadas (Recurrentes):

- [keras.layers.RNN](#) clase base
- [keras.layers.LSTM](#)
- [keras.layers.GRU](#)
- [keras.layers.SimpleRNN](#)
- [keras.layers.TimeDistributed](#)
- [keras.layers.Bidirectional](#)
- [keras.layers.ConvLSTM1D](#)
- [keras.layers.ConvLSTM2D](#)
- [keras.layers.ConvLSTM3D](#)

- Capas Regularización: (solo actúan en el entrenamiento)

- [keras.layers.Dropout](#)
- [keras.layers.SpatialDropout1D](#)
- [keras.layers.SpatialDropout2D](#)
- [keras.layers.SpatialDropout3D](#)
- [keras.layers.GaussianDropout](#)
- [keras.layers.GaussianNoise](#)
- [keras.layers.ActivityRegularization](#)
- [keras.layers.AlphaDropout](#)

- [keras.layers.Dropout](#)(**rate**, noise_shape=None, seed=None)

Pone aleatoriamente entradas (salida neuronas capa previa) a **0** con frecuencia **rate** en cada paso de la fase de entrenamiento. Reescala el resto para que la suma se mantenga.

Permite evitar el Sobreajuste 'Overfitting'

- También se puede configurar el regularizador en cada capa (parámetro):

- kernel_regularizer = [keras.regularizers.l1_l2](#)(l1=1e-5, l2=1e-4),
- bias_regularizer = [keras.regularizers.l2](#)(1e-4),
- activity_regularizer = [keras.regularizers.l2](#)(1e-5)

Keras: Layers

https://keras.io/api/layers/reshaping_layers/

- Capas: `keras.layers`

- Capas Concatenación:

- `keras.layers.Concatenate`
- `keras.layers.Average`
- `keras.layers.Maximum`
- `keras.layers.Minimum`
- `keras.layers.Add`
- `keras.layers.Subtract`
- `keras.layers.Multiply`
- `keras.layers.Dot`

- Capas Atención

- `keras.layers.GroupQueryAttention`
- `keras.layers.MultiHeadAttention`
- `keras.layers.Attention`
- `keras.layers.AdditiveAttention`

- Capas Redimensionado

- `keras.layers.Reshape`
- `keras.layers.Flatten`
- `keras.layers.RepeatVector`
- `keras.layers.RandomRotation`
- `keras.layers.Permute`
- `keras.layers.RandomHeight`
- `keras.layers.Cropping1D`
- `keras.layers.Cropping2D`
- `keras.layers.Cropping3D`
- `keras.layers.UpSampling1D`
- `keras.layers.UpSampling2D`
- `keras.layers.UpSampling3D`
- `keras.layers.ZeroPadding1D`
- `keras.layers.ZeroPadding2D`
- `keras.layers.ZeroPadding3D`

- `keras.layers.Flatten(data_format=None)`

Convierte capas convolucionales en vectoriales (planas)

Keras: Layers

https://keras.io/api/layers/preprocessing_layers/

- Capas: `keras.layers`

- Capas Preprocesamiento y Normalización:

- `keras.layers.Resizing`(height, width, interpolation, crop_to_aspect_ratio=False)
- `keras.layers.Rescaling`(scale, offset=0.0)
- `keras.layers.CenterCrop`(height, width)
- `keras.layers.BatchNormalization`()
- `keras.layers.LayerNormalization`()
- `keras.layers.UnitNormalization`()
- `keras.layers.GroupNormalization`()

- Capas aumento de datos 'Data-Augmentation':

- Solo actúan en el entrenamiento
- La transformación es la misma para todas las imágenes en el mismo lote (**batch**)
- Se pasaría el lote múltiples veces para entrenar la red (aumento de datos)
- `keras.layers.RandomCrop`
- `keras.layers.RandomFlip`
- `keras.layers.RandomTranslation`
- `keras.layers.RandomRotation`
- `keras.layers.RandomZoom`
- `keras.layers.RandomHeight`
- `keras.layers.RandomWidth`
- `keras.layers.RandomContrast`

Keras: models

- Ejemplo Modelos Funcionales: [keras.models](#)
 - `keras.models.Model(inputs, outputs, name) → model`
 - La idea básica de **keras** es crear la red como un grafo que va conectando diferentes capas.
 - A cada capa le pasamos como parámetro el tensor de salida de la capa previa usando el operador de **llamada al constructor ()** que nos devolverá el nuevo tensor de salida
 - Los datos que se pasan entre capas se denomina **tensores** (*arrays multidimensionales*)
 - Un **modelo keras** se crea indicando los tensores de la primera y la última capa
 - Por ultimo debemos '**compilarlo**', seleccionado el tipo de función de coste, el optimizador y la métricas de evaluación para entrenar la red

```
import keras
import tensorflow as tf

input_tsr = keras.Input(shape=(240,240,3), name='input')
layer_tsr = keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu', name='conv1')(input_tsr)
layer_tsr = keras.layers.MaxPooling2D(pool_size=(2, 2), name='pooling1')(layer_tsr)
layer_tsr = keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', name='conv2')(layer_tsr)
layer_tsr = keras.layers.MaxPooling2D(pool_size=(2, 2), name='pooling2')(layer_tsr)
layer_tsr = keras.layers.Flatten(name='flatten')(layer_tsr)
layer_tsr = keras.layers.Dropout(rate=0.2)(layer_tsr)

output_tsr = keras.layers.Dense(units=20, activation='softmax', name='output')(layer_tsr)

model = keras.models.Model( inputs=input_tsr, outputs=output_tsr, name='MyNet')
model.compile( loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'] )
model.summary()          # prints model summary
```

Keras: models

- Ejemplo:

Model: "MyNet"

Layer (type)	Output Shape	Param #
input (InputLayer)	[(None, 240, 240, 3)]	0
conv1 (Conv2D)	(None, 238, 238, 64)	1792
pooling1 (MaxPooling2D)	(None, 119, 119, 64)	0
conv2 (Conv2D)	(None, 117, 117, 32)	18464
pooling2 (MaxPooling2D)	(None, 58, 58, 32)	0
flatten (Flatten)	(None, 107648)	0
dropout (Dropout)	(None, 107648)	0
output (Dense)	(None, 20)	2152980

Total params: 2,173,236

Trainable params: 2,173,236

Non-trainable params: 0

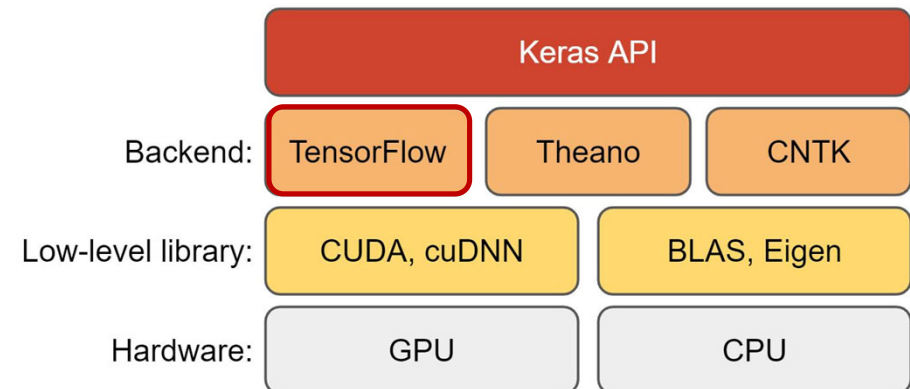
KERAS

Ejemplo Clasificador KERAS (ek1.py):

- Modelo secuencial
- Red Neuronal MLP (Perceptrón Multicapa)
- Datos entrenamiento práctica 4 TITERE

```
import keras
import tensorflow as tf
import titere
```

<https://keras.io>



Keras: models

<https://keras.io/api/models/>

- Ejemplo Modelos Secuenciales: [keras.models](#)
 - `keras.models.Sequential(layers=None)` → model
 - Se utiliza el **método** `add()` para conectar capas o bien se pasan como una lista al constructor
 - `keras.models.Sequential.add(layer)` añade la capa al final
 - `keras.models.Sequential.pop()` elimina la última capa
 - La especificación (shape) de la entrada: `keras.Input()` es opcional (cualquier dimensión)
 - El modelo debemos '**compilarlo**', seleccionando el tipo de función de coste, el optimizador y la métricas de evaluación para entrenar la red

```
import keras
import tensorflow as tf

numFeatures = 4
numClasses = 4

model = keras.models.Sequential()

model.add( keras.Input(shape=(numFeatures,), name='input', dtype='float32') )
model.add( keras.layers.Dense(units=20, activation='tanh', name='hidden') )
model.add( keras.layers.Dense(units=numClasses, activation='softmax', name='output') )

model.compile( loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'] )
model.summary()            # prints model summary
```

Keras: models

<https://keras.io/api/optimizers/>

- Configuración modelo: método **compile()** [keras.models.Model](#)
 - Selecciona el tipo de función de coste, el optimizador y la métricas de evaluación para entrenar la red
 - [keras.models.Model.compile\(optimizer, loss, metrics, loss_weights, weighted_metrics, run_eagerly, steps_per_execution, jit_compile\)](#)

`model.compile(optimizer =sgd')` **optimizer:** string / objeto keras

- **SGD** `'sgd'` Gradient descent (with momentum) optimizer
- **RMSprop** `'rmsprop'` Root Mean Square Propagation
- **Adam** `'adam'` (SGD) Adaptive estimation of first-order and second-order moments
- **Adadelta** `'adadelta'` (SGD-based) Adaptive learning rate per dimension
- **Adagrad** `'adagrad'` (SGD-based) Adaptive Gradient Algorithm
- **Adamax** `'adamax'` Adam with infinite norm (max)
- **Nadam** `'nadam'` Nesterov-accelerated Adaptive Moment Estimation,
- **Ftrl** `'ftrl'` Follow The Regularized Leader

Keras: models

<https://keras.io/api/metrics/>

- Funciones de coste entrenamiento: 'loss'

- `model.compile(metrics=['accuracy'])`

metrics: list(string / objeto keras)

Regression metrics

- MeanSquaredError class
- RootMeanSquaredError class
- MeanAbsoluteError class
- MeanAbsolutePercentageError class
- MeanSquaredLogarithmicError class
- CosineSimilarity class
- LogCoshError class

Classification metrics based on True/False

- AUC class ROC
- Precision class
- Recall class
- TruePositives class
- TrueNegatives class
- FalsePositives class
- FalseNegatives class
- PrecisionAtRecall class
- SensitivityAtSpecificity class
- SpecificityAtSensitivity class

Clasificación Binaria

Accuracy metrics

- Accuracy class
- BinaryAccuracy class
- CategoricalAccuracy class
- SparseCategoricalAccuracy class
- TopKCategoricalAccuracy class
- SparseTopKCategoricalAccuracy class

Probabilistic metrics

- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- KLDivergence class
- Poisson class

Keras: models

https://keras.io/api/models/model_training_apis/

- Funciones de coste entrenamiento: **'loss'**

- `model.compile(loss='categorical_crossentropy')` **loss**: string / objeto keras

Regression losses

- MeanSquaredError class
- MeanAbsoluteError class
- MeanAbsolutePercentageError class
- MeanSquaredLogarithmicError class
- CosineSimilarity class
- `mean_squared_error` function
- `mean_absolute_error` function
- `mean_absolute_percentage_error` function
- `mean_squared_logarithmic_error` function
- `cosine_similarity` function
- Huber class
- `huber` function
- LogCosh class
- `log_cosh` function

Probabilistic losses

- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- Poisson class
- `binary_crossentropy` function
- `categorical_crossentropy` function
- `sparse_categorical_crossentropy` function
- `poisson` function
- KLDivergence class
- `kl_divergence` function

Hinge losses for "maximum-margin" classification

- Hinge class
- SquaredHinge class
- CategoricalHinge class
- `hinge` function
- `squared_hinge` function
- `categorical_hinge` function

Keras: models

<https://keras.io/api/models/>

- Desensambado: `keras.models.Model`

- `keras.models.Model.layers` → list of layers Almacena las capas del modelo
- `keras.models.Model.get_layer(name=None, index=None)` → layer
Devuelve la capa por nombre o índice

```
# Disassemble model layers
layer_names = [ layer.name for layer in model.layers ]
layers = dict( [ (layer.name, layer) for layer in model.layers] )
```

- Nos permitirá modificar una red previa o evaluar la respuesta de una capa
 - `layer.input_shape` → dimensiones tensor entrada
 - `layer.output_shape` → dimensiones tensor de salida
 - `layer.input` → tensor de entrada de la capa
 - `layer.output` → tensor de salida de la capa

```
in_tsr = layers['hidden'].input
out_tsr = layers['hidden'].output

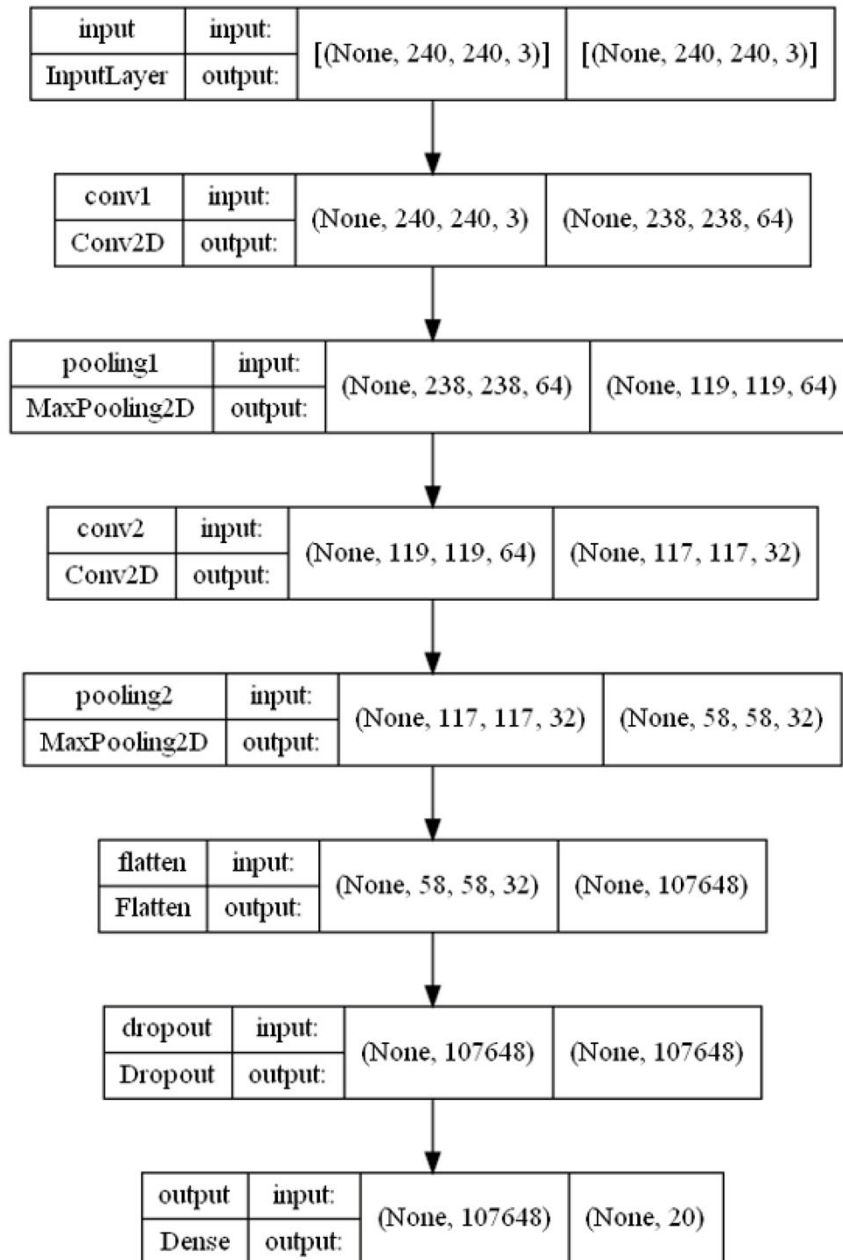
print(f"layer input shape:", in_tsr.shape)
print(f"layer output shape:", out_tsr.shape)
```

- Utilidades de visualización:

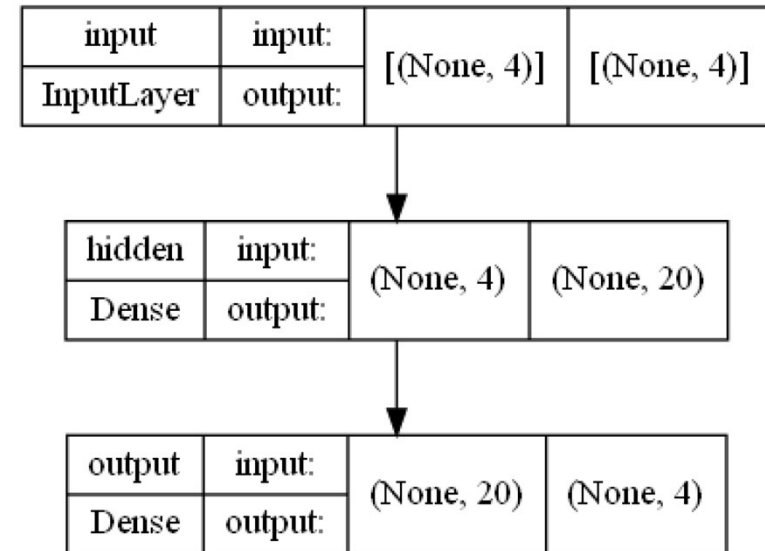
- `model.summary()` → imprime el resumen del modelo (capas)
- `keras.utils.plot_model(model, to_file='net.png', show_shapes=True)`
 - Requiere `pydot` y `graphviz` instalado (<https://graphviz.gitlab.io/download/>)

Keras: models: [keras.utils.plot_model](#)

net1



net2



Keras: models

https://keras.io/api/models/model_training_apis/

- Entrenamiento: método **fit()** `keras.models.Model`

- `keras.models.Model.fit(x, y, batch_size=None, epochs=1, verbose="auto", shuffle=True, validation_split=0.0, validation_data=None) → history`

history: **History** object, curvas de aprendizaje ('loss', 'accuracy')

x: tensor de entrada, matriz de datos → fila: muestras, columnas: entradas

y: tensor de salidas (etiquetas), matriz de datos → fila: muestras, una columna por salida

batch_size: (int or None) muestras por cada actualización del gradiente

epochs: iteraciones del entrenamiento con todas las muestras

verbose: 'auto', 0, 1, or 2 0= silent, 1= progress bar, 2= one line per epoch

shuffle: (bool) True: barajar los datos (filas) en cada epoch

validation_split: float [0-1] fracción del conjunto de datos de entrenamiento usado para validación al final de cada epoch

validation_data: (x_val, y_val) conjunto datos para validación al final de cada epoch

- `keras.utils.to_categorical(y, num_classes=None) → matrix`

Vector de etiquetas a una matriz categórica → fila: muestras, una columna por clase

```
import keras
import numpy as np
import time
```

```
DATA_FILE = 'train.arff'            # default data file
TEST_FILE = 'test.arff'            # default data file
```

Keras: models

https://keras.io/api/models/model_training_apis/

- Entrenamiento: método `fit()` `keras.models.Model`

```
# import titere reduced data with only 4 colums: ('Compacidad', 'Excentricidad', 'Rel_Invar_1', 'Rel_Invar_2'))
trainDataDict = titere.readWekaDataTitere(DATA_FILE)
trainData = trainDataDict['dataMat']
trainLabels = trainDataDict['label']          # vector with numeric labels (starts at 1)
labelRange = trainDataDict['labelRange']     # list with the numeric/string label association

testDataDict = titere.readWekaDataTitere(TEST_FILE, labelRange)
testData = testDataDict['dataMat']
testLabels = testDataDict['label']

numFeatures = trainData.shape[-1]   # numFeatures trainData last dimension
numClasses = len(labelRange)

# adapt trainLabels/testLabels vector to a float32 matrix with one colum per class
trainLabels = keras.utils.to_categorical(trainLabels-1, num_classes=numClasses )
testLabels = keras.utils.to_categorical(testLabels-1, num_classes=numClasses )

# train the network
train_res = model.fit(trainData, trainLabels, epochs=100, batch_size=1, verbose=1,
                      validation_data=(testData, testLabels))
```

$$y \Rightarrow [y]$$

$$\begin{bmatrix} 0 \\ 2 \\ 1 \\ 3 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
Epoch 97/100
24/24 [=====] - 0s 2ms/step - loss: 0.4925 - accuracy: 1.0000 - val_loss: 0.4135 - val_accuracy: 1.0000
Epoch 98/100
24/24 [=====] - 0s 2ms/step - loss: 0.4881 - accuracy: 1.0000 - val_loss: 0.4104 - val_accuracy: 1.0000
Epoch 99/100
24/24 [=====] - 0s 2ms/step - loss: 0.4831 - accuracy: 1.0000 - val_loss: 0.4055 - val_accuracy: 1.0000
Epoch 100/100
24/24 [=====] - 0s 2ms/step - loss: 0.4788 - accuracy: 1.0000 - val_loss: 0.4005 - val_accuracy: 1.0000
```

Keras: models

https://keras.io/api/models/model_training_apis/

- Visualizar historial de entrenamiento: Objeto tipo `keras.History`
 - `trainHistory.history` → dict ('accuracy', 'loss', 'val_accuracy', 'val_loss')
 - `matplotlib.pyplot` : misma sintaxis que Matlab
 - Función `plotLearningCurves` disponible en el paquete `titere`

```
import matplotlib.pyplot as plt
import matplotlib

def plotLearningCurves(metrics):

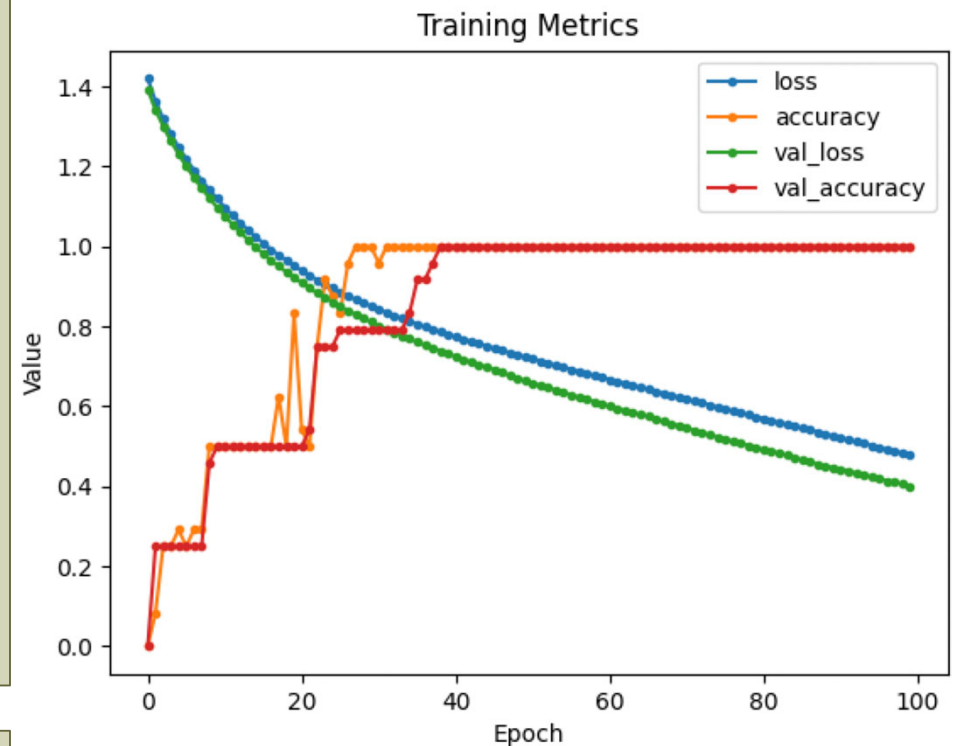
    plt.ion() #interactive mode, non blocking plots

    for metric_name, values in metrics.items():
        plt.plot(values, '-.', label=metric_name)

    plt.title('Training Metrics')
    plt.xlabel('Epoch')
    plt.ylabel('Value')
    plt.legend(loc='best')
    plt.pause(0.1) # non-blocking window update

# end of plotLearningCurves function
```

```
titere.plotLearningCurves(history)
```



Keras: models

https://keras.io/api/models/model_training_apis/

- Evaluación: método **evaluate()** [keras.models.Model](#)
 - [keras.models.Model.evaluate](#)(**x**, **y**, batch_size=None, verbose=1) → loss_and_metrics
 - **x**: tensor de entrada, matriz de datos → fila: muestras, columnas: entradas
 - **y**: tensor de salidas (etiquetas), matriz de datos → fila: muestras, una columna por salida

```
# Evaluate classifier
loss_and_metrics = model.evaluate(testData, testLabels)
print(f"Loss and accuracy: ", loss_and_metrics)
```

```
Loss and accuracy: [0.40046897530555725, 1.0]
```

Keras: models

https://keras.io/api/models/model_training_apis/

- Predicción: método `predict()` `keras.models.Model`
 - `keras.models.Model.predict(x) → y`
 - `x`: tensor de entrada, matriz de datos → fila: muestras, columnas: entradas
 - `y`: tensor de salidas (etiquetas), matriz de datos → fila: muestras, una columna por salida

```
# Prediction
classes = model.predict(testData)

# convert predictionMat to vector: search for max response across classes (columns)
prediction = classes.argmax(axis=1) + 1
prob = classes.max(axis=1)

print(f"Predicted: ", prediction)
print(f"Real:     ", testLabels.argmax(axis=1) + 1)
print(f"Prob:     ", prob)
```

```
Predicted: [2 2 2 2 2 2 1 1 1 1 1 1 3 3 3 3 3 4 4 4 4 4 4]
Actual:    [2 2 2 2 2 2 1 1 1 1 1 1 3 3 3 3 3 4 4 4 4 4 4]
Prob:      [0.38526383 0.38290054 0.383272  0.38313437 0.39126816 0.3667419
0.39058518 0.38580078 0.38512275 0.38695583 0.38495722 0.39017776
0.94704336 0.93950456 0.9411299  0.94533074 0.9429094  0.95257026
0.51671547 0.52243274 0.5266937  0.5166282  0.52617544 0.48555493]
```

REDES NEURONALES CONVOLUCIONALES (CNN)

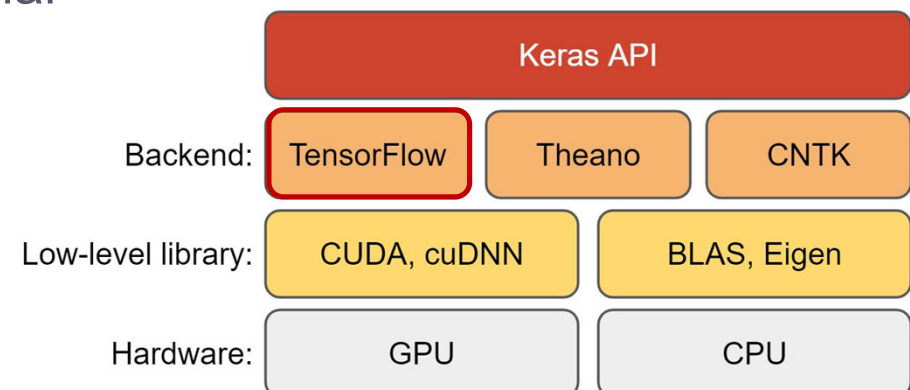
Transfer Learning ([keras](https://keras.io)) (ek2.py)

<https://keras.io>

- Importar redes pre-entrenadas (tensorflow)
- Clasificación: VGG16, DenseNet121, Inceptionv3
- Modificar capas de salida y reentrenar
- Librerías: **keras-tensorflow**

```
import keras
import tensorflow as tf
```

```
tf.keras.applications
```



Transfer Learning (CNN)

<https://keras.io/api/applications/>

- Redes CNN pre-entrenadas Clasificación, formatos (frameworks):
 - Caffe, **Tensorflow**, PyTorch, Darknet, ONNX
- Modelos Tensorflow-Keras: [tf.keras.applications](https://keras.io/api/applications/). (HDF5 file)
 - Xception
 - EfficientNet B0 to B7
 - EfficientNetV2 B0 to B3 and S, M, L
 - **VGG16** and VGG19
 - ResNet and ResNetV2. (101,152)
 - MobileNet and MobileNetV2
 - **DenseNet121**, DenseNet169, DenseNet201
 - NasNetLarge and NasNetMobile
 - **InceptionV3**
 - InceptionResNetV2
- Datasets:
 - **ImageNet** Large Scale Visual Recognition Challenge (ILSVRC):
 - <https://image-net.org/challenges/LSVRC/>



solo disponible
en **tf.keras**

Transfer Learning (CNN)

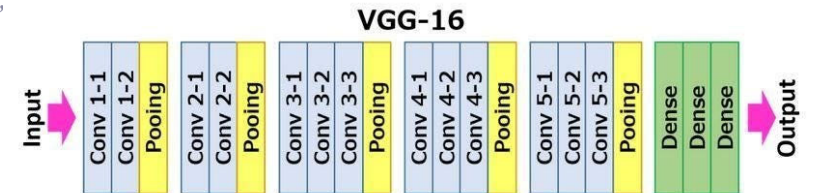
<https://keras.io/api/applications/>

- Modelos Tensorflow: [tf.keras.applications](#)
 - [tf.keras.applications.VGG16](#)(include_top=True, weights="imagenet", input_tensor=None, input_shape=None, pooling=None, classes=1000) → modelo
 - include_top**: True → incluye capas finales clasificación
False → No incluye las capas finales de clasificación (depende de la red)
 - weights**: 'imagenet' or 'None' → random weights
 - input_shape**: dimensiones entrada si **include_top == False** (≤ dim. entrada red orig.)
'channels_last' (h,w,chan) (def) / 'channels_first' (chan,h,w)
 - input_tensor**: tensor de entrada opcional si **include_top == False**
or (shape == dim. entrada red orig.)
 - [tf.keras.applications.vgg16.preprocess_input](#)(x, data_format=None) → tensor/numpy
escalado de entradas específico de la red [scale*(X-mean)] (numpy → np.copy(x))
 - [tf.keras.applications.vgg16.decode_predictions](#)(preds, top=5) → list of tuples
(class_name, class_description, score)
- Leer/Guardar ficheros modelo en formato Tensorflow-Keras (*.h5)
 - [keras.models.load_model](#)(filepath, custom_objects=None, compile=True, options=None) → modelo
<http://umh1782.umh.es/python> (HDF5 file)
 - [keras.models.save_model](#)(model, filepath)
 - Método clase **Model**: [keras.models.Model.save](#)(filepath)

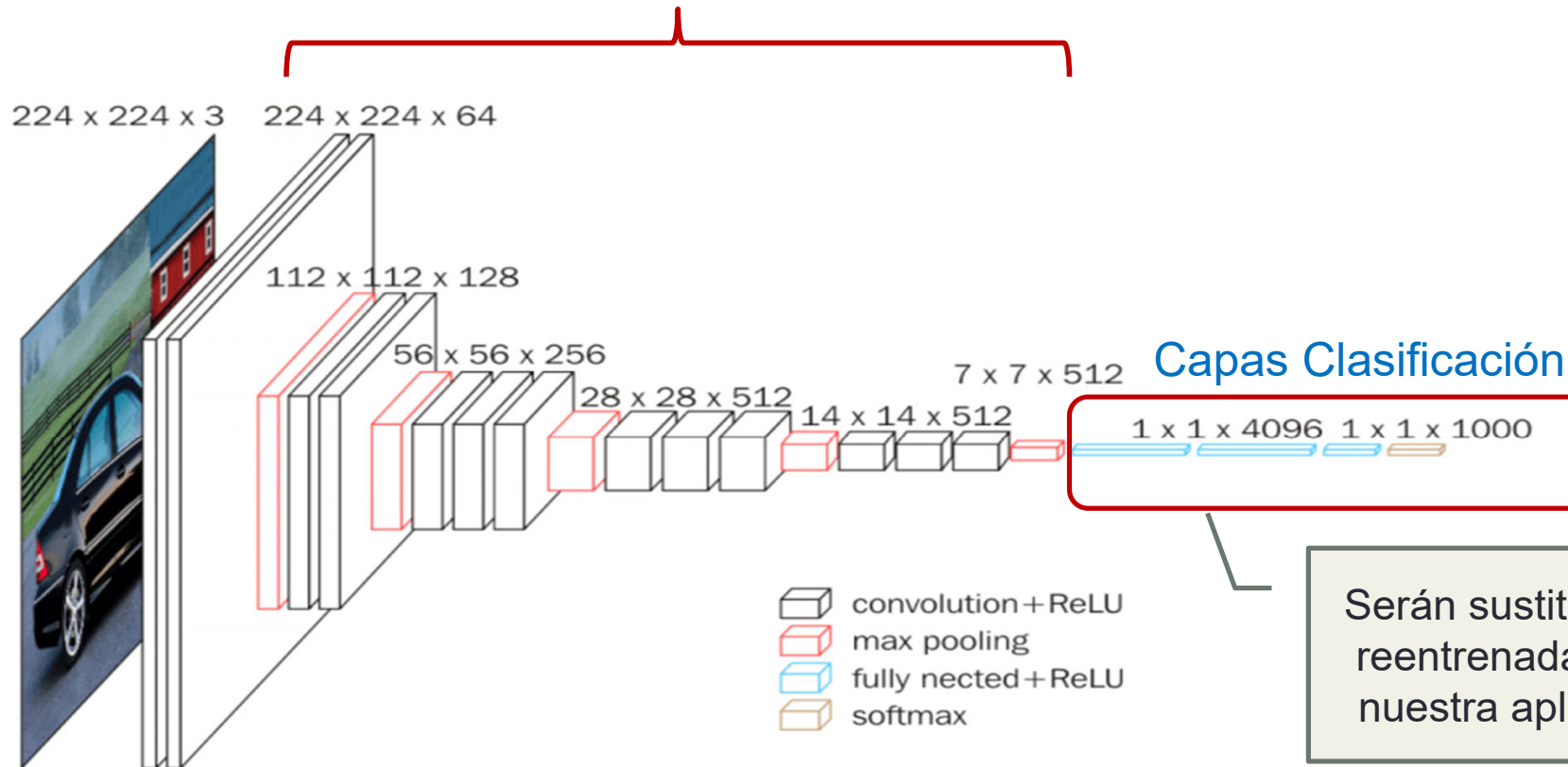
Transfer Learning (CNN)

VGG16: (16 capas-138 Millones de parámetros) - input 224x224x3

“Very Deep Convolutional Networks for Large-Scale Image Recognition”
 Karen Simonyan and Andrew Zisserman Univ. Oxford
 arXiv 1409.1556 ICLR 2015



Capas Convolucionales: Descriptores



Serán sustituidas y reentrenadas para nuestra aplicación

Transfer Learning (CNN)

<https://keras.io/api/applications/>

- Cargar Modelo Tensorflow: `tf.keras.applications.keras.models.load_model()`
- Configurar orden de canales tensor: **def**: 'channels_last' (n,h,w,c), **opencv**: 'channels_first' (n,c,h,w)
 - `keras.backend.set_image_data_format(data_format)`
 - `keras.backend.image_data_format(data_format) → data_format (string)`

```
# change input tensor axis order: keras: (n,h,w,c) / opencv blob format is (n,c,h,w) 'channels_first'  
# VGG16 is trained with (n,h,w,c) 'channels_last' order  
keras.backend.set_image_data_format('channels_last')
```

```
# download model (base network)  
model = tf.keras.applications.VGG16()
```

```
# alternatively we can use pre-downloaded models (http://umh1782.umh.es/python)  
# model = keras.models.load_model('../tensorflow-keras/vgg16.h5')
```

```
# show model summary  
model.summary()
```

```
# sets all layers for imported model not trainable (weights won't be trained)  
model.trainable = False
```

```
# Build new model
```

Transfer Learning (CNN)

<https://keras.io/api/applications/>

- Cargar Modelo Tensorflow:

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Capas Convolucionales
Descriptores

layer.trainable==False

Serán sustituidas y
reentrenadas para
nuestra aplicación

Transfer Learning (CNN)

<https://keras.io/api/layers/>

- Modificar Modelo Tensorflow: [keras.layers](#)

```
# Build new model
# get base network input tensor (first layer)
base_in_tsr = model.get_layer(index=0).output

# get network Flatten-Conv output tensor
base_out_tsr = model.get_layer(name='flatten').output

# get input image size (n,h,w,c) / (n,c,h,w)
if keras.backend.image_data_format() == 'channels_first':
    image_size = tuple(base_in_tsr.shape)[2:4]
else:
    image_size = tuple(base_in_tsr.shape)[1:3]

# add new classification layers (trainable == True) default
layer_tsr = keras.layers.Dropout(rate=0.2)(base_out_tsr) # Regularization layer
layer_tsr = keras.layers.Dense(units=1000, activation='tanh', name='fc1')(layer_tsr)
layer_tsr = keras.layers.Dense(units=100, activation='tanh', name='fc2')(layer_tsr)
output_tsr = keras.layers.Dense(units=16, activation='softmax', name='prediction')(layer_tsr)

model = keras.models.Model(inputs=base_in_tsr, outputs=output_tsr, name='TunnedVGG16')

model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
model.summary() # prints model summary
```


Transfer Learning(CNN)

https://keras.io/api/data_loading/

- Cargar Datos entrenamiento: `tf.keras.utils` / `tf.data.Dataset`
 - `tf.keras.utils.image_dataset_from_directory`(`directory`, `labels="inferred"`, `label_mode="int"`, `class_names=None`, `color_mode="rgb"`, `batch_size=32`, `image_size=(256, 256)`, `shuffle=True`, `seed=None`, `validation_split=None`, `subset=None`, `interpolation="bilinear"`, `follow_links=False`, `crop_to_aspect_ratio=False`) → `tf.data.Dataset`

Crea un objeto `tf.data.Dataset` a partir de un directorio de imágenes/descriptores. El objeto contiene los lotes de imágenes (batches). Ajusta las dimensiones si indicamos el tamaño.

`labels` 0...n,

`class_names`: lista con nombres de clase (si no se indica se obtiene del nombre de la carpeta)

`label_mode`: "int", "categorical", "binary"

`validation_split`: [0.0-1.0]

`seed`: semilla para selección aleatoria

(mismo valor para los subsets en `validation_split`)

`subset`: "training" or "validation"

Solo si `validation_split` está activo

```
main_directory/
...class_a/
.....a_image_1.jpg
.....a_image_2.jpg
...class_b/
.....b_image_1.jpg
.....b_image_2.jpg
```

- `tf.keras.utils.load_img`(`path`, `grayscale=False`, `color_mode="rgb"`, `target_size=None`, `interpolation="nearest"`) → `img` (PIL image)
- `tf.keras.utils.img_to_array`(`img`, `data_format=None`, `dtype=None`). → numpy array
- `tf.data.Dataset.from_generator`(generator) → dataset A partir de un generador de datos dataset
- `tf.data.Dataset.from_tensors`(tensors) → dataset A partir de array de tensores
- `tf.data.Dataset.from_tensor_slices`(tensors) → dataset A partir de array de tensores

Transfer Learning(CNN)

https://keras.io/api/data_loading/

- Cargar Datos entrenamiento: `tf.keras.utils` / `tf.data.Dataset`
 - Leeremos el directorio de imágenes dividiendo los datos (**validation_split=0.2**) en dos grupos (**subset**): 'training', 'validation'
 - Opcionalmente podemos usar una semilla fija (**seed=1**) para dividir los datos, de forma que la selección aleatoria sea la misma en múltiples llamadas.
 - Los datos están barajados por defecto (**shuffle=True**)
 - En cada llamada identificamos el **subset**: 'training', 'validation' o bien podemos obtener ambos simultáneamente **both**
 - Los datos se agrupan en lotes **batch=10** (numero de imágenes usadas en cada actualización del gradiente)
 - En `tf.data.Dataset` disponemos también el nombre de las clases obtenido de los directorios (`tf.data.Dataset.class_names`): lista cuyo índice es el id de clase en el tensor de salida.

```
# Load training and test data
trainDataset, testDataset = tf.keras.utils.image_dataset_from_directory( './images/',
    batch_size=10, label_mode='categorical',
    image_size=image_size, color_mode="rgb", crop_to_aspect_ratio=True,
    shuffle=True, validation_split=0.2, seed=1, subset='both')

# extract labels
class_names = trainDataset.class_names
```

Transfer Learning (CNN)

[Documentación](#)

- Cargar Datos entrenamiento: [tf.keras.utils](#) / [tf.data.Dataset](#)
 - Extraer lotes de imágenes y etiquetas: [tf.data.Dataset](#)
 - El objeto **tf.data.Dataset** es iterable y está constituido por los lotes de imágenes y sus etiquetas (tensores)
 - Con un **bucle for** extraeremos los lotes de imágenes. Cada ítem es una tupla con dos tensores: el primero es el tensor de entrada (x) y el segundo es el tensor de salida (y)
 - Opcionalmente con el método **as_numpy_iterator()** se pueden obtener arrays numpy
 - Métodos:
 - [tf.data.Dataset.apply](#)(transformation_func) → dataset aplica transformación predefinida al dataset
 - [tf.data.Dataset.filter](#)(predicate) → dataset aplica filtro booleano (selecciona elementos)
 - [tf.data.Dataset.map](#)(map_func) → dataset aplica transformación a cada elemento del dataset
 - [tf.data.Dataset.flat_map](#)() → dataset aplica transformación/mapa al dataset y →
flatten
 - [tf.data.Dataset.as_numpy_iterator](#)() → dataset Devuelve los datos como arrays numpy
 - [tf.data.Dataset.batch](#)(batch_size) → dataset combina elementos consecutivos en lotes
 - [tf.data.Dataset.cache](#)() → dataset la primera vez que es iterado se guardan en memoria
 - [tf.data.Dataset.concatenate](#)(dataset) Une dos datasets
 - [tf.data.Dataset.shuffle](#)() → dataset baraja el dataset
- Preprocesar entradas:
 - [tf.keras.applications.vgg16.preprocess_input](#)(x, data_format=None) → tensor/numpy
escalado de entradas específico de la red [scale*(X-mean)] **(numpy → np.copy(x))**

Transfer Learning (CNN)

[Documentación](#)

- Cargar Datos entrenamiento: `tf.keras.utils` / `tf.data.Dataset`
 - Preprocesar entradas:
 - `tf.data.Dataset.map`(map_func)
 - `tf.keras.applications.vgg16.preprocess_input`(x, data_format=None) → tensor/numpy
escalado de entradas específico de la red [scale*(X-mean)]
data_format: 'channels_last' (h,w,chan) (def) / 'channels_first' (chan,h,w)
 - `map()` itera cada elemento del dataset y lo transforma mediante la función **map_func**:
`tf.keras.applications.vgg16.preprocess_input()`
 - Vamos a crear una función **lambda** (macro) que recibe en la iteración los datos del dataset **(x,y)** preprocesando los datos **x** (imagen)
 - La función devuelve la tupla **(x_map, y)**

```
# normalize input image color (net specific) [scale*(x-mean)]  
trainDataset = trainDataset.map( lambda x,y: (tf.keras.applications.vgg16.preprocess_input(x), y) )  
  
testDataset = testDataset.map( lambda x,y: (tf.keras.applications.vgg16.preprocess_input(x), y) )
```

Transfer Learning (CNN)

https://keras.io/api/models/model_training_apis/

- **Entrenamiento: método `fit()` `keras.models.Model`**
 - `keras.models.Model.fit(x, y=None, batch_size=None, verbose="auto", shuffle=True, epochs=1, steps_per_epoch=None, validation_split=0.0, validation_data=None) → history`
 - x:** DataSet ó tensor de entrada, matriz de datos → fila: muestras, columnas: entradas
 - y:** tensor de salidas (etiquetas), matriz de datos → fila: muestras, una columna por salida
None si usamos un DataSet
 - validation_data:** dataset de validación o tupla (x_val,y_val)
 - validation_split:** [0.0-1.0] fija el% de datos usados para validación (optimización red).
No Disponible para DataSets, solo para arrays numpy o tensores
 - steps_per_epoch:** número de lotes de datos para completar una iteración (epoch) (Opcional)

```
# train the network
train_res = model.fit( trainDataset, epochs=5, verbose=1, validation_data=testDataset)
```

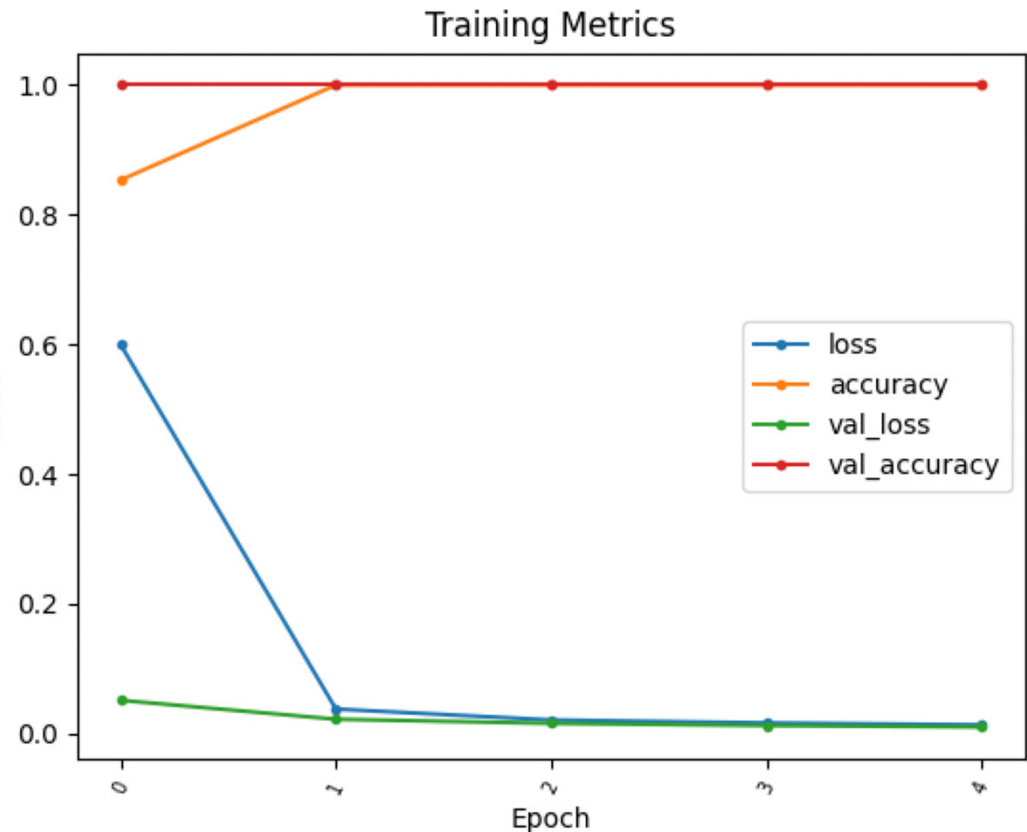
```
Epoch 1/5
338/338 [=====] - 42s 121ms/step - loss: 1.1650 - accuracy: 0.6479 - val_loss: 0.3114 - val_accuracy: 0.9189
Epoch 2/5
338/338 [=====] - 40s 118ms/step - loss: 0.1686 - accuracy: 0.9260 - val_loss: 0.1338 - val_accuracy: 0.9189
Epoch 3/5
338/338 [=====] - 40s 118ms/step - loss: 0.0804 - accuracy: 0.9675 - val_loss: 0.0367 - val_accuracy: 1.0000
Epoch 4/5
338/338 [=====] - 42s 123ms/step - loss: 0.0382 - accuracy: 0.9911 - val_loss: 0.0790 - val_accuracy: 0.9730
Epoch 5/5
338/338 [=====] - 41s 120ms/step - loss: 0.0163 - accuracy: 0.9970 - val_loss: 0.0087 - val_accuracy: 1.0000
```

Transfer Learning (CNN)

https://keras.io/api/models/model_training_apis/

- Entrenamiento: método **fit()**

```
from titere import plotLearningCurves  
plotLearningCurves(train_res.history)
```



- Evaluación: método **evaluate()** `keras.models.Model`
 - `keras.models.Model.evaluate(x, y=None)` → `loss_and_metrics`

```
# Evaluate classifier  
loss_and_metrics = model.evaluate(testDataset)  
print(f"Loss and accuracy: ", loss_and_metrics)
```

```
Loss and accuracy: [0.008703882806003094, 1.0]
```

Transfer Learning (CNN)

https://keras.io/api/models/model_training_apis/

- Predicción: método `predict()` `keras.models.Model`
 - `keras.models.Model.predict(x) → y`
 - `x`: DataSet ó tensor de entrada, matriz de datos → fila: muestras, columnas: entradas
 - `y`: tensor de salidas (etiquetas), matriz de datos → fila: muestras, una columna por salida

```
# Prediction
classes = model.predict(testData)

# convert predictionMat to vector: search for max response across classes (columns)
y_pred = classes.argmax(axis=1)
prob = classes.max(axis=1)

# extract labels as numpy array, y tensor has shape (m,b,n) -> (m*b)
# batch in testDataset (batch[0]->x, batch[1]->y)
y_test = np.array( [item for batch in testDataset.as_numpy_iterator() for item in batch[1]] ).argmax(axis=1)

print(f"(Predicted,Actual):\n", list(zip(y_pred, y_test)))
print(f"Prob:\n ", prob)

# save trained model
model.save('MyNet_vgg16.h5')      # File formats: .h5 (HDF5) / .keras (New Keras native)

# wait for a key to keep plot on screen
input('click a key')
```

Exportar modelos Keras

<https://github.com/onnx/tensorflow-onnx>

- Exportar modelos Keras a formatos estándar: ONNX
 - Keras almacena los modelos en formato propio en ficheros HDF5/Keras
 - Debemos instalar el paquete: `pip install tf2onnx`
`import tf2onnx`
 - Conversión:
 - `tf2onnx.convert.from_keras(model, input_signature=None, opset=None, custom_ops=None, custom_op_handlers=None, custom_rewriter=None, inputs_as_nchw=None, extra_opset=None, shape_override=None, target=None, large_model=False, output_path=None)` → `onnx_model`, `external_tensor_storage`
 - Almacenar modelo:
 - Desde la función de conversión como parámetro: `output_path`
 - Almacenando el modelo devuelto: `onnx_model`
 - `onnx_model.SerializeToString()` Genera una cadena binaria para almacenar en un fichero

```
import tf2onnx

# convert model to standard format ONNX
tf2onnx.convert.from_keras(model, output_path="MyNet_vgg16.onnx")

# save class names file
with open("aloi-16-labels.txt", "w") as f:
    for label in class_names:
        f.write(CLASSES_TEXT[label] + '\n')
```

Transfer Learning(CNN)

https://keras.io/api/data_loading/

- Manejo alternativo de DataSets mediante arrays numpy:

```
# Load training data
dataset = tf.keras.utils.image_dataset_from_directory( './images/', batch_size=None, label_mode='categorical',
                                                    image_size=image_size, color_mode="rgb", crop_to_aspect_ratio=True)

# extract image data / labels
class_names = dataset.class_names
X=[]; y=[]
for batch in dataset.as_numpy_iterator():
    X.append(batch[0]); y.append(batch[1])

# normalize input image color (net specific) [scale*(x-mean)]
X = [ tf.keras.applications.vgg16.preprocess_input(np.copy(image)) for image in X ]

# convert array lists to numpy arrays
X = np.array(X); y = np.array(y)

# split into train test sets (sklearn.model_selection.train_test_split)
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=1, stratify=y)

# train the network
train_res = model.fit(X_train, y_train, epochs=5, batch_size=10, verbose=1, validation_split=0.1)
plotLearningCurves(train_res.history)

# Evaluate classifier
loss_and_metrics = model.evaluate(X_test, y_test)
print(f"Loss and accuracy: ", loss_and_metrics)
```

Transfer Learning(CNN)

[Documentación](#)

- Manejo alternativo de DataSets mediante arrays numpy:
 - Dividir dataset en dos conjuntos: Entrenamiento/Test

```
from sklearn.model_selection import train_test_split
```

- `sklearn.model_selection.train_test_split(X, y, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)`
→ `(X_train, X_test, y_train, y_test)`
 - Divide los datos de entrada en dos conjuntos: train , test
 - **test_size**: [0.0-1.0] porcentaje de datos conjunto test (1-train_size if None)
 - **train_size**: [0.0-1.0] porcentaje de datos conjunto train (1-test_size if None)
 - **random_state**: semilla para selección aleatoria. Si ponemos un valor fijo, en las siguientes llamadas se hace la misma selección aleatoria → permite comparar algoritmos con los mismos datos
 - **shuffle**: (bool) barajar los datos antes de dividirlos
 - **stratify**: (array de etiquetas) mantiene el mismo porcentaje de datos de cada clase en los dos

Transfer Learning (CNN)

- **Aceleración:**

- El entrenamiento es lento ya que en cada iteración es preciso realizar la inferencia de todas las imágenes de entrenamiento en las capas convolucionales de la red VGG16.
- Solución:
 - Construir un modelo de red separado (**subred1**) para las primeras capas de la red convolucional VGG16 (**Input** → **Flatten**) (*capas ya entrenadas*)
 - Construir un modelo de red separado (**subred2**) para las últimas capas de clasificación (*capas a entrenar*)
 - Precalcular los descriptores a la salida de la capa '**Flatten**' de la red VGG16 (**subred1**) para las imágenes de entrenamiento
 - Usar estos descriptores de las imágenes de entrenamiento para entrenar la **subred2** de clasificación.
 - Testear la red completa con las imágenes de Test
- Este método no permite aprovechar las opciones de aumento de datos mediante rotaciones y traslaciones aleatorias de Keras-Tensorflow (clase **ImageDataGenerator**) (**model.fit_generator()**)

Transfer Learning (CNN)

[Documentación](#)

- Preprocesamiento (*'Data Augmentation'*): `tf.keras.preprocessing.image`
`keras.models.Model.fit_generator()`

Classes

[class DirectoryIterator](#): Iterator capable of reading images from a directory on disk.

[class ImageDataGenerator](#): Generate batches of tensor image data with real-time data augmentation.

[class Iterator](#): Base class for image data iterators.

[class NumpyArrayIterator](#): Iterator yielding data from a Numpy array.

Functions

[apply_affine_transform\(...\)](#): Applies an affine transformation specified by the parameters given.

[apply_brightness_shift\(...\)](#): Performs a brightness shift.

[apply_channel_shift\(...\)](#): Performs a channel shift.

[array_to_img\(...\)](#): Converts a 3D Numpy array to a PIL Image instance.

[img_to_array\(...\)](#): Converts a PIL Image instance to a Numpy array.

[load_img\(...\)](#): Loads an image into PIL format.

[random_brightness\(...\)](#): Performs a random brightness shift.

[random_channel_shift\(...\)](#): Performs a random channel shift.

[random_rotation\(...\)](#): Performs a random rotation of a Numpy image tensor.

[random_shear\(...\)](#): Performs a random spatial shear of a Numpy image tensor.

[random_shift\(...\)](#): Performs a random spatial shift of a Numpy image tensor.

[random_zoom\(...\)](#): Performs a random spatial zoom of a Numpy image tensor.

[save_img\(...\)](#): Saves an image stored as a Numpy array to a path or file object.

[smart_resize\(...\)](#): Resize images to a target size without aspect ratio distortion.

Data Augmentation (CNN)

[Documentación](#)

- Clase `ImageDataGenerator`
- `tf.keras.preprocessing.image.ImageDataGenerator()` → Objeto `ImageDataGenerator`

Args

<code>featurewise_center</code>	Boolean. Set input mean to 0 over the dataset, feature-wise.
<code>samplewise_center</code>	Boolean. Set each sample mean to 0.
<code>featurewise_std_normalization</code>	Boolean. Divide inputs by std of the dataset, feature-wise.
<code>samplewise_std_normalization</code>	Boolean. Divide each input by its std.
<code>zca_epsilon</code>	epsilon for ZCA whitening. Default is 1e-6.
<code>zca_whitening</code>	Boolean. Apply ZCA whitening.
<code>rotation_range</code>	Int. Degree range for random rotations.
<code>width_shift_range</code>	Float, 1-D array-like or int •float: fraction of total width, if < 1, or pixels if >= 1. •1-D array-like: random elements from the array. •int: integer number of pixels from interval (-width_shift_range, +width_shift_range)
<code>height_shift_range</code>	Float, 1-D array-like or intfloat: fraction of total height,
<code>brightness_range</code>	Tuple or list of two floats. Range for picking a brightness shift value from.
<code>shear_range</code>	Float. Shear Intensity (Shear angle in counter-clockwise direction in degrees)
<code>zoom_range</code>	Float or [lower, upper]. Range for random zoom. If a float, [lower, upper] = [1-zoom_range, 1+zoom_range].
<code>channel_shift_range</code>	Float. Range for random channel shifts.
<code>fill_mode</code>	One of {"constant", "nearest", "reflect" or "wrap"}.
<code>cval</code>	Float or Int. Value used for points outside the boundaries when fill_mode = "constant".

.....

Data Augmentation (CNN)

[Documentación](#)

- Clase `ImageDataGenerator`
- `tf.keras.preprocessing.image.ImageDataGenerator()`

Args

horizontal_flip	Boolean. Randomly flip inputs horizontally.
vertical_flip	Boolean. Randomly flip inputs vertically.
rescale	rescaling factor. Defaults to None. If None or 0, no rescaling is applied, otherwise we multiply the data by the value provided (after applying all other transformations).
preprocessing_function	function that will be applied on each input. The function will run after the image is resized and augmented. The function should take one argument: one image (Numpy tensor with rank 3), and should output a Numpy tensor with the same shape.
data_format	Image data format, either "channels_first" or "channels_last"
validation_split	Float. Fraction of images reserved for validation (strictly between 0 and 1).
dtype	Dtype to use for the generated arrays.

- Métodos:

- `ImageDataGenerator.fit(x, augment=False, rounds=1, seed=None)`
compute quantities required for featurewise/sample normalization

x	Sample data. Should have rank 4. In case of grayscale data, the channels axis should have value 1, in case of RGB data, it should have value 3, and in case of RGBA data, it should have value 4.
augment	Boolean (default: False). Whether to fit on randomly augmented samples.
rounds	Int (default: 1). If using data augmentation (augment=True), this is how many augmentation passes over the data to use.
seed	Int (default: None). Random seed.

Data Augmentation (CNN)

[Documentación](#)

- Clase `ImageDataGenerator`

- Métodos:

- `ImageDataGenerator.flow(x, y)` → augmented data
takes data & label arrays, generates batches of augmented data

x	Input data. Numpy array of rank 4 or a tuple. If tuple, the first element should contain the images and the second element another numpy array or a list of numpy arrays that gets passed to the output without any modifications.
y	Labels.
batch_size	Int (default: 32).
shuffle	Boolean (default: True). Shuffle data before validation_split
sample_weight	Sample weights.
seed	Int (default: None). Optional random seed for shuffling and transformations / validation_split
save_to_dir	None or str (default: None). This allows you to optionally specify a directory to which to save the augmented pictures being generated (useful for visualizing what you are doing).
save_prefix	Str (default: ""). Prefix to use for filenames of saved pictures (only relevant if save_to_dir is set).
save_format	one of "png", "jpeg", "bmp", "pdf", "ppm", "gif", "tif", "jpg" (only relevant if save_to_dir is set). Default: "png".
subset	Subset of data ("training" or "validation") if validation_split is set in ImageDataGenerator.

Data Augmentation (CNN)

[Documentación](#)

- Clase **ImageDataGenerator**

- Métodos:

- **ImageDataGenerator.flow_from_directory**(directory) → augmented data
takes path to a directory & generates batches of augmented data

directory,

target_size=(256, 256), (height, width)

color_mode='rgb',

classes=None, Optional list of class subdirectories

class_mode='categorical', "categorical", "binary", "sparse" (int), "input" (image-Autoencoders)

batch_size=32,

shuffle=True, batch samples are selected randomly for each epoch

save_to_dir=None,

save_prefix="",

save_format='png',

follow_links=False,

seed=None, Optional random seed for shuffling and transformations

subset=None, "training" or "validation" if validation_split is set in ImageDataGenerator.

interpolation='nearest'

- **ImageDataGenerator.flow_from_dataframe**(dataframe) → augmented data
takes a Pandas dataframe & generates batches of augmented data

Data Augmentation (CNN)

- Clase `ImageDataGenerator`

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Configure DataGenerator for Data augmentation (sets split for train/validation)
# preprocessing_function: normalize input image color (net specific) [scale*(x-mean)]
train_generator = ImageDataGenerator( rescale = 1., validation_split=0.2, fill_mode='nearest',
                                     rotation_range = 40, width_shift_range = 0.2, height_shift_range = 0.2,
                                     shear_range = 0.2, zoom_range = 0.2, horizontal_flip=True, vertical_flip=True,
                                     preprocessing_function = tf.keras.applications.vgg16.preprocess_input )

# For validation data do only normalization
val_generator = ImageDataGenerator( rescale = 1., validation_split=0.2,
                                   preprocessing_function = tf.keras.applications.vgg16.preprocess_input )

# Reads train and validation Datasets from ImageDataGenerator
trainDataset = train_generator.flow_from_directory( './images/', target_size = image_size,
                                                  batch_size = 10, class_mode = 'categorical', color_mode='rgb'
                                                  keep_aspect_ratio=True, shuffle=True, seed= 1, subset='training' )

testDataset = val_generator.flow_from_directory( './images/', target_size = image_size,
                                                batch_size = 10, class_mode = 'categorical', color_mode='rgb'
                                                keep_aspect_ratio=True, shuffle=False, seed= 1, subset='validation' )

print(f"{trainDataset.samples=} / {testDataset.samples=}")

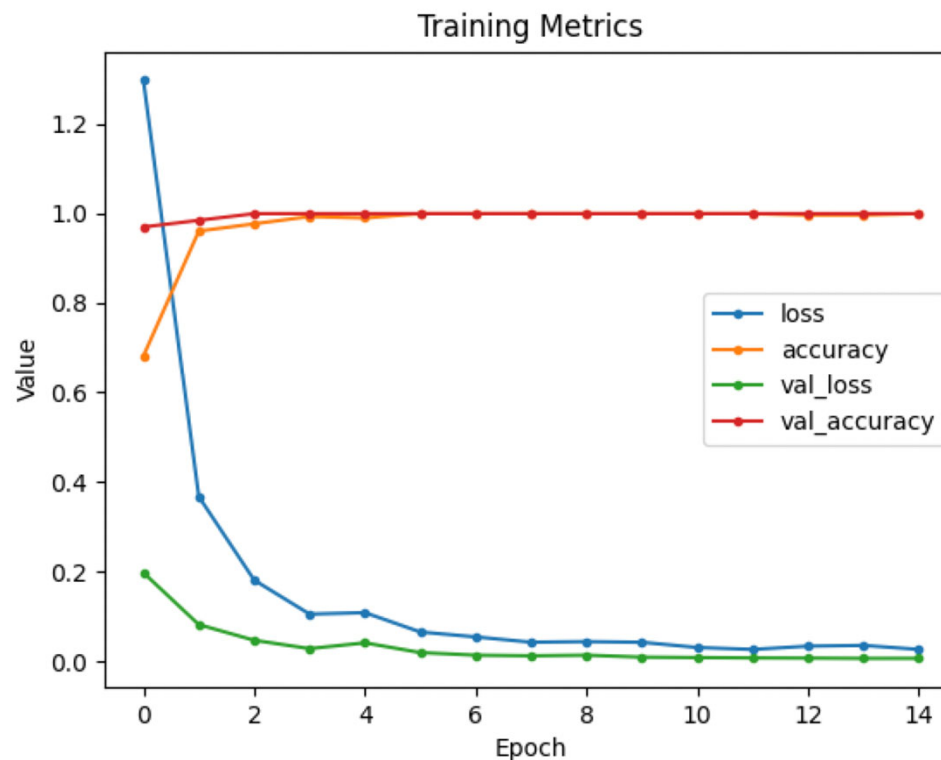
# extract labels idx list: ([idx] -> labels) trainDataset.class_indices is a dictionary label-> idx
class_names = [label for label, index in sorted(trainDataset.class_indices.items(), key=lambda item: item[1])]
print(f"ClassNames ({len(class_names)}):", class_names)
```

Data Augmentation (CNN)

- Clase `ImageDataGenerator`. Entrenamiento:
 - En versiones antiguas de Keras es preciso usar `fit_generator()` en lugar de `fit()`
 - `keras.models.Model.fit_generator(dataGenerator, epochs=1, validation_data=None verbose="auto", shuffle=True) → history`
 - `keras.models.Model.fit(dataGenerator,) → history`

```
# train the network (only new layers) using fit_generator instead of fit (only for older versions of keras)  
train_res = model.fit ( trainDataset, epochs=15, verbose=1, validation_data=testDataset )
```

```
plotLearningCurves(train_res.history)
```



Data Augmentation (CNN)

- Clase `ImageDataGenerator`

- Predicción: método `predict()` `keras.models.Model`
 - `keras.models.Model.predict(x) → y`

```
# Prediction
classes = model.predict(testData)

# convert predictionMat to vector: search for max response across classes (columns)
y_pred = classes.argmax(axis=1)
prob = classes.max(axis=1)
```

- Extraer etiquetas de un `DataGenerator` como array numpy de enteros
- El iterador devuelto por `ImageDataGenerator` nunca termina por lo que tenemos que limitar el número de iteraciones a los lotes disponibles

```
# extract labels as numpy array from ImageDataGenerator (Iterator never ends)
# shuffle == False in flow_from_directory to keep labels order
y_test = []
num_batches = int(np.ceil(testDataset.samples / testDataset.batch_size))
for i in range(num_batches):
    data, labels = next(testDataset)
    y_test.extend(labels)

y_test = np.array(y_test).argmax(axis=1) # from categorical to int
print(f"(Predicted,Actual):\n", list(zip(y_pred, y_test)))
print(f"Prob:\n ", prob)
```

Leer Datasets predefinidos

<https://keras.io/api/datasets/>

- Módulo: `keras.datasets`

- `keras.datasets.mnist.load_data()` → (x_train, y_train), (x_test, y_test)
dataset 60,000 28x28 grayscale images of the 10 digits, test set of 10,000 images
- `keras.datasets.cifar10.load_data()` → (x_train, y_train), (x_test, y_test)
dataset 50,000 32x32 color training images and 10,000 test images, labeled over 10 categories
[automobile,bird,cat,deer,dog,frog,horse,ship,truck]
- `keras.datasets.cifar100.load_data(label_mode="fine")` → (x_train, y_train), (x_test, y_test)
dataset 50,000 32x32 color training images and 10,000 test images, labeled over 100 categories
- `keras.datasets.imdb.load_data()` → (x_train, y_train), (x_test, y_test)
dataset of 25,000 movies reviews from IMDB labeled by sentiment (positive/negative).
- `keras.datasets.reuters.load_data()` → (x_train, y_train), (x_test, y_test)
dataset of of 11,228 newswires from Reuters, labeled over 46 topics.
- `keras.datasets.fashion_mnist.load_data()` → (x_train, y_train), (x_test, y_test)
dataset of 60,000 28x28 grayscale images of 10 fashion categories
[T-shirt/top,Trouser,Pullover,Dress,Coat,Sandal,Shirt,Sneaker,Bag,Ankle boot]
- `keras.datasets.boston_housing.load_data()` → (x_train, y_train), (x_test, y_test)
dataset 13 attributes of houses at different locations around the Boston suburbs in the late 1970s. Targets are the median values of the houses at a location (in k\$).