

Departamento de Física Médica - Centro atómico Bariloche - IB

Deep Learning Software: TensorFlow y Keras

Ariel Hernán Curiale

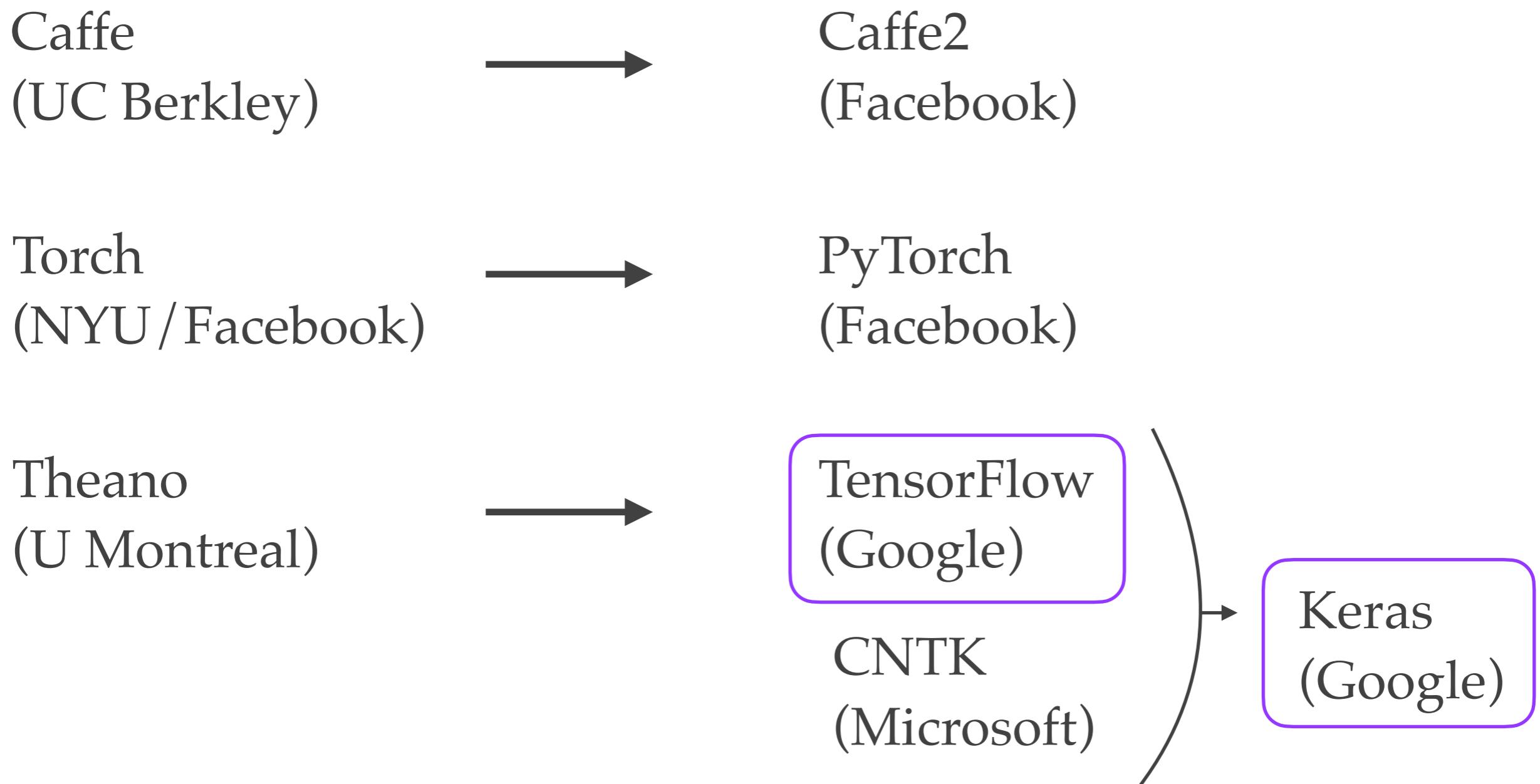
ariel.curiale@cab.cnea.gov.ar



UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO



Deep learning software



TensorFlow

TensorFlow

https://www.tensorflow.org/api_docs/python/

The screenshot shows the TensorFlow API documentation for Python. The page title is "All symbols in TensorFlow". Under the "Primary symbols" section, several modules are listed, including `tf`, `tf.AggregationMethod`, `tf.Assert`, `tf.AttrValue`, `tf.AttrValue.ListValue`, `tf.ConditionalAccumulator`, `tf.ConditionalAccumulatorBase`, `tf.ConfigProto`, `tf.ConfigProto.DeviceCountEntry`, `tf.ConfigProto.Experimental`, and `tf.CriticalSection`. On the left sidebar, there's a navigation menu with items like Overview, Python, JavaScript, C++, Java, and All Symbols.

<https://www.tensorflow.org/tutorials/>

The screenshot shows the TensorFlow tutorials page. The main heading is "Get Started with TensorFlow". Below it, a section titled "Learn and use ML" provides an overview of the Keras API for building deep learning models. It includes a code snippet for loading the MNIST dataset and compiling a sequential model. The code is as follows:

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
```

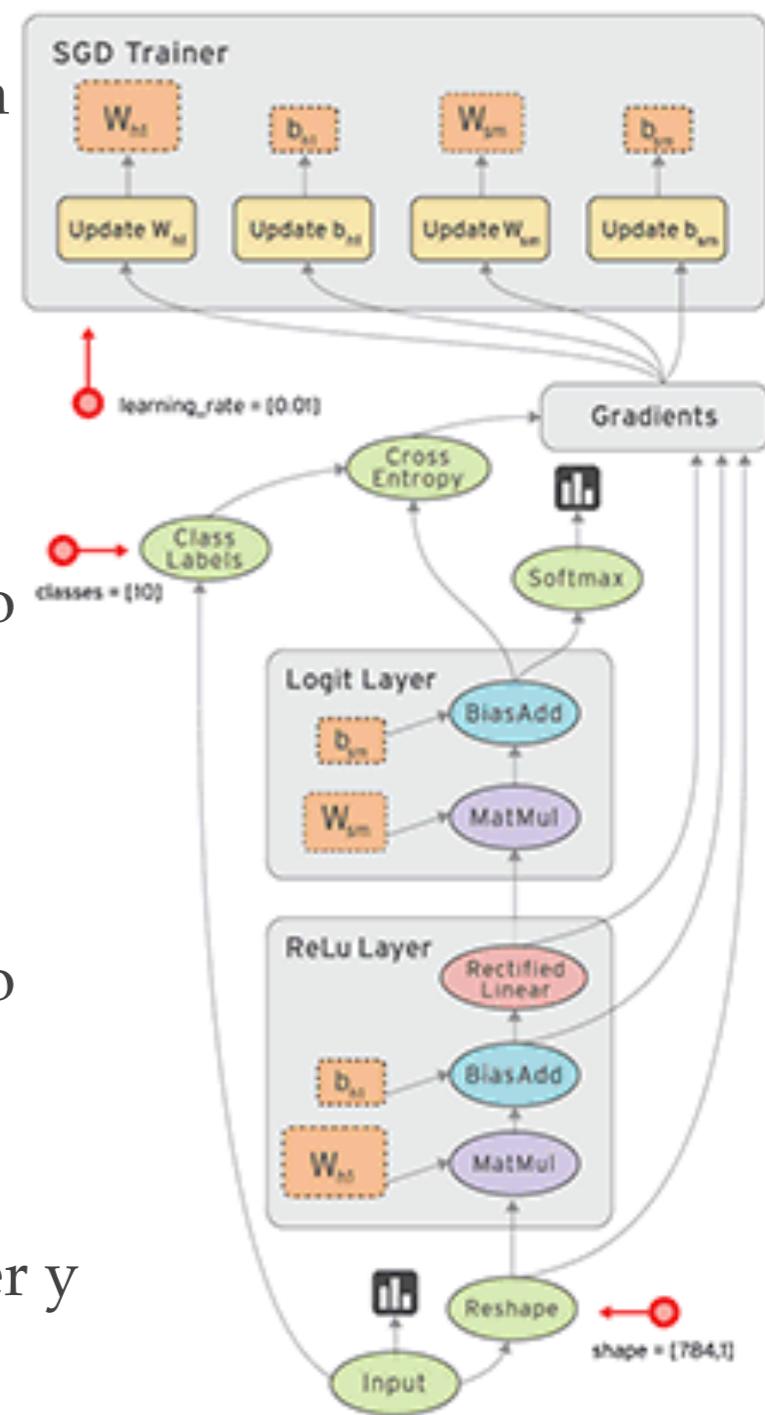
TensorFlow

- ❖ Los datos son Tensores (numpy arrays para los valores de los tensores)

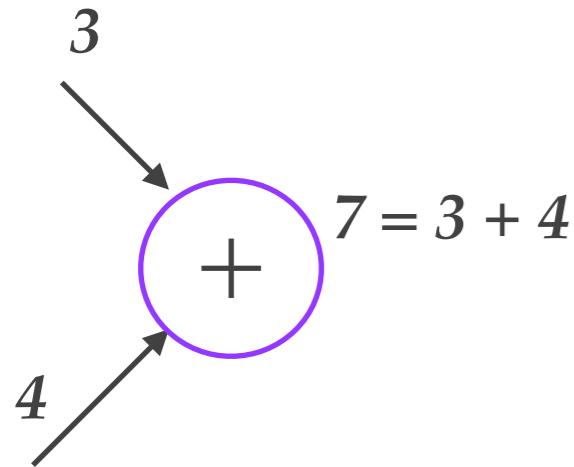
```
3. # a rank 0 tensor; a scalar with shape [],  
[1., 2., 3.] # a rank 1 tensor; a vector with shape [3]  
[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]  
[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
```
- ❖ TensorFlow utiliza el flujo de datos en los grafos para representar los cálculos computacionales y las dependencias entre operaciones y variables.
- ❖ Todo programa en TF esta formado por dos grandes secciones:
 1. Creación del grafo computacional
 2. Ejecución del grafo

TensorFlow: Data Flow Graphs

- ❖ Data Flow Graphs es un modelo de programación muy común en computación paralela y tiene varias ventajas:
 - ❖ **Paralelismo.** Las aristas del grafo permiten identificar las dependencias entre operaciones (nodos), haciendo más simple aquellas que se pueden ejecutar en paralelo
 - ❖ **Ejecución distribuida.** Al utilizar las aristas del grafo como tensores/datos que fluyen entre las operaciones es más simple dividir las operaciones entre múltiples dispositivos (CPUs, GPUs y TPUs)
 - ❖ **Compilación.** Es más simple optimizar el código en tiempo de compilación
 - ❖ **Portabilidad.** El modelo de grafos es independiente del lenguaje. Modelos almacenados desde python se puede leer y ejecutar en C++



TensorFlow: Ejemplo



Ejemplo: $3 + 4 = 7$

```
import numpy as np
import tensorflow as tf
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
```

Creamos el grafo

Ejecutamos el grafo

```
sess = tf.Session()
result = sess.run(total)
print(result)
```

7.0

```
print(a)
print(b)
print(total)
```

```
Tensor("Const:0", shape=(), dtype=float32)
Tensor("Const_1:0", shape=(), dtype=float32)
Tensor("add:0", shape=(), dtype=float32)
```

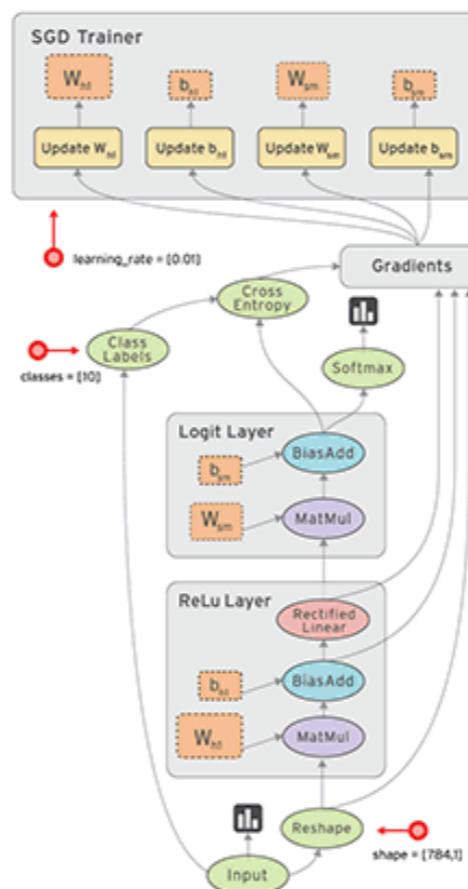
?

TensorFlow: Sesión

La sesión es necesaria para evaluar cualquier tensor que esta definido en el grado computacional

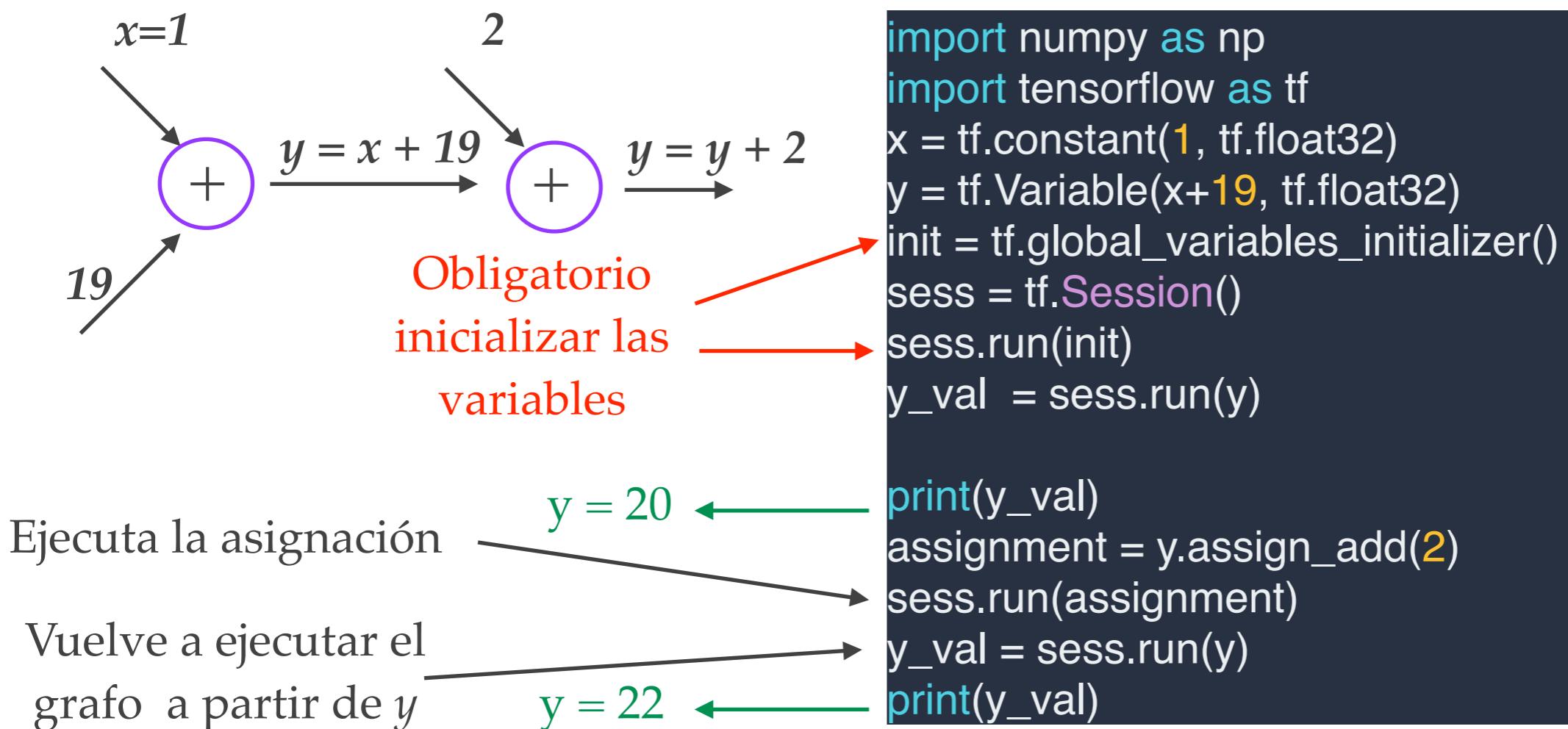
- ❖ Al ejecutar el método run Tensorflow hace un seguimiento hacia atrás del grafo para ejecutar todos los nodos

Durante la ejecución los tensores solo tienen un valor



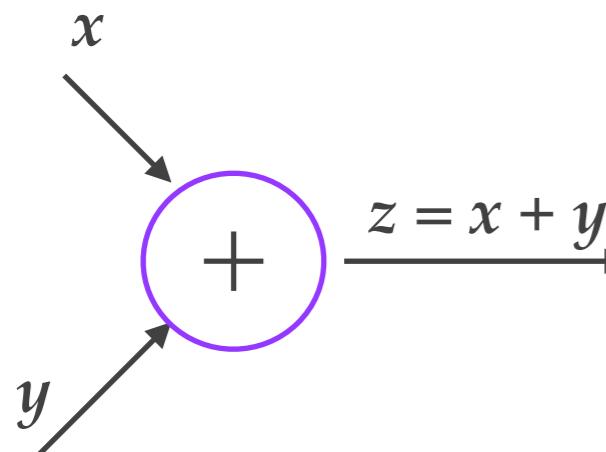
TensorFlow: Constantes y Variables

- ❖ **Constantes:** tensores que no cambian de valor
- ❖ **Variables:** son tensores que pueden cambiar de valor que persisten a lo largo de múltiples llamadas *Session.run()*



TensorFlow: Placeholder

- ❖ **Placeholder:** son similares a las variables pero no es necesario proporcionar un valor inicial ya que se puede proveer en tiempo de ejecución con el argumento *feed_dict* del método *Session.run()*



Vuelve a ejecutar el
grafo partir de z

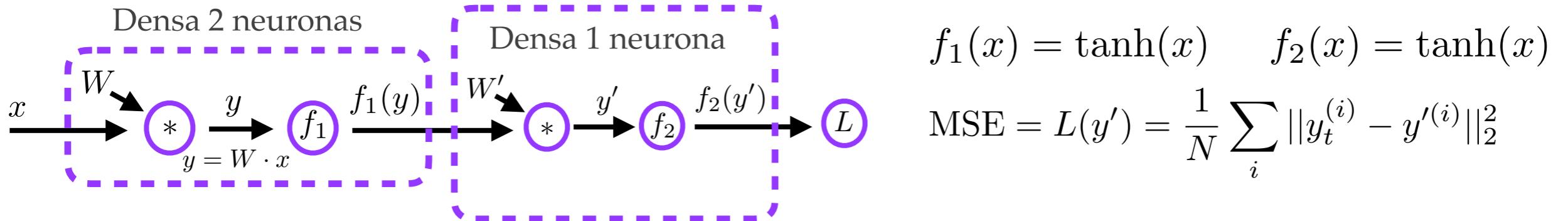
7.5
[3., 7.]

```
import numpy as np
import tensorflow as tf
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = x + y
sess = tf.Session()
z_value1 = sess.run(z, feed_dict={x: 3, y: 4.5})
z_value2 = sess.run(z, feed_dict={x: [1, 3], y: [2, 4]})

print(z_value1)
print(z_value2)
```

Feed: Diccionario de python

Problema 2.6 con TF



```
1 # Datos
2 x_data = np.array([[-1,-1], [-1,1], [1,-1], [1,1]])
3 y_data = np.array([[1],[-1],[-1],[1]])
4 # Variables
5 x = tf.placeholder(tf.float32, shape=[None, 2])
6 y = tf.placeholder(tf.float32, shape=[None, 1])
7 W1 = tf.Variable(tf.random_normal((2,2))) # (neuronas, input)
8 b1 = tf.Variable(tf.random_normal((2,)))
9 W2 = tf.Variable(tf.random_normal((2,1))) # (neurona, input f1)
10 b2 = tf.Variable(tf.random_normal((1,)))
11 # Armamos el grafo
12 y1 = tf.matmul(x,W1) + b1
13 f1 = tf.tanh(y1)
14 y2 = tf.matmul(f1, W2) + b2
15 f2 = tf.tanh(y2)
16 # Ejecutamos
17 with tf.Session() as sess:
18     init = tf.global_variables_initializer()
19     sess.run(init)
20     out = sess.run(f2, feed_dict={x:x_data, y:y_data})
```

Definimos las variables
y los placeholder
Variables: indicamos
su valor inicial par

Armamos el grafo
No hay computo

Lo ejecutamos
Hay computo

Problema 2.6 en TF

```
12 y1 = tf.matmul(x,w1) + b1
13 f1 = tf.tanh(y1)
14 y2 = tf.matmul(f1, w2) + b2
15 f2 = tf.tanh(y2)
16 diff = y - f2
17 loss = tf.reduce_mean(tf.reduce_sum(diff**2, axis=1))
18 grad_w1, grad_b1, grad_w2, grad_b2 = tf.gradients(loss, [W1, b1, W2, b2])
19 lr = 1
20 # Actualizamos los parametros de la red
21 w1_update = W1.assign_sub(lr*grad_w1)
22 b1_update = b1.assign_sub(lr*grad_b1)
23 w2_update = W2.assign_sub(lr*grad_w2)
24 b2_update = b2.assign_sub(lr*grad_b2)
25 updates = tf.group(w1_update, b1_update, w2_update, b2_update)
26
27 # Ejecutamos
28 with tf.Session() as sess:
29     init = tf.global_variables_initializer()
30     sess.run(init)
31     losses = []
32     for epoch in range(100):
33         loss_val, _ = sess.run([loss, updates], feed_dict={x:x_data, y:y_data})
34         losses.append(loss_val)
35     y_pred = sess.run(f2, feed_dict={x:x_data, y:y_data})
36     acc = np.isclose(y_data, y_pred, atol=1e-1).mean()
37     print('epoch: {:03d}\t loss: {:.3f}\t acc:{:.3f}'.format(epoch,
38                 loss_val, acc))
```

Grafo: costo,
gradientes y
updates

No hay
computo

Actualizamos

En cada época
computamos
el loss y

actualizamos

Dispara
computo
gradientes

TensorBoard

Definimos una función para registrar varias cosas

```
1 # Para usar tensorboard
2 def variable_summaries(var):
3     """Attach a lot of summaries to a Tensor (for TensorBoard visualization)."""
4     with tf.name_scope('summaries'):
5         mean = tf.reduce_mean(var)
6         tf.summary.scalar('mean', mean)
7         with tf.name_scope('stddev'):
8             stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
9             tf.summary.scalar('stddev', stddev)
10            tf.summary.scalar('max', tf.reduce_max(var))
11            tf.summary.scalar('min', tf.reduce_min(var))
12            tf.summary.histogram('histogram', var)
```

```
33 variable_summaries(W1)
34 variable_summaries(b1)
35 variable_summaries(W2)
36 variable_summaries(b2)
```

```
44 diff = y - f2
45 loss = tf.reduce_mean(tf.reduce_sum(diff**2, axis=1))
46 is_close = tf.less_equal(tf.abs(y-f2), 0.1)
47 acc = tf.reduce_mean(tf.cast(is_close, tf.float32))
48 tf.summary.scalar('loss', loss)
49 tf.summary.scalar('accuracy', acc)
```

TensorBoard

```
26 merged = tf.summary.merge_all() ←
29 # Ejecutamos
30 with tf.Session() as sess:
31     train_writer = tf.summary.FileWriter('tensorboard_train', sess.graph) ←
32     init = tf.global_variables_initializer()
33     sess.run(init)
34     losses = []
35     for epoch in range(100):
36         summary, acc_val, loss_val, _ = sess.run([merged, acc, loss, updates],
37             feed_dict={x:x_data, y:y_data}) ←
38         losses.append(loss_val)
39         train_writer.add_summary(summary, epoch) ←
41         print('epoch: {:03d}\t loss: {:.3f}\t acc:{:.3f}'.format(epoch, loss_val, acc_val))
```

```
tensorboard --logdir=path/to/log-directory
```

TensorBoard

localhost:8008/#graphs&run=.

TensorBoard SCALARS GRAPHS DISTRIBUTIONS HISTOGRAMS INACTIVE

Search nodes. Regexes supported.

Fit to Screen Download PNG

Run () Tag () Default

Upload Choose File

Graph Conceptual Graph Profile Trace inputs

Color Structure Device XLA Cluster Compute time Memory

Close legend.

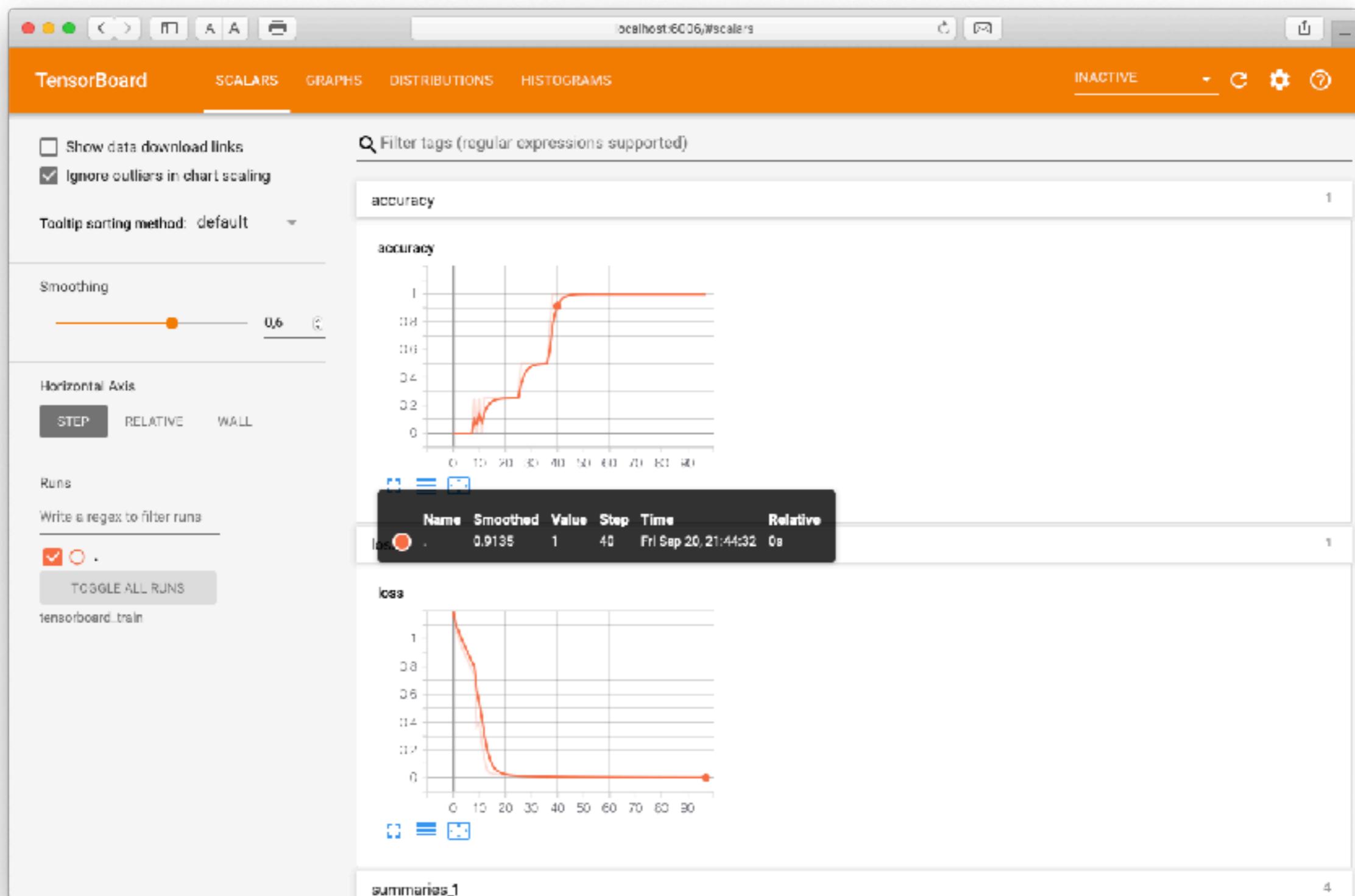
Graph (* = expandable)

- NameScope 2
- CpNode 2
- Unconnected series* 2
- Connected series* 2
- Constant 2
- Summary 2
- Dataflow edge 2
- Control dependency edge 2
- Reference edge 2

con `tf.name_scope`
agrupamos operaciones
en el grafo

The screenshot displays the TensorBoard interface for monitoring TensorFlow runs. The main area shows a complex computational graph with various operations and their dependencies. A specific section of the graph is highlighted with a red box, illustrating the use of `tf.name_scope` to group operations. This grouping is clearly visible in the highlighted portion of the graph, where operations like `Mean`, `Sum`, `Pow`, `Abs`, `LessEqual`, `Add`, `MatMul`, `Tanh`, and `PlaceHolder` are clustered under a single scope. The right side of the interface provides a detailed view of the 'gradients' section, showing multiple `Assign` operations for variables like `weights` and `bias`. The overall interface is orange-themed, with tabs for SCALARS, GRAPHS, DISTRIBUTIONS, and HISTOGRAMS, and a sidebar for managing runs and uploads.

TensorBoard



Métricas, loss y optimizadores

```
7 # Armamos el grafo
8 x = tf.placeholder(tf.float32, shape=[None, 2], name='InputData')
9 y = tf.placeholder(tf.float32, shape=[None, 1], name='OutputData')
10 with tf.name_scope('Layer1'):
11     W1 = tf.Variable(tf.random_normal((2,2)), name='W1') # (neuronas, input)
12     b1 = tf.Variable(tf.zeros((2,)), name='b1')
13     l1 = tf.nn.tanh(tf.matmul(x,W1) + b1)
14 with tf.name_scope('Layer2'):
15     W2 = tf.Variable(tf.random_normal((2,1)), name='W2') # (neurona, input f1)
16     b2 = tf.Variable(tf.zeros((1,)), name='b2')
17     l2 = tf.nn.tanh(tf.matmul(l1,W2) + b2)
18 loss = tf.losses.mean_squared_error(labels=y, predictions=l2)
19 acc = tf.reduce_mean(tf.cast(tf.less_equal(tf.abs(y-l2),0.1), tf.float32))
20 acc2 = tf.metrics.mean_absolute_error(labels=y, predictions=l2)
21 tf.summary.scalar('Loss', loss)
22 tf.summary.scalar('Accuracy', acc)
23 merged = tf.summary.merge_all()
24 optimizer = tf.train.GradientDescentOptimizer(learning_rate=1)
25 train = optimizer.minimize(loss)
26 # Ejecutamos
27 with tf.Session() as sess:
28     train_writer = tf.summary.FileWriter('tensorboard_train2', sess.graph)
29     sess.run(tf.global_variables_initializer())
30     sess.run(tf.local_variables_initializer())
31     losses = []
32     for epoch in range(100):
33         summary, acc_val, acc2_val, loss_val, _ = sess.run([merged, acc, acc2, loss, train],
34             feed_dict={x:x_data, y:y_data})
35         losses.append(loss_val)
36         train_writer.add_summary(summary, epoch)
37         print('epoch: {:03d}\t loss: {:.3f}\t acc:{:.3f}\t acc2:{:.3f}'.format(epoch,
38             loss_val, acc_val, acc2_val[0]))
```

?

loss

- ❖ Ojo cuando se utiliza la función de TF de *cross-entropy* ya que la implementación aplica la activación *softmax* por cuestiones de optimización y **no hay que llamarla con *softmax*** al crear la función de costo

TensorFlow

Install Learn API More

Search Language GitHub Sign in

normalize_moments
pool
quantized_avg_pool
quantized_conv2d
quantized_max_pool
quantized_relu_x
raw_rnn
relu
relu6
relu_layer
safe_embedding_lookup_sparse
sampled_softmax_loss
selu
separable_conv2d
sigmoid_cross_entropy_with_logits
softmax
softmax_cross_entropy_with_logits
softmax_cross_entropy_with_logits_v2
softmax
space_to_batch
space_to_depth
sparse_softmax_cross_entropy_with_logits
static_bidirectional_rnn
static_rnn
static_state_saving_rnn
sufficient_statistics
weighted_cross_entropy_with_logits
weighted_moments
with_space_to_batch
xw_plus_b
» rnn_cell
» tf.profiler
» tf.python_io

tf.nn.softmax_cross_entropy_with_logits_v2

```
tf.nn.softmax_cross_entropy_with_logits_v2(  
    _sentinel=None,  
    labels=None,  
    logits=None,  
    dim=-1,  
    name=None  
)
```

Defined in [tensorflow/python/ops/nn_ops.py](#).

See the guide: [Neural Network > Classification](#)

Computes softmax cross entropy between `logits` and `labels`.

Measures the probability error in discrete classification tasks in which the classes are mutually exclusive (each entry is in exactly one class). For example, each CIFAR-10 image is labeled with one and only one label: an image can be a dog or a truck, but not both.

NOTE: While the classes are mutually exclusive, their probabilities need not be. All that is required is that each row of `labels` is a valid probability distribution. If they are not, the computation of the gradient will be incorrect.

If using exclusive `labels` (wherein one and only one class is true at a time), see [sparse_softmax_cross_entropy_with_logits](#).

WARNING: This op expects unscaled logits, since it performs a `softmax` on `logits` internally for efficiency. Do not call this op with the output of `softmax`, as it will produce incorrect results.

A common use case is to have `logits` and `labels` of shape `[batch_size, num_classes]`, but higher dimensions are supported, with the `dim` argument specifying the class dimension.

`logits` and `labels` must have the same `dtype` (either `float16`, `float32`, or `float64`).

Guardar y leer modelos

Guardar el modelo (stackoverflow)

```
# Create some variables.  
v1 = tf.get_variable("v1", shape=[3], initializer = tf.zeros_initializer)  
v2 = tf.get_variable("v2", shape=[5], initializer = tf.zeros_initializer)  
inc_v1 = v1.assign(v1+1)  
dec_v2 = v2.assign(v2-1)  
# Add an op to initialize the variables.  
init_op = tf.global_variables_initializer()  
# Add ops to save and restore all the variables.  
saver = tf.train.Saver()  
# Later, launch the model, initialize the variables, do some work, and save the  
# variables to disk.  
with tf.Session() as sess:  
    sess.run(init_op)  
    # Do some work with the model.  
    inc_v1.op.run()  
    dec_v2.op.run()  
    # Save the variables to disk.  
    save_path = saver.save(sess, "/tmp/model.ckpt")  
    print("Model saved in path: %s" % save_path)
```

Una mejor forma de crear variables

Guardar y leer modelos

Leer el modelo (stackoverflow)

```
tf.reset_default_graph()
# Create some variables.
v1 = tf.get_variable("v1", shape=[3])
v2 = tf.get_variable("v2", shape=[5])
# Add ops to save and restore all the variables.
saver = tf.train.Saver()
# Later, launch the model, use the saver to restore variables from disk, and
# do some work with the model.
with tf.Session() as sess:
    # Restore variables from disk.
    saver.restore(sess, "/tmp/model.ckpt")
    print("Model restored.")
    # Check the values of the variables
    print("v1 : %s" % v1.eval())
    print("v2 : %s" % v2.eval())
```

No se inicializan las variables

Capas de alto nivel

Modulo layer en TF 1.x: tiene dense, conv2d, dropout, etc..

The screenshot shows a web browser displaying the TensorFlow API documentation at www.tensorflow.org/api_docs/python/tf/layers/Dense. The page is for the Python API. On the left, there's a sidebar with navigation links for Overview, Python, JavaScript, C++, and Java. The 'Dense' link in the Python section is highlighted with a blue background. The main content area shows the `Dense` class definition:

```
__init__(  
    units,  
    activation=None,  
    use_bias=True,  
    kernel_initializer=None,  
    bias_initializer=tf.zeros_initializer(),  
    kernel_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    bias_constraint=None,  
    trainable=True,  
    name=None,  
    **kwargs  
)
```

Below the code, there are sections for **Properties** (graph) and **DEPRECATED FUNCTION**. A red warning box at the bottom states: **⚠ Warning: THIS FUNCTION IS DEPRECATED.** It will be removed in a future version. Instructions for updating: Stop using this property because tf.layers layers no longer track their graph.

The right sidebar contains links for Contents, Class Dense, Aliases, __init__, Properties, graph, and scope_name.

Capas de alto nivel

The screenshot shows a web browser displaying the TensorFlow API documentation for the `tf.layers.dense` function. The page has a navigation bar at the top with links for TensorFlow logo, Install, Learn, API, Resources, Community, More, Search, Language, GitHub, and Sign In. The main content area shows the `tf.layers.dense` function definition, its aliases, and a warning message. A sidebar on the left lists other API functions, and a sidebar on the right shows navigation links.

Aliases:

- `tf.compat.v1.layers.dense`

```
tf.layers.dense(  
    inputs,  
    units,  
    activation=None,  
    use_bias=True,  
    kernel_initializer=None,  
    bias_initializer=tf.zeros_initializer(),  
    kernel_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    bias_constraint=None,  
    trainable=True,  
    name=None,  
    reuse=None  
)
```

Warning: THIS FUNCTION IS DEPRECATED. It will be removed in a future version. Instructions for updating: Use `keras.layers.dense` instead.

This layer implements the operation: `outputs = activation(inputs * kernel + bias)` where `activation` is the activation function passed as the `activation` argument (if not `None`), `kernel` is a weights matrix created by the layer, and `bias` is a bias vector created by the layer (only if `use_bias` is `True`).

Arguments:

- inputs
- units
- activation
- use_bias
- kernel_initializer
- bias_initializer
- kernel_regularizer
- bias_regularizer
- activity_regularizer
- kernel_constraint
- bias_constraint
- trainable
- name
- reuse

Capas de alto nivel => Keras

The screenshot shows a GitHub issue page for a Tensorflow 2.0 deprecation concern. The URL in the address bar is `github.com/tensorflow/tensorflow/issues/26844`. The main title of the issue is "Tensorflow 2.0: please do not deprecate important functions!!! #26844". The status is "Closed" by erikchhwang on Mar 18, with 26 comments.

Comments:

- erikchhwang** commented on Mar 18 • edited
According to Tensorflow 2.0, many important functions are deprecated, such as:
`tf.layers.dense()`
`tf.layers.dropout()`
`tf.layers.flatten()`
`tf.layers.batch_normalization()`
...
Since these functions are very widely used, and also very helpful for building complicated models, would you please keep them in the future versions?
- galeone** commented on Mar 18
You have to use `tf.keras.layers` instead of `tf.layers` to build the models.
- erikchhwang** commented on Mar 18
there is no `dense()`, `dropout()`, etc. to use, no matter keras or layers

Assignees: No one assigned

Labels: None yet

Projects: None yet

Milestone: No milestone

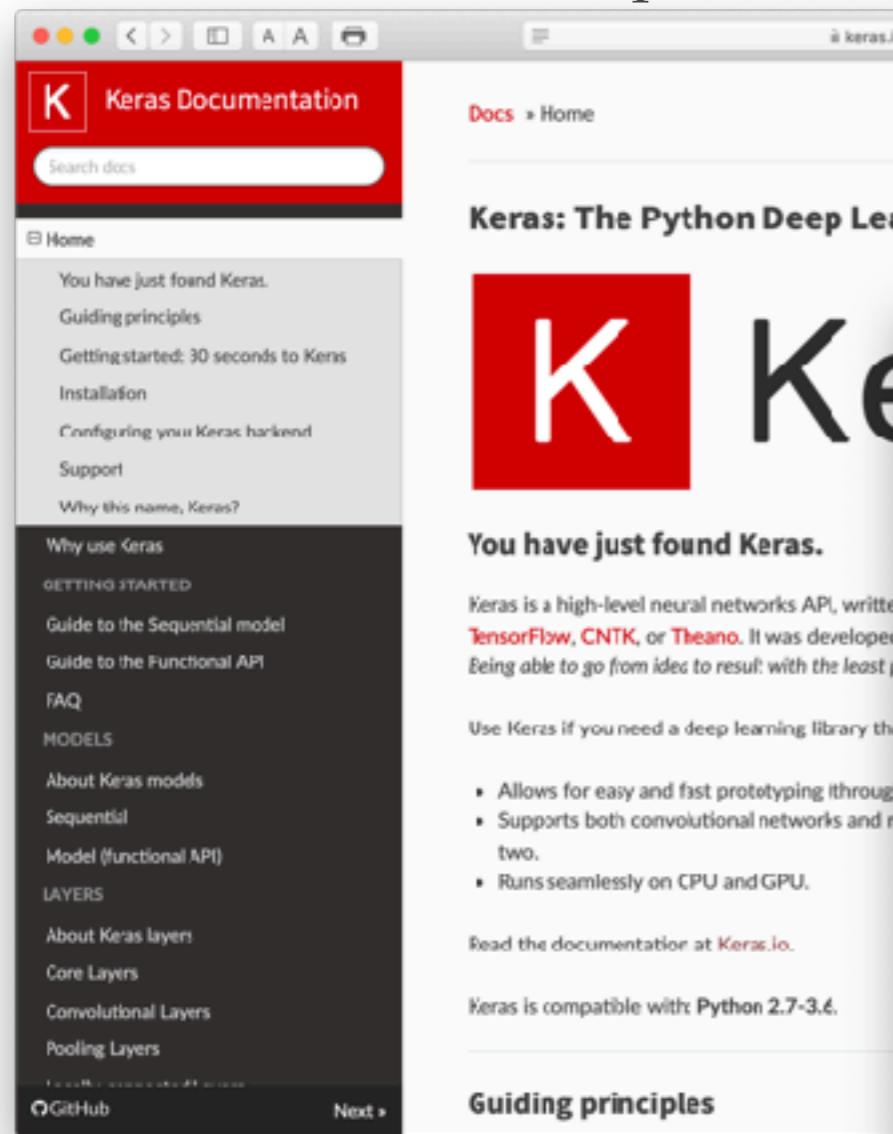
Notifications: Customize
You're not receiving notifications from this thread.

Participants: 8 participants (with small profile icons)

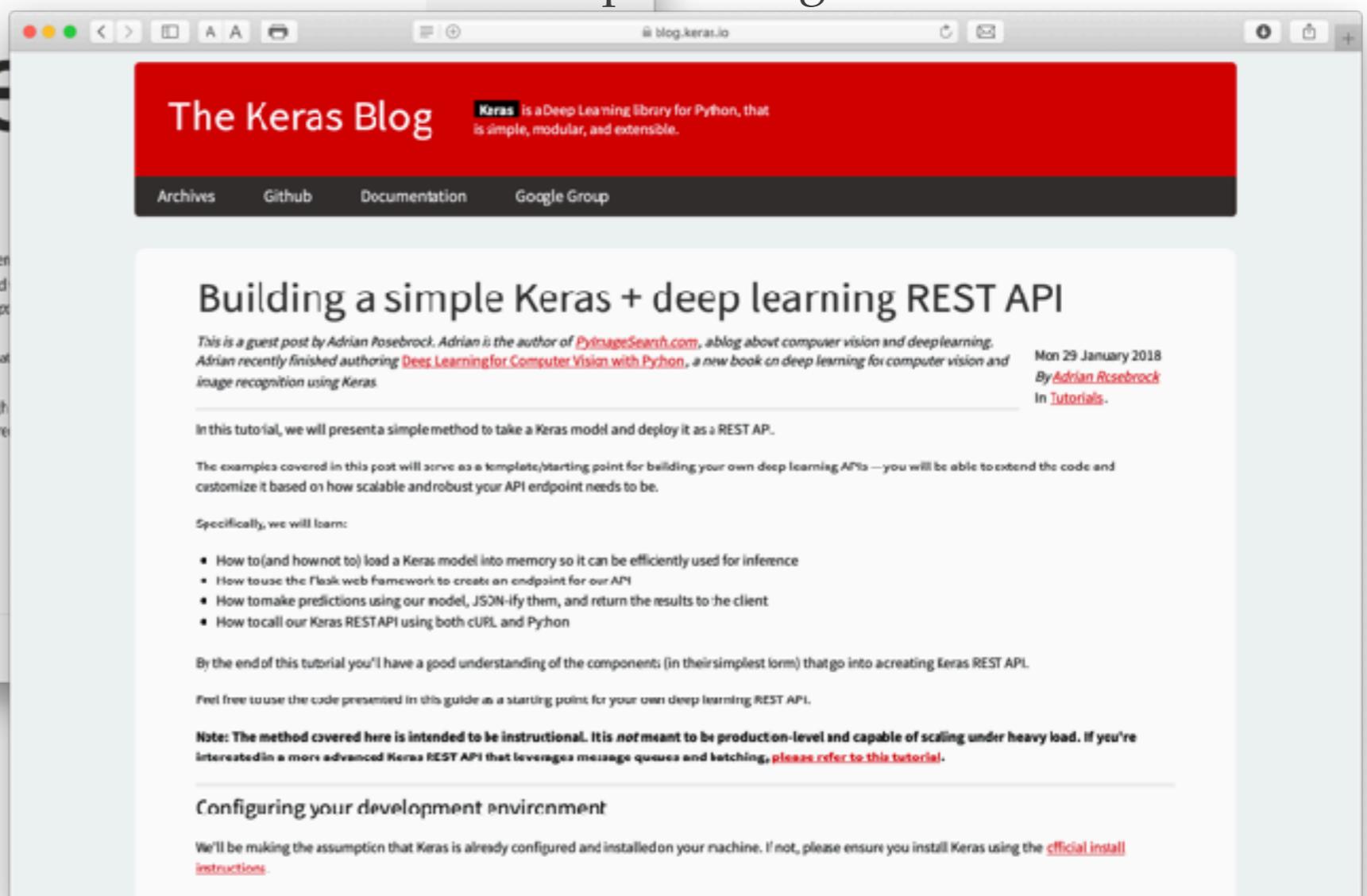
TensorFlow 2.0 with Keras

Keras

<https://keras.io>



The screenshot shows the Keras Documentation homepage. The top navigation bar includes a search bar and links for "Docs" and "Edit on GitHub". The main content area features a large red "K" logo and the title "Keras: The Python Deep Learning library". Below the title, there's a section titled "You have just found Keras." which explains that Keras is a high-level neural networks API built on top of TensorFlow, CNTK, or Theano. It highlights features like easy prototyping, support for convolutional networks, and GPU compatibility. A sidebar on the left contains links to various Keras components such as Guiding principles, Getting started, Installation, Configuration, Support, Why this name, and Why use Keras. Another sidebar at the bottom lists GETTING STARTED, MODELS, LAYERS, and GitHub.



The screenshot shows a blog post titled "Building a simple Keras + deep learning REST API". The post is by Adrian Rosebrock and was published on Mon 29 January 2018. It discusses how to take a Keras model and deploy it as a REST API. The author provides a guest post by Adrian Rosebrock, who is the author of PyImageSearch.com, a blog about computer vision and deep learning. The post covers how to load a Keras model into memory, use the Flask web framework to create an endpoint, make predictions using JSON, and return results to a client. It also notes that the examples can serve as a template for building your own deep learning APIs. The post concludes with a note that the method is instructional and not meant for production-level use, and a link to a more advanced tutorial.

<https://blog.keras.io>

TensorFlow 2.0 and Keras

- ❖ keras en tensorflow (tf.keras) esta unido a un solo backend, tensorflow
- ❖ keras de <https://keras.io> es la librería completa que soporta tensorflow, theano y CNTK. Si se va a utilizar con otro backend distinto a TF entonces necesitamos la librería completa

TensorFlow 2.0 with Keras

A screenshot of a Twitter post from Francois Chollet (@fchollet). The post is a tweet about the new release of multi-backend Keras 2.3.0. It includes a link to the GitHub repository, a list of features, and a recommendation to switch code to tf.keras. The GitHub logo is visible at the bottom.

François Chollet  
@fchollet

New release of multi-backend Keras:
2.3.0

[github.com/keras-team/keras...](https://github.com/keras-team/keras)

- First release of multi-backend Keras with full TF 2 support
- Continued support for Theano/CNTK
- Will be the last major release of multi-backend Keras

We recommend you switch your Keras code to `tf.keras`.

 **keras-team/keras**
Deep Learning for humans. Contribute to keras-team/keras development by creating an account on GitHub.
github.com/keras-team/keras

10:53 AM - 17 Sep 2019

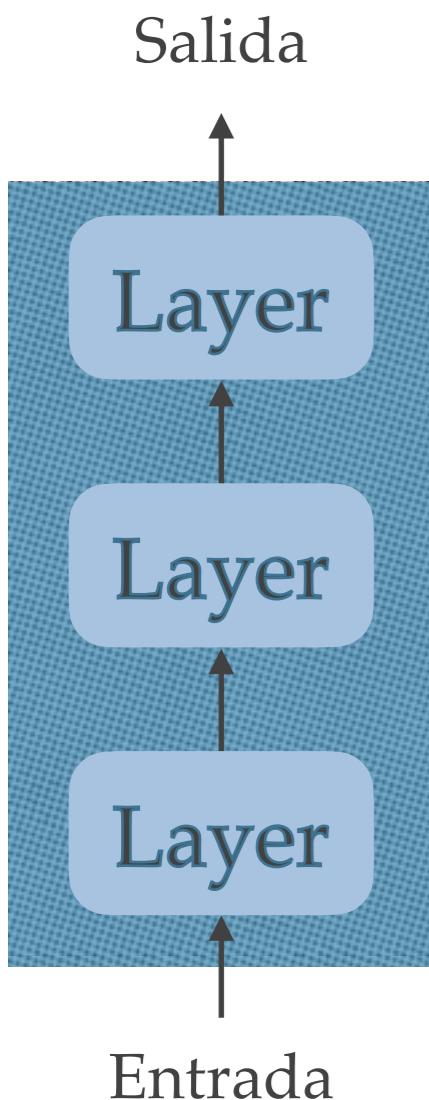
Configuración Keras

```
ariel@gpu-office:~/keras$ cat keras.json
{
    "floatx": "float32",
    "epsilon": 1e-07,
    "backend": "tensorflow",
    "image_data_format": "channels_last"
}ariel@gpu-office:~/keras$
```

- **image_data_format**: String, either "channels_last" or "channels_first". It specifies which data format convention Keras will follow. (`keras.backend.image_data_format()` returns it.)
- For 2D data (e.g. image), "channels_last" assumes (rows, cols, channels) while "channels_first" assumes (channels, rows, cols).
- For 3D data, "channels_last" assumes (conv_dim1, conv_dim2, conv_dim3, channels) while "channels_first" assumes (channels, conv_dim1, conv_dim2, conv_dim3).
- **epsilon**: Float, a numeric fuzzing constant used to avoid dividing by zero in some operations.
- **floatx**: String, "float16", "float32", or "float64". Default float precision.
- **backend**: String, "tensorflow", "theano", or "cntk".

Sequential vs. Model

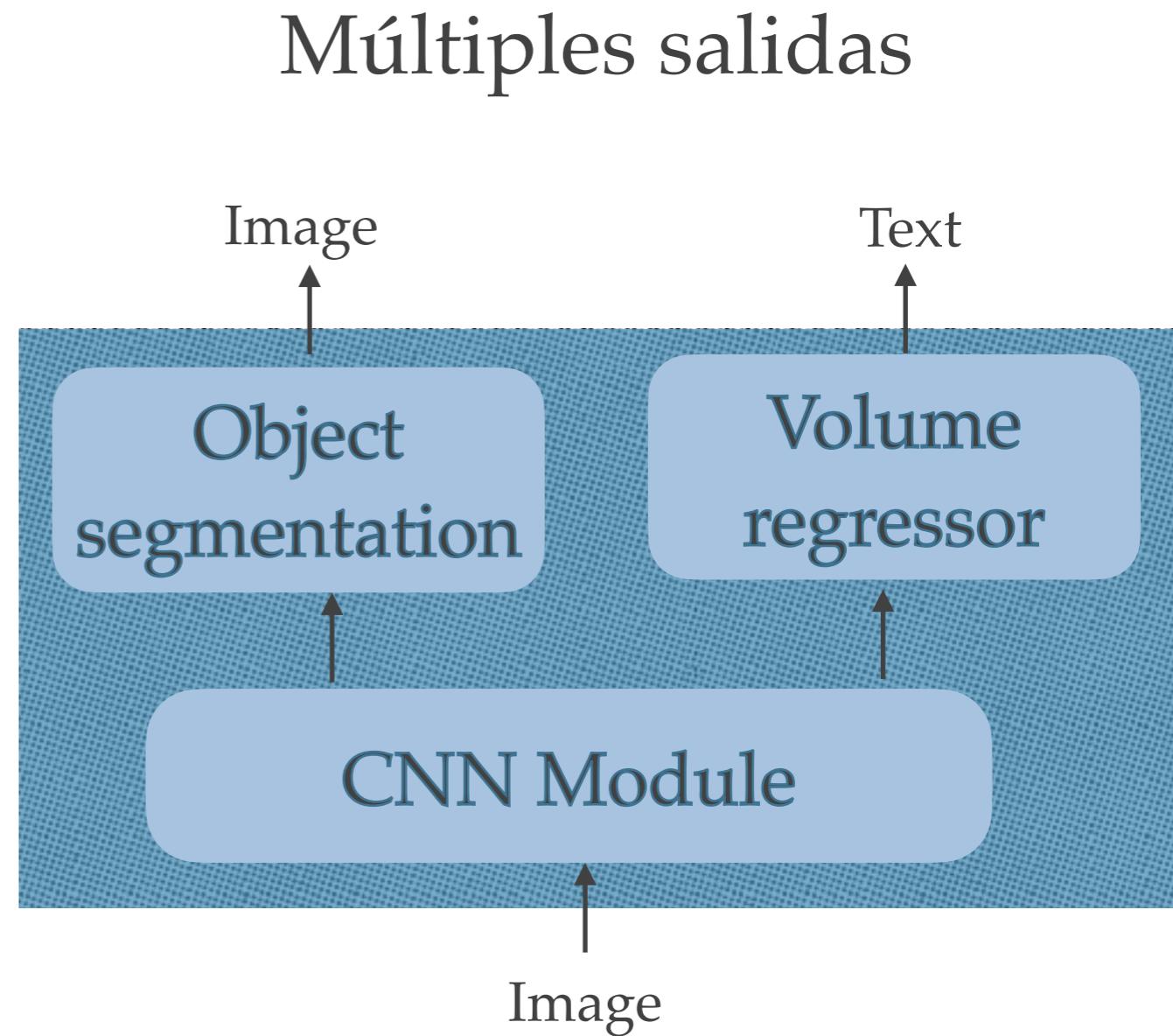
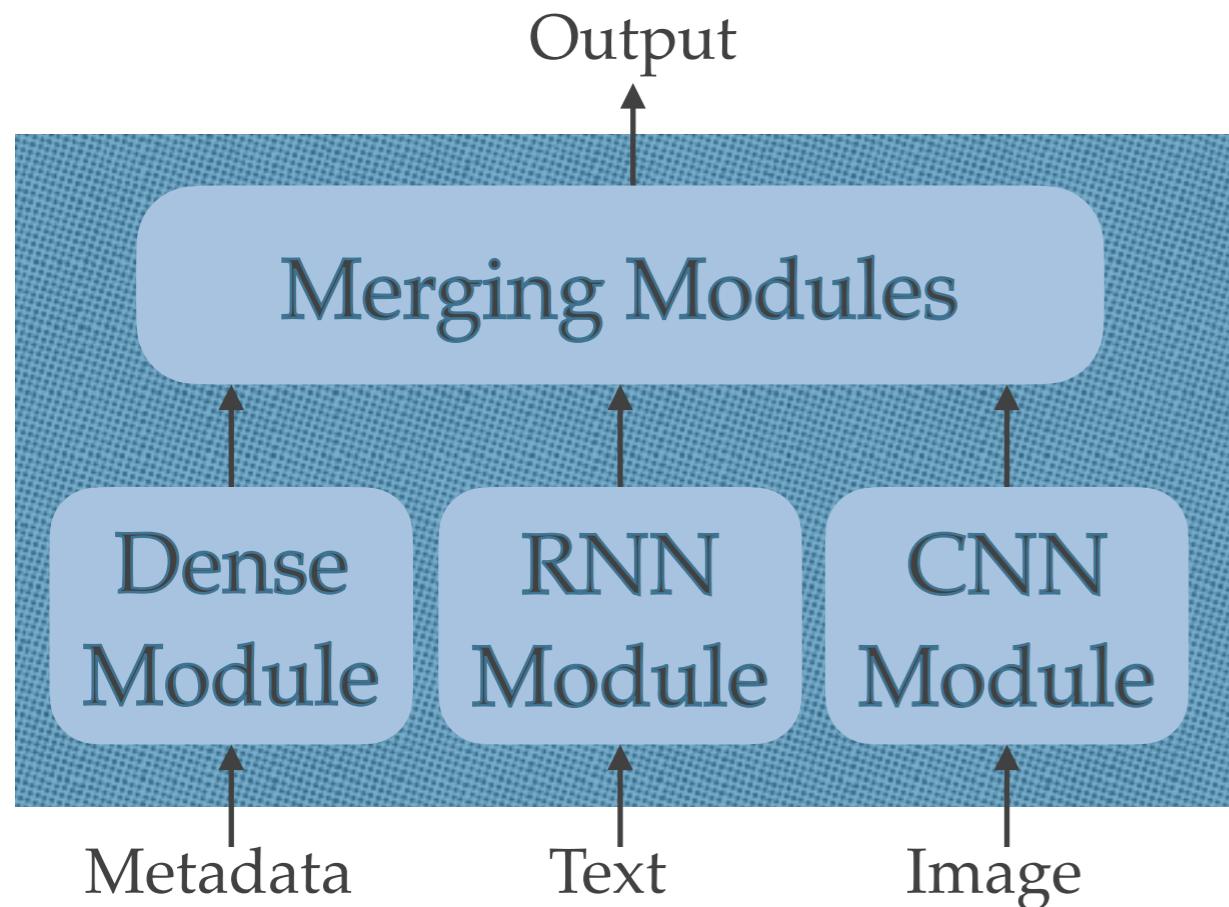
Una entrada y una salida conectada con diversas capas de forma lineal



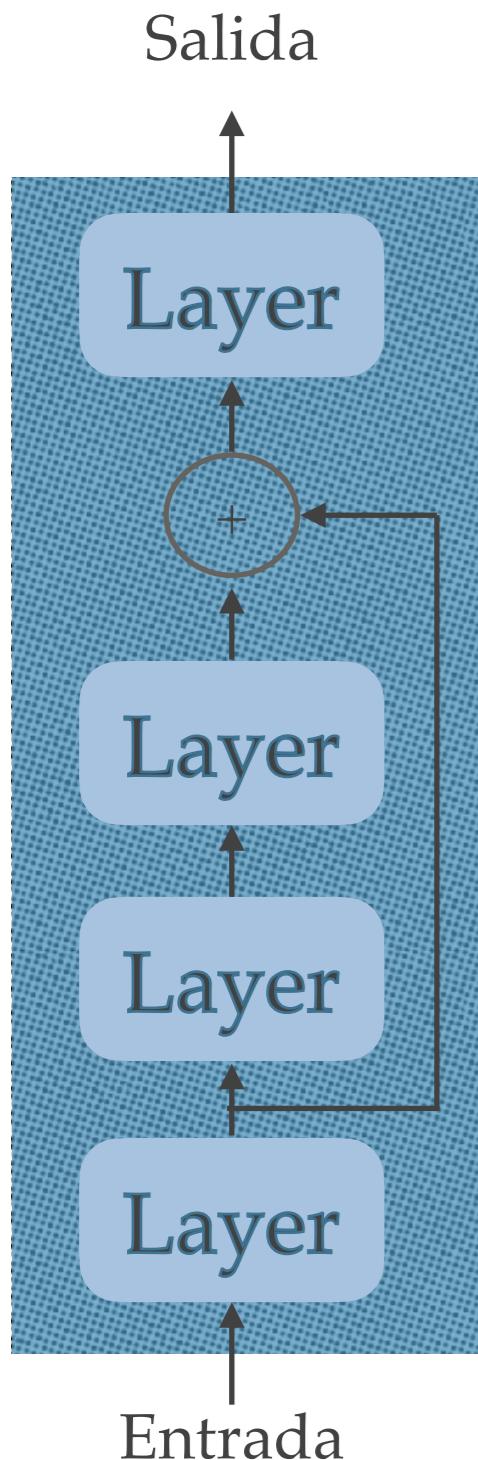
```
1 model = tf.keras.models.Sequential()  
2 model.add(tf.keras.layers.Dense(100, input_shape=(n_dim,)))  
3 model.add(tf.keras.layers.Dense(30,  
4 activation='linear', use_bias=True))
```

shape, en Input, o input_shape en Dense
o Conv no incluyen la dim de batch_size
o ejemplos. Es la dimensionalidad del
problema, es decir, la de un ejemplo

Sequential vs. Model

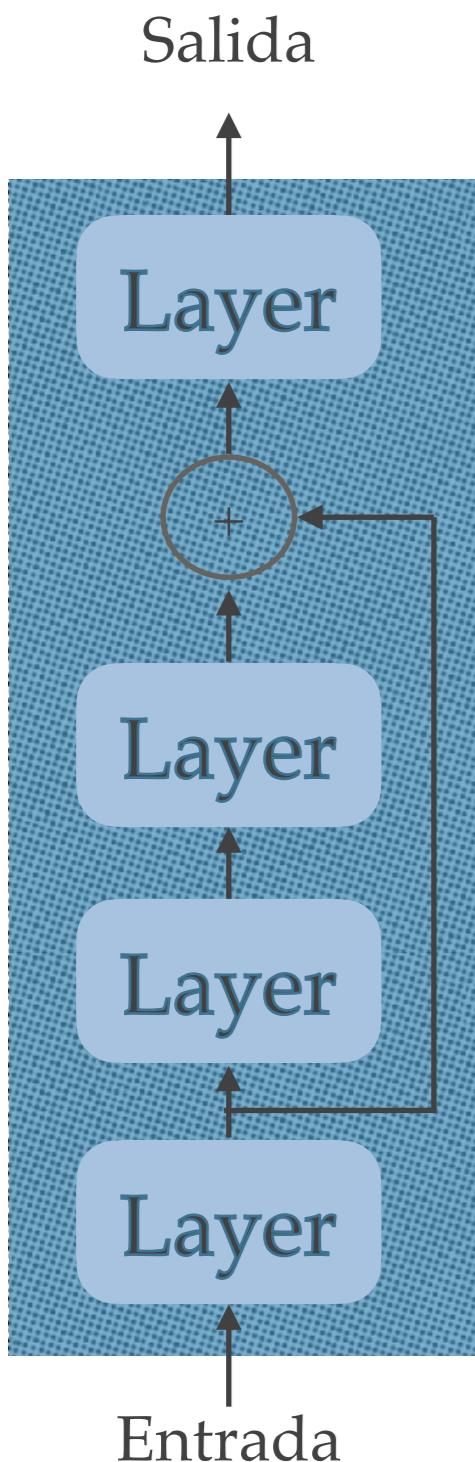


Sequential vs. Model



```
1 inputs = tf.keras.layers.Input(shape=(28,28,1))
2 l1 = tf.keras.layers.Dense(100, activation='relu')(inputs)
3 l2 = tf.keras.layers.Dense(100, activation='relu')(l1)
4 l3 = tf.keras.layers.Dense(100, activation='relu')(l2)
5 l4 = tf.keras.layers.Add()([l1,l3])
6 l5 = tf.keras.layers.Dense(200, activation='relu')(l4)
7 model = tf.keras.models.Model(inputs=inputs, outputs=l5)
```

model.summary()



```
1 inputs = tf.keras.layers.Input(shape=(28,28,1))
2 l1 = tf.keras.layers.Dense(100, activation='relu')(inputs)
3 l2 = tf.keras.layers.Dense(100, activation='relu')(l1)
4 l3 = tf.keras.layers.Dense(100, activation='relu')(l2)
5 l4 = tf.keras.layers.Add()([l1,l3])
6 l5 = tf.keras.layers.Dense(200, activation='relu')(l4)
7 model = tf.keras.models.Model(inputs=inputs, outputs=l5)
```

```
In [20]: model.summary()
Model: "model"

Layer (type)          Output Shape       Param #     Connected to
=====
input_3 (InputLayer)  [(None, 28, 28, 1)]  0
dense_4 (Dense)        (None, 28, 28, 100)  200        input_3[0][0]
dense_5 (Dense)        (None, 28, 28, 100)  10100      dense_4[0][0]
dense_6 (Dense)        (None, 28, 28, 100)  10100      dense_5[0][0]
add_1 (Add)            (None, 28, 28, 100)  0          dense_4[0][0]
                                            dense_6[0][0]
dense_7 (Dense)        (None, 28, 28, 200)  20200      add_1[0][0]
=====
Total params: 40,600
Trainable params: 40,600
Non-trainable params: 0
```

tf.keras.layers

- ❖ Los layers son clases que al ser invocados como funciones, normalmente con un tensor, devuelven otro tensor
 - ❖ Input, Dense, Conv1D (2D, 3D), Activation
 - ❖ MaxPooling1D (2D, 3D), AveragePooling1D (2D, 3D)
 - ❖ UpSampling1D (2D, 3D)
 - ❖ Lambda: convierte cualquier función en un layer
 - ❖ Add, Concatenate, Maximum, Minimum

Estos layers (mayúscula) se encuentran en su versión de función (minúscula)

Regularización

- ❖ tf.keras.layers
 - ❖ Dropout, BatchNormalization
- ❖ tf.keras.regularizers
 - ❖ l1, l2, l1_l2

Como veremos luego es posible pasar la métrica como texto al momento de compilar el modelo

Para el caso de acc se determina si usar Accuracy o CategoricalAccuracy según los datos

Module: `tf.keras.metrics`

TensorFlow 2.0 version

View source on GitHub

Aliases:

- Module `tf.compat.v1.keras.metrics`

Classes

Por ser clase, necesitamos crear el objeto

`class AUC`: Computes the approximate AUC (Area under the curve) via a Riemann sum.

`class Accuracy`: Calculates how often predictions matches labels.

`class BinaryAccuracy`: Calculates how often predictions matches labels.

`class BinaryCrossentropy`: Computes the crossentropy metric between the labels and predictions.

`class CategoricalAccuracy`: Calculates how often predictions matches labels.

`class CategoricalCrossentropy`: Computes the crossentropy metric between the labels and predictions.

`class CategoricalHinge`: Computes the categorical hinge metric between `y_true` and `y_pred`.

`class CosineSimilarity`: Computes the cosine similarity between the labels and predictions.

`class FalseNegatives`: Calculates the number of false negatives.

`class FalsePositives`: Calculates the number of false positives.

`class Hinge`: Computes the hinge metric between `y_true` and `y_pred`.

`class KLDivergence`: Computes Kullback-Leibler divergence metric between `y_true` and `y_pred`.

`class LogCoshError`: Computes the logarithm of the hyperbolic cosine of the prediction error.

`class Mean`: Computes the (weighted) mean of the given values.

MSE
MAE
Categorical cross entropy
.....

www.tensorflow.org/versions/r1.14/api_docs/python/tf/keras/metrics

TensorFlow

Install Learn API Resources Community Why TensorFlow

Search Language GitHub Sign in

Sequential
activations
applications
backend
callbacks
constraints
datasets
estimator
experimental
initializers
layers
losses
metrics
Overview
Accuracy
AUC
BinaryAccuracy
BinaryCrossentropy
binary_accuracy
CategoricalAccuracy
CategoricalCrossentropy
CategoricalHinge
categorical_accuracy
CosineSimilarity
deserialize
FalseNegatives
FalsePositives
get
Hinge
KLDivergence
LogCoshError
Mean
MeanAbsoluteError
MeanAbsolutePercentageError
MeanIoU
MeanRelativeError
MeanSquaredError
MeanSquaredLogarithmicError
MeanTensor
Metric

TruePositives : Calculates the number of true positives.

Functions

KLD(...)
MAE(...)
MAPE(...)
MSE(...)
MSLE(...)
binary_accuracy(...)
binary_crossentropy(...)
categorical_accuracy(...)
categorical_crossentropy(...) : Computes the categorical crossentropy loss.
cosine(...) : Computes the cosine similarity between labels and predictions.
cosine_proximity(...) : Computes the cosine similarity between labels and predictions.
deserialize(...)
get(...)
hinge(...) : Computes the hinge loss between y_true and y_pred.
kld(...)
kullback_leibler_divergence(...)
mae(...)
mape(...)
mean_absolute_error(...)
mean_absolute_percentage_error(...)

Contents
Aliases
Classes
Functions

Se importan y usan directamente
MSE(y_true, y_pred)

La mayoría de estas funciones son alias
de las funciones de costo

www.tensorflow.org/versions/r1.4/api_docs/python/tf/keras/losses

TensorFlow

Install Learn API Resources Community Why TensorFlow

Search Language GitHub Sign in

tf.keras

- Overview
- Input
- Model
- Sequential
- activations
- applications
- backend
- callbacks
- constraints
- datasets
- estimator
- experimental
- initializers
- layers
- losses

Overview

BinaryCrossentropy
binary_crossentropy
CategoricalCrossentropy
CategoricalHinge
categorical_crossentropy
categorical_hinge
cosine
CosineSimilarity
deserialize
get
Hinge
hinge
Huber
KLD
KLDivergence
LogCosh
logcosh
Loss
MAE
MAPE
MeanAbsoluteError
MeanAbsolutePercentageError

Module: tf.keras.losses

TensorFlow 2.0 version View source on GitHub

Aliases:

- Module `tf.compat.v1.keras.losses`

Classes

Por ser clase, necesitamos crear el objeto

`class BinaryCrossentropy`: Computes the cross-entropy loss between true labels and predicted labels.

`class CategoricalCrossentropy`: Computes the crossentropy loss between the labels and predictions.

`class CategoricalHinge`: Computes the categorical hinge loss between `y_true` and `y_pred`.

`class CosineSimilarity`: Computes the cosine similarity between `y_true` and `y_pred`.

`class Hinge`: Computes the hinge loss between `y_true` and `y_pred`.

`class Huber`: Computes the Huber loss between `y_true` and `y_pred`.

`class KLDivergence`: Computes Kullback Leibler divergence loss between `y_true` and `y_pred`.

`class LogCosh`: Computes the logarithm of the hyperbolic cosine of the prediction error.

`class Loss`: Loss base class.

`class MeanAbsoluteError`: Computes the mean of absolute difference between labels and predictions.

`class MeanAbsolutePercentageError`: Computes the mean absolute percentage error between `y_true` and `y_pred`.

`class MeanSquaredError`: Computes the mean of squares of errors between labels and predictions.

`class MeanSquaredLogarithmicError`: Computes the mean squared logarithmic error between `y_true` and `y_pred`.

`class Poisson`: Computes the Poisson loss between `y_true` and `y_pred`.

Contents
Aliases
Classes
Functions

www.tensorflow.org/versions/r1.4/api_docs/python/tf/keras/losses

TensorFlow

Install Learn API Resources Community Why TensorFlow

Search Language GitHub Sign in

tf.keras

- Overview
- Input
- Model
- Sequential

activations

applications

backend

callbacks

constraints

datasets

estimator

experimental

initializers

layers

losses

- Overview
- BinaryCrossentropy
- binary_crossentropy
- CategoricalCrossentropy
- CategoricalHinge
- categorical_crossentropy
- categorical_hinge
- cosine
- CosineSimilarity
- deserialize
- get
- Hinge
- hinge
- Huber
- KLD
- KLDivergence
- LogCosh
- logcosh
- Loss
- MAE
- MAPE
- MeanAbsoluteError
- MeanAbsolutePercentageError
- MeanSquaredError

Functions

KLD(...)

MAE(...)

MAPE(...)

MSE(...)

MSLE(...)

binary_crossentropy(...)

categorical_crossentropy(...): Computes the categorical crossentropy loss.

categorical_hinge(...)

cosine(...): Computes the cosine similarity between labels and predictions.

cosine_proximity(...): Computes the cosine proximity between labels and predictions.

cosine_similarity(...): Computes the cosine similarity between labels and predictions.

deserialize(...)

get(...)

hinge(...): Computes the hinge loss between `y_true` and `y_pred`.

kld(...)

kullback_leibler_divergence(...)

logcosh(...): Logarithm of the hyperbolic cosine of the prediction error.

mae(...)

mape(...)

mean_absolute_error(...)

mean_absolute_percentage_error(...)

mean_squared_error(...)

Contents

Aliases

Classes

Functions

Se importan y usan directamente
MSE(`y_true`, `y_pred`)

www.tensorflow.org/versions/r1.14/api_docs/python/tf/keras/optimizers

TensorFlow

Install Learn API Resources Community Why TensorFlow

Search Language GitHub Sign in

Overview Python JavaScript C++ Java

Sequential
activations
applications
backend
callbacks
constraints
datasets
estimator
experimental
initializers
layers
losses
metrics
mixed_precision
models
optimizers
schedules
preprocessing
regularizers
utils
wrappers
tf.layers
tf.linsig
tf.lite
tf.logging

Module: tf.keras.optimizers

[TensorFlow 2.0 version](#) [View source on GitHub](#)

Aliases:

- Module `tf.compat.v1.keras.optimizers`

Modules

`schedules` module

Classes

`Adadelta`: Optimizer that implements the Adadelta algorithm.

`Adagrad`: Optimizer that implements the Adagrad algorithm.

`Adam`: Optimizer that implements the Adam algorithm.

`Adamax`: Optimizer that implements the Adamax algorithm.

`Ftrl`: Optimizer that implements the FTRL algorithm.

`Nadam`: Optimizer that implements the NAdam algorithm.

`Optimizer`: Updated base class for optimizers.

`RMSprop`: Optimizer that implements the RMSprop algorithm.

`SGD`: Stochastic gradient descent and momentum optimizer.

Contents
Aliases
Modules
Classes
Functions

Compilar, entrenar y predecir

```
13 model = tf.keras.Model(inputs=x, outputs=l2)
14 optimizer = tf.keras.optimizers.SGD(learning_rate=1)
15 model.compile(optimizer, loss=tf.keras.losses.MSE, metrics=[my_acc, 'acc', 'mse'])
16 model.fit(x_data, y_data, epochs=100, validation_data=(x_data, y_data),
17             batch_size=4, verbose=2)
18 y_pred = model.predict(x_data)
```

Multiples outputs: loss y train

- ❖ Se usa un diccionario (si definimos nombres en los layers) o una lista para definir la entrada y/o la función de costo para cada salida
- ❖ Si solo se define una función de costo, entonces se aplica la misma función para todas las salidas
- ❖ La función a minimizar será la suma de las funciones de costo
- ❖ Se puede definir la contribución de cada loss mediante el argumento `loss_weights` al compilar el modelo

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy',  
              loss_weights=[1., 0.2])
```

```
model.fit([headline_data, additional_data], [headline_labels, additional_labels],  
          epochs=50, batch_size=32)
```

Multiples outputs: loss y train

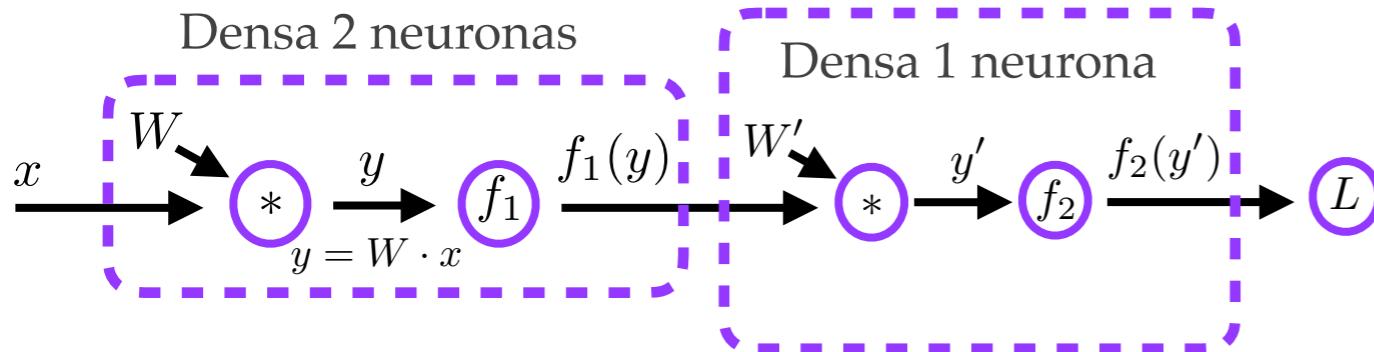
- ❖ Se usa un diccionario (**si definimos nombres en los layers**) o una lista para definir la entrada y/o la función de costo para cada salida
- ❖ Si solo se define una función de costo, entonces se aplica la misma función para todas las salidas
- ❖ La función a minimizar será la suma de las funciones de costo
- ❖ Se puede definir la contribución de cada loss mediante el argumento `loss_weights` al compilar el modelo

```
model.compile(optimizer='rmsprop',
              loss={'main_output': 'binary_crossentropy', 'aux_output': 'binary_crossentropy'},
              loss_weights={'main_output': 1., 'aux_output': 0.2})
```

And trained it via:

```
model.fit({'main_input': headline_data, 'aux_input': additional_data},
          {'main_output': headline_labels, 'aux_output': additional_labels},
          epochs=50, batch_size=32)
```

Problema 2.6 en Keras



$$f_1(x) = \tanh(x) \quad f_2(x) = \tanh(x)$$

$$\text{MSE} = L(y') = \frac{1}{N} \sum_i \|y_t^{(i)} - y'^{(i)}\|_2^2$$

Armamos el grafo
No hay computo

Entrenamos

Ejecutamos el grafo y
modificamos las variables

```
5 def my_acc(y_true, y_pred):
6     acc = tf.reduce_mean(tf.cast(tf.less_equal(tf.abs(y_true-y_pred), 0.1), tf.float32))
7     return acc
8
9 x = tf.keras.layers.Input(shape=(2,))
10 l1 = tf.keras.layers.Dense(units=2, activation='tanh', use_bias=True,
11     kernel_regularizer=tf.keras.regularizers.l2(1e-3))(x)
12 l2 = tf.keras.layers.Dense(units=1, activation='tanh', use_bias=True)(l1)
13 model = tf.keras.Model(inputs=x, outputs=l2)
14 optimizer = tf.keras.optimizers.SGD(learning_rate=1)
15 model.compile(optimizer, loss=tf.keras.losses.MSE, metrics=[my_acc, 'acc', 'mse'])
16 model.fit(x_data, y_data, epochs=100, validation_data=(x_data, y_data),
17             batch_size=4, verbose=2)
18 y_pred = model.predict(x_data)
```

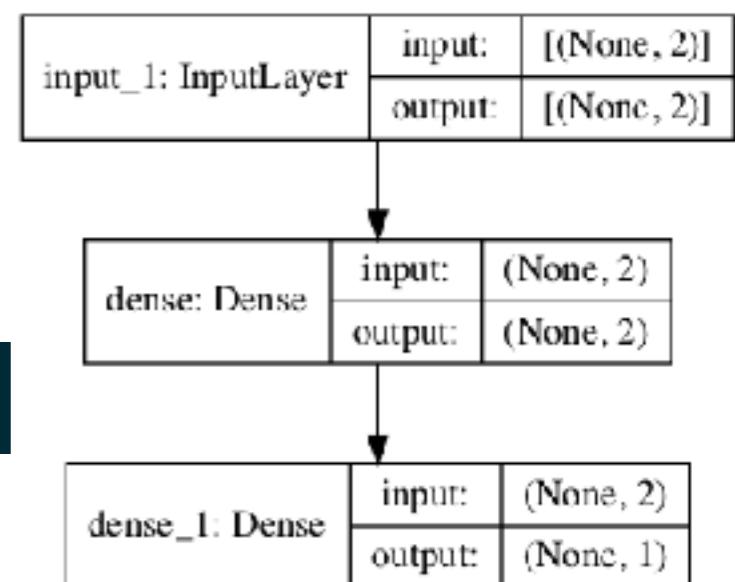
Problema 2.6 en Keras

- ❖ Observemos el modelo con el método `summary()`

```
In [12]: model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 2]	0
dense (Dense)	(None, 2)	6
dense_1 (Dense)	(None, 1)	3
Total params: 9		
Trainable params: 9		
Non-trainable params: 0		



- ❖ Exportamos el grafo

```
433 from tensorflow.keras.utils import plot_model  
434 plot_model(model, show_shapes=True, to_file='model.png')
```

The screenshot shows a web browser displaying the TensorFlow API documentation. The URL is www.tensorflow.org/versions/r1.14/api_docs/python/tf/keras/callbacks. The page title is "Module: tf.keras.callbacks". On the left, there is a sidebar with a tree view of TensorFlow modules. The "callbacks" module is expanded, and its "Overview" and "CSVLogger" classes are highlighted with a green border. At the top right, there is a search bar, language selection, GitHub link, and sign-in button. The main content area contains sections for "Aliases" (listing "Module tf.compat.v1.keras.callbacks") and "Classes" (listing various callback classes with their descriptions). The "CSVLogger" class is also highlighted with a green border.

Module: **tf.keras.callbacks**

[TensorFlow 2.0 version](#) [View source on GitHub](#)

Aliases:

- Module `tf.compat.v1.keras.callbacks`

Classes

`class BaseLogger`: Callback that accumulates epoch averages of metrics.

`class CSVLogger`: Callback that streams epoch results to a csv file.

`class Callback`: Abstract base class used to build new callbacks.

`class EarlyStopping`: Stop training when a monitored quantity has stopped improving.

`class History`: Callback that records events into a `History` object.

`class LambdaCallback`: Callback for creating simple, custom callbacks on-the-fly.

`class LearningRateScheduler`: Learning rate scheduler.

`class ModelCheckpoint`: Save the model after every epoch.

`class ProgbarLogger`: Callback that prints metrics to stdout.

`class ReduceLROnPlateau`: Reduce learning rate when a metric has stopped improving.

`class RemoteMonitor`: Callback used to stream events to a server.

`class TensorBoard`: Enable visualizations for TensorBoard.

`class TerminateOnNaN`: Callback that terminates training when a NaN loss is encountered.

Nos permiten interactuar con el modelo durante el entrenamiento, por ejemplo, para ajustar dinámicamente algunos parámetros como el learning rate

Callbacks are passed to the model via the callbacks argument in fit, which takes a list of callbacks. You can pass any number of callbacks.

```
import keras

> callbacks_list = [
    keras.callbacks.EarlyStopping(
        monitor='acc',
        patience=1,
    ),
    keras.callbacks.ModelCheckpoint(
        filepath='my_model.h5',
        monitor='val_loss',
        save_best_only=True,
    )
]

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

model.fit(x, y,
          epochs=10,
          batch_size=32,
          callbacks=callbacks_list,
          validation_data=(x_val, y_val))
```

Interrupts training when improvement stops

Monitors the model's validation accuracy

Interrupts training when accuracy has stopped improving for more than one epoch (that is, two epochs)

Saves the current weights after every epoch

Path to the destination model file

These two arguments mean you won't overwrite the model file unless val_loss has improved, which allows you to keep the best model seen during training.

You monitor accuracy, so it should be part of the model's metrics.

Note that because the callback will monitor validation loss and validation accuracy, you need to pass validation_data to the call to fit.

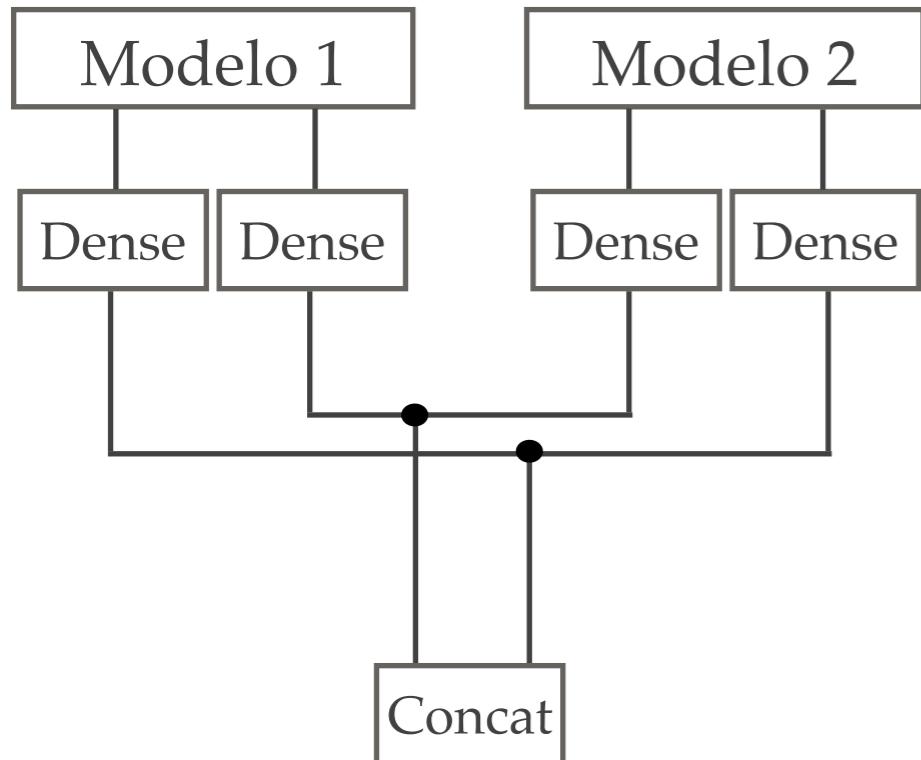
Crear un nuevo layer

- ❖ Lambda: útil para hacer operaciones aritméticas

```
1 import tensorflow.keras.backend as K
2 def antirectifier(x):
3     x -= K.mean(x, axis=1, keepdims=True)
4     x = K.l2_normalize(x, axis=1)
5     pos = K.relu(x)
6     neg = K.relu(-x)
7     return K.concatenate([pos, neg], axis=1)
8
9 model.add(tf.keras.layers.Lambda(antirectifier))
10 l10 = tf.keras.layers.Lambda(antirectifier)(li)
```

Crear un nuevo layer

- ❖ Lambda mezclando TF



```
2 def cross(l1, l2):
3     return tf.linalg.cross(l1,l2)
4
5 d1 = tf.keras.layers.Dense(64)(Model_1.output)
6 d2 = tf.keras.layers.Dense(64)(Model_1.output)
7 d3 = tf.keras.layers.Dense(64)(Model_2.output)
8 d4 = tf.keras.layers.Dense(64)(Model_2.output)
9
10 cross1 = tf.keras.layers.Lambda(cross)([d1,d4])
11 cross2 = tf.keras.layers.Lambda(cross)([d2,d3])
12
13 output = tf.keras.layers.concatenate([cross1,cross2])
```

Crear un nuevo layer

- ❖ Clase Layer con múltiples entradas y salidas

```
3 from tensorflow.keras import backend as K
4 from tensorflow.keras.layers import Layer
6 class MyLayer(Layer):
8     def __init__(self, output_dim, **kwargs):
9         self.output_dim = output_dim
10        super(MyLayer, self).__init__(**kwargs)
11
12    def build(self, input_shape): ← Se definen las variables
13        assert isinstance(input_shape, list) internas del layer
14        # Create a trainable weight variable for this layer.
15        self.kernel = self.add_weight(name='kernel',
16                                     shape=(input_shape[0][1], self.output_dim),
17                                     initializer='uniform',
18                                     trainable=True)
se puede hacer a mano con el backend K.variable()
19        super(MyLayer, self).build(input_shape) # Be sure to call this at the end
20
21    def call(self, x): ← Se definen la lógica
22        assert isinstance(x, list) del layer
23        a, b = x
24        return [K.dot(a, self.kernel) + b, K.mean(b, axis=-1)]
25
26    def compute_output_shape(self, input_shape): ← Si se modifica el tamaño
27        assert isinstance(input_shape, list) de la entrada hay que
28        shape_a, shape_b = input_shape especificarlo
29        return [(shape_a[0], self.output_dim), shape_b[:-1]]
```

Crear una métricas



y_true e y_pred son tensores

```
5 def my_metric(y_true, y_pred):  
6     val = . . . . .  
7     return val
```

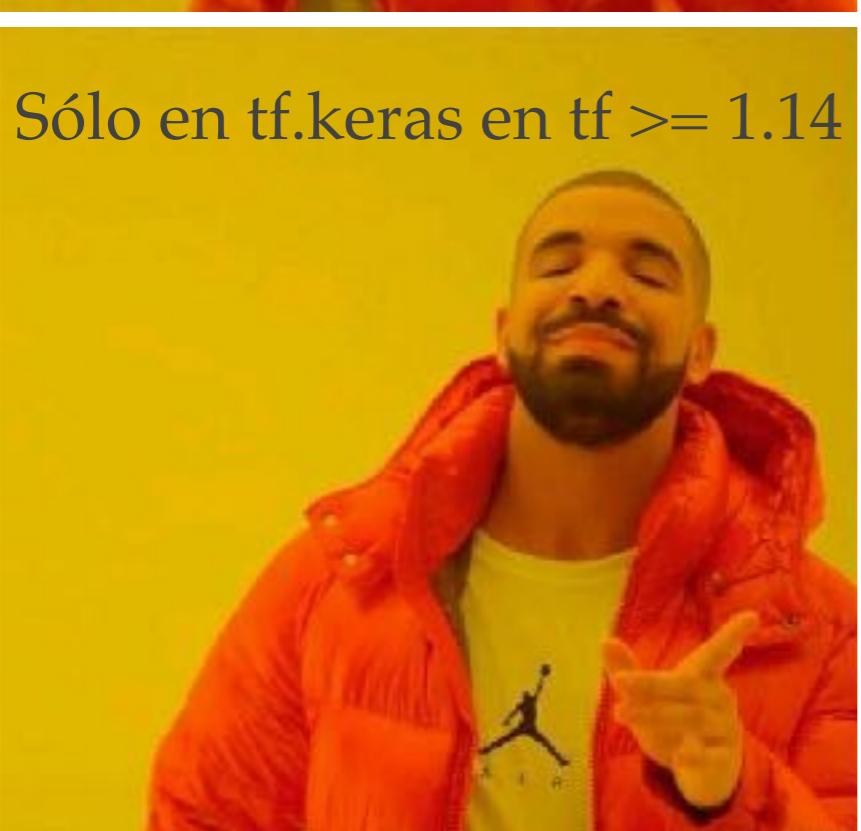
```
6 class MyMetric(object):  
7     def __init__(self, a1, a2, a3, *args):  
8         self.a1 = a1  
9         self.__name__ = 'MyMetric'  
10    pass  
11  
12    def __call__(self, y_true, y_pred):  
13        if self.a1 is not None:  
14            val = ...  
15            pass  
16        else:  
17            val = ...  
18            pass  
19  
20    return val  
21
```

Crear una métricas



y_true e y_pred son tensores

```
5 def my_metric(y_true, y_pred):  
6     val = . . . . .  
7     return val
```



Sólo en tf.keras en tf >= 1.14

```
1 class BinaryTruePositives(tf.keras.metrics.Metric):  
2     def __init__(self, name='binary_true_positives', **kwargs):  
3         super(BinaryTruePositives, self).__init__(name=name,  
4                                         **kwargs)  
5         self.true_positives = self.add_weight(name='tp',  
6                                             initializer='zeros')  
7  
8     def update_state(self, y_true, y_pred, sample_weight=None):  
9         y_true = tf.cast(y_true, tf.bool)  
10        y_pred = tf.cast(y_pred, tf.bool)  
11        values = tf.logical_and(tf.equal(y_true, True),  
12                                tf.equal(y_pred, True))  
13        values = tf.cast(values, self.dtype)  
14        if sample_weight is not None:  
15            sample_weight = tf.cast(sample_weight, self.dtype)  
16            sample_weight = tf.broadcast_weights(sample_weight,  
17                                                    values)  
18            values = tf.multiply(values, sample_weight)  
19            self.true_positives.assign_add(tf.reduce_sum(values))  
20  
21    def result(self):  
22        return self.true_positives
```

Crear una función de costo



y_true e y_pred son tensores

```
5 def my_loss(y_true, y_pred):  
6     val = . . . . .  
7     return val
```

```
6 class MyLoss(object):  
7     def __init__(self, a1, a2, a3, *args):  
8         self.a1 = a1  
9         self.__name__ = 'MyMetric'  
10    pass  
11  
12    def __call__(self, y_true, y_pred):  
13        if self.a1 is not None:  
14            val = ...  
15            pass  
16        else:  
17            val = ...  
18            pass  
19  
20    return val
```

Crear una función de costo



y_true e y_pred son **tensores**

```
5 def my_loss(y_true, y_pred):  
6     val = . . . . .  
7     return val
```

al ser llamados desde model.fit() y_true e y_pred
son **tensores**

```
1 class MeanSquaredError(tf.keras.losses.Loss):  
2     def call(self, y_true, y_pred):  
3         y_pred = tf.convert_to_tensor(y_pred)  
4         y_true = tf.cast(y_pred, y_pred.dtype)  
5         values = tf.logical_and(tf.equal(y_true, True),  
6                                tf.equal(y_pred, True))  
7         return K.mean(K.square(y_pred-y_true), axis=-1)
```

Crear un nuevo callback

Hay que heredar de la clase Callback e implementar (los que se necesiten) los métodos:

- ❖ set_model
- ❖ on_epoch_begin, on_epoch_end
- ❖ on_batch_begin, on_batch_end
- ❖ on_train_begin, on_train_end

```

1 import os
2 import tensorflow.keras as keras
3 class ActivationLogger(keras.callbacks.Callback):
4
5     def __init__(self, filepath, layer2save=None):
6         self.filepath = filepath
7         self.layer2save = layer2save
8
9     def set_model(self, model): ← Llamado antes de entrenar para
10        self.model = model           informar que modelo se utiliza
11        if self.layer2save is not None:
12            layer_outputs = model.layers[self.layer2save]
13        else:
14            layer_outputs = [layer.output for layer in model.layers]
15        self.activation_model = keras.models.Model(model.input,
16                                              layer_outputs)

```

Armamos un modelo para retornar la activación

```

19     def on_epoch_end(self, epoch, logs=None):
20         if self.validation_data is None:
21             raise RuntimeError('Requires validation data')
22
23         validation_sample = self.validation_data[0][0:1]
24         activations = self.activation_model.predict(validation_sample)
25         fname = 'activation_at_epoch_' + str(epoch)
26         if self.layer2save is not None:
27             fname += '_layer_' + str(self.layer2save)
28         f = open(os.path.join(self.filepath, fname + '.npy'), 'w')
29         np.savez(f, activations)
30         f.close()

```

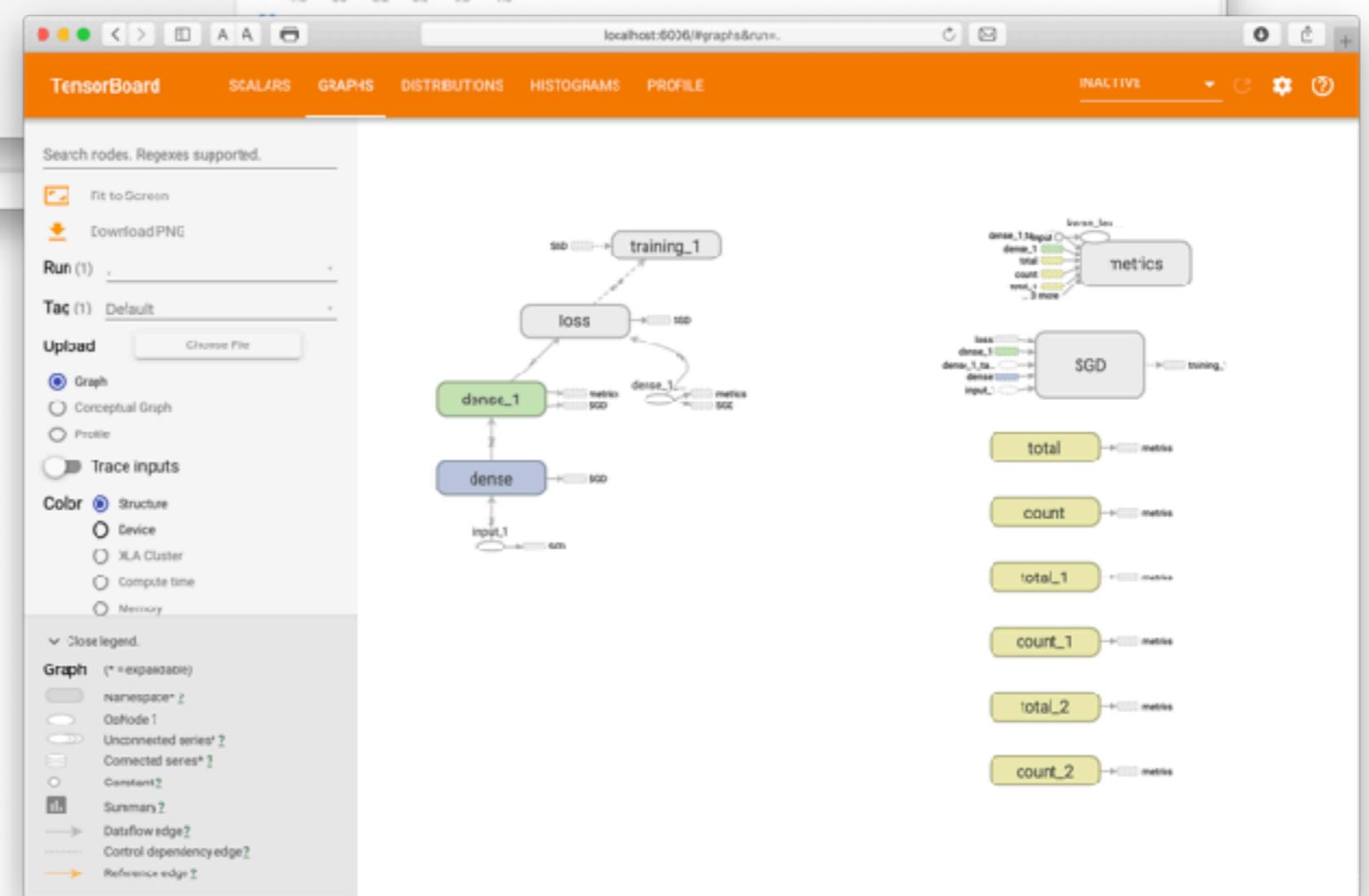
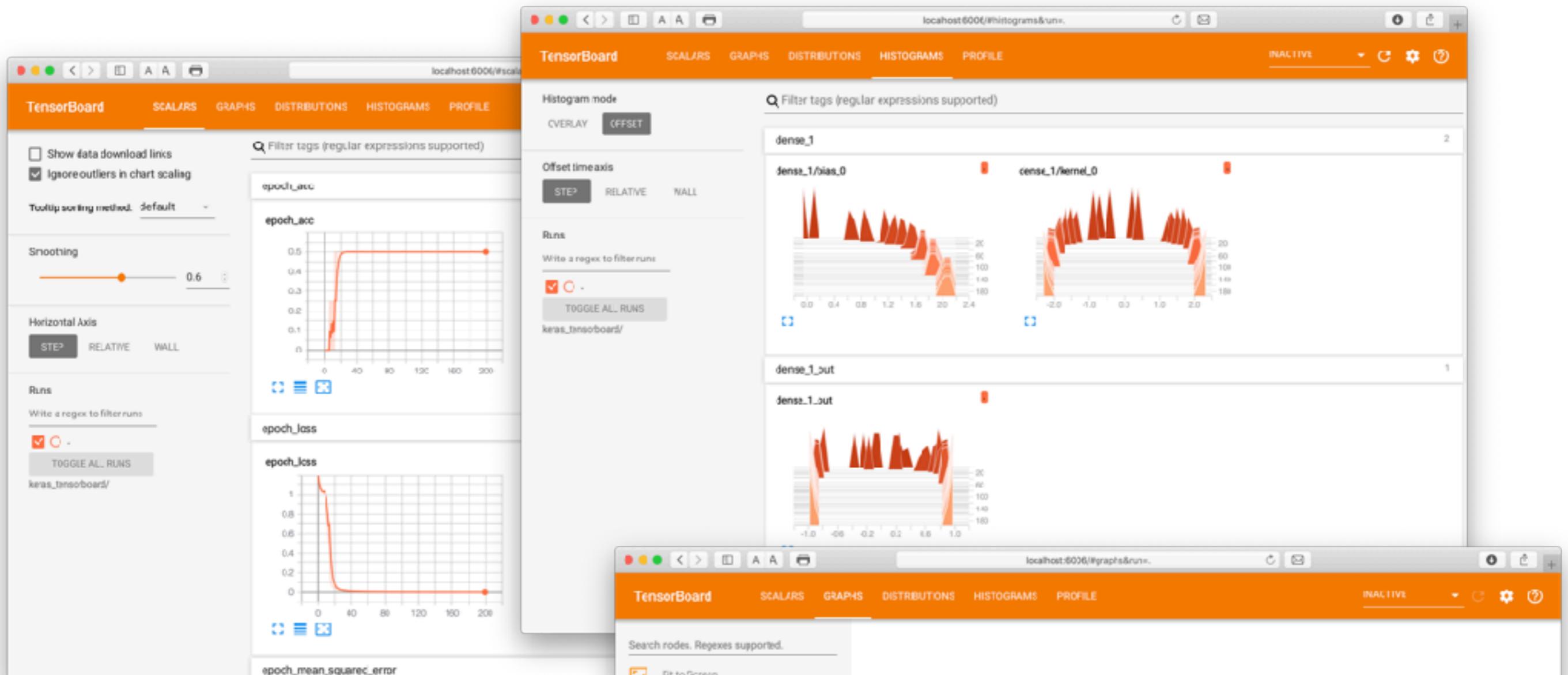
Keras + TensorBoard

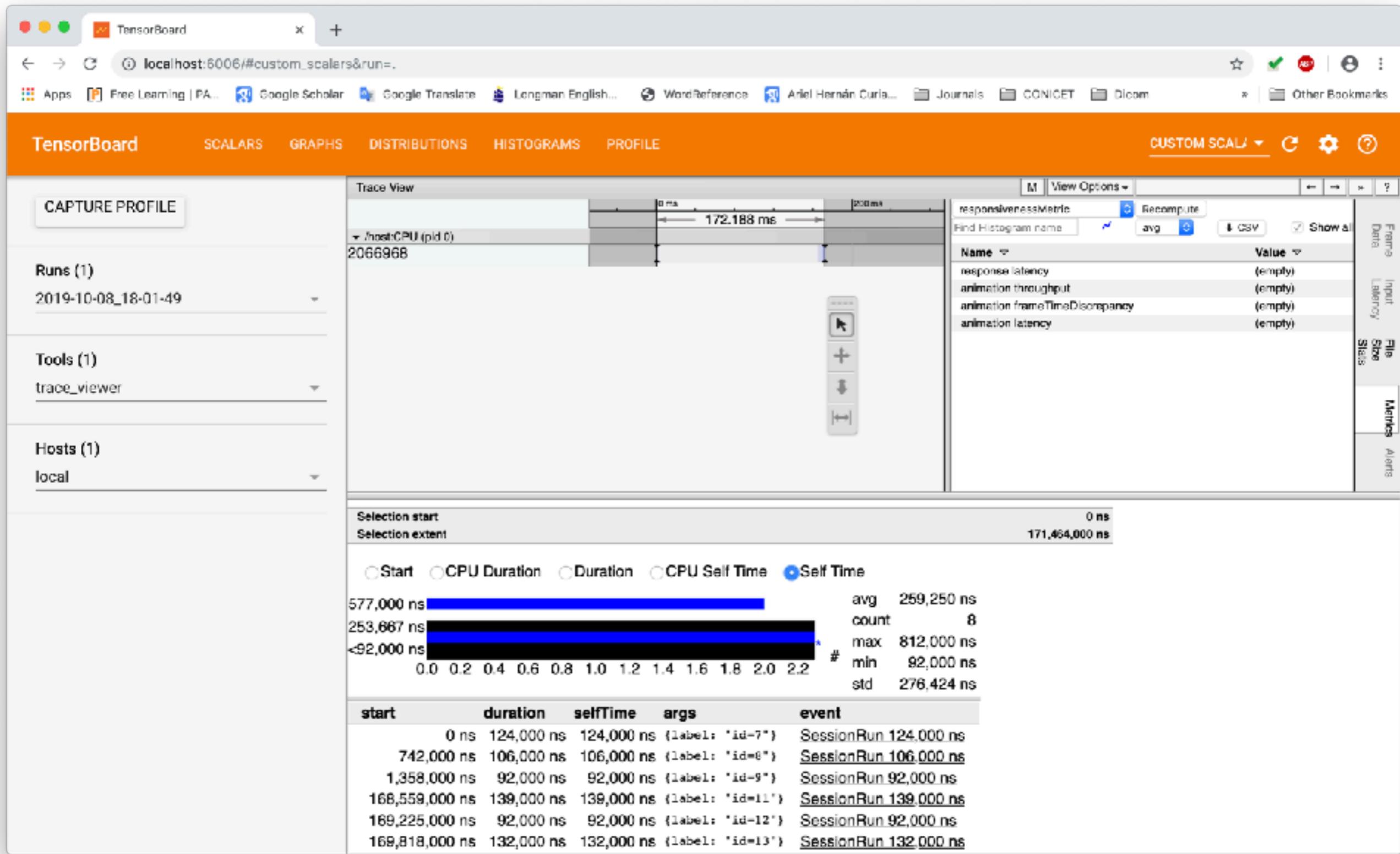
- ❖ Definir el path donde se va a almacenar la información de log de TensorBoard, si no existe se crea el directorio
- ❖ Usamos el callback de Keras

```
1 callbacks = [  
2     tf.keras.callbacks.TensorBoard(  
3         log_dir='keras_tensorboard',  
4         histogram_freq=1,)]  
5  
6 model.fit(x_data, y_data, epochs=200, validation_data=(x_data, y_data),  
7             batch_size=4, callbacks=callbacks, verbose=2)
```

- ❖ Iniciamos TensorBoard

```
ariel@Goku:~/Docencia/Balseiro/Materias/CV_DeepLearning/Teoria/09$ tensorboard --logdir=keras_tensorboard/  
TensorBoard 1.14.0 at http://localhost:6006/ (Press CTRL+C to quit)
```





Usando Chrome se puede obtener la información de profiling

Leer y guardar los modelos

- ❖ Todo el modelo

```
434 model.save('my_model.h5')
```

```
433 from tensorflow.keras.models import load_model  
434 model = load_model('my_model.h5')
```

- ❖ Solamente los pesos

```
1 model.save_weights('my_model_weights.h5')
```

```
4 model.load_weights('my_model_weights.h5')
```

Visualización de datos intermedios

Mapa de activación

```
2 import tensorflow as tf
3 from tensorflow.keras import preprocessing
4 from tensorflow.keras.applications.vgg16 import preprocess_input
5 from tensorflow.keras import backend as K
6
7 model = tf.keras.applications.VGG16(weights='imagenet', include_top=False)
8
9 img_path = 'fig/elephant.jpg'
10 img = preprocessing.image.load_img(img_path, target_size=(224, 224))
11
12 x = preprocessing.image.img_to_array(img)
13 x = np.expand_dims(x, axis=0)
14 x = preprocess_input(x)
15
16 layer_name = 'block4_conv1'
17 layer = model.get_layer(layer_name)
18 activation = layer.output
19
20 run = K.function([model.input], [activation])
21 I = run([x])[0]
```

Diagrama explicativo:

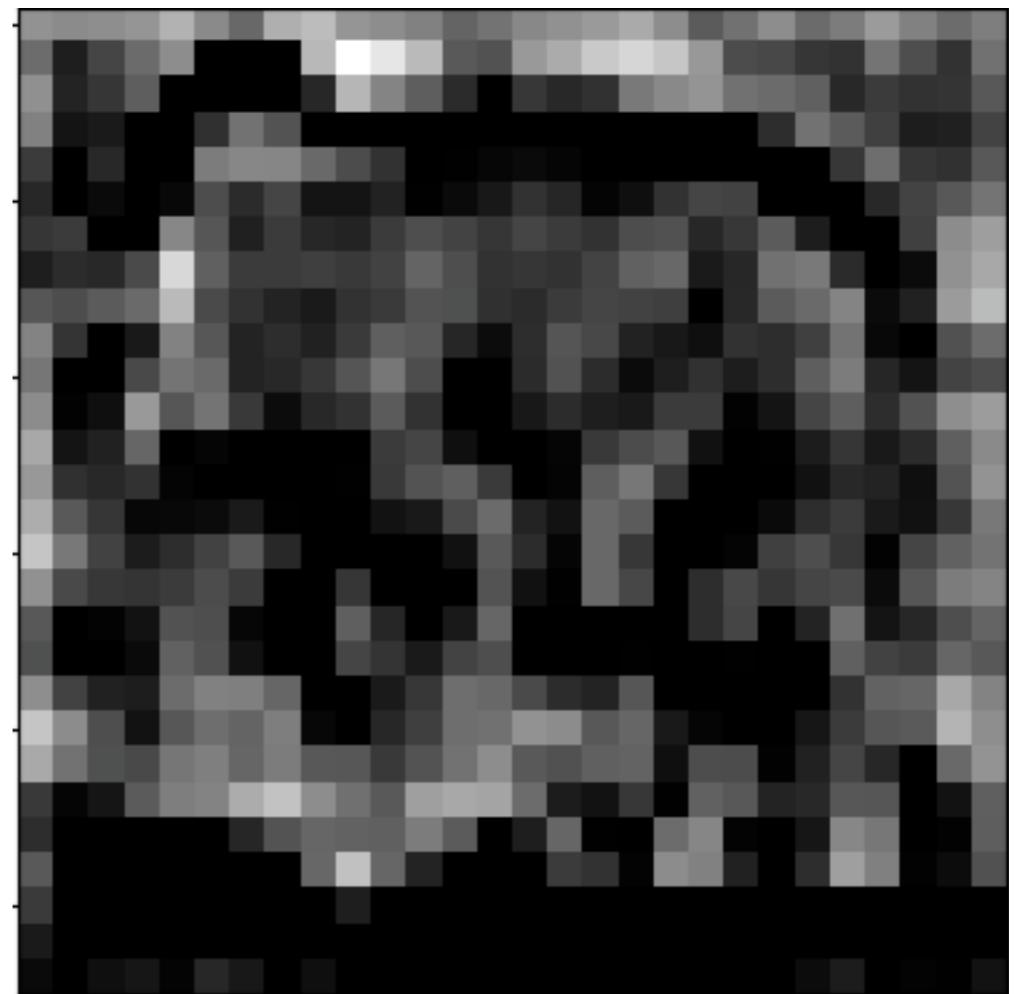
- Linea 16: `layer_name = 'block4_conv1'` → Layer
- Linea 17: `layer = model.get_layer(layer_name)` → Tensor
- Linea 20: `run = K.function([model.input], [activation])`
- Linea 21: `I = run([x])[0]`

Visualización de datos intermedios

Mapa de activación (layer block4conv1, filtro 10) dada una entrada



```
5 plt.figure()  
6 plt.imshow(np.squeeze(I[...,10]), 'gray')
```



Visualización de datos intermedios

¿No había una forma más simple de hacerlo?

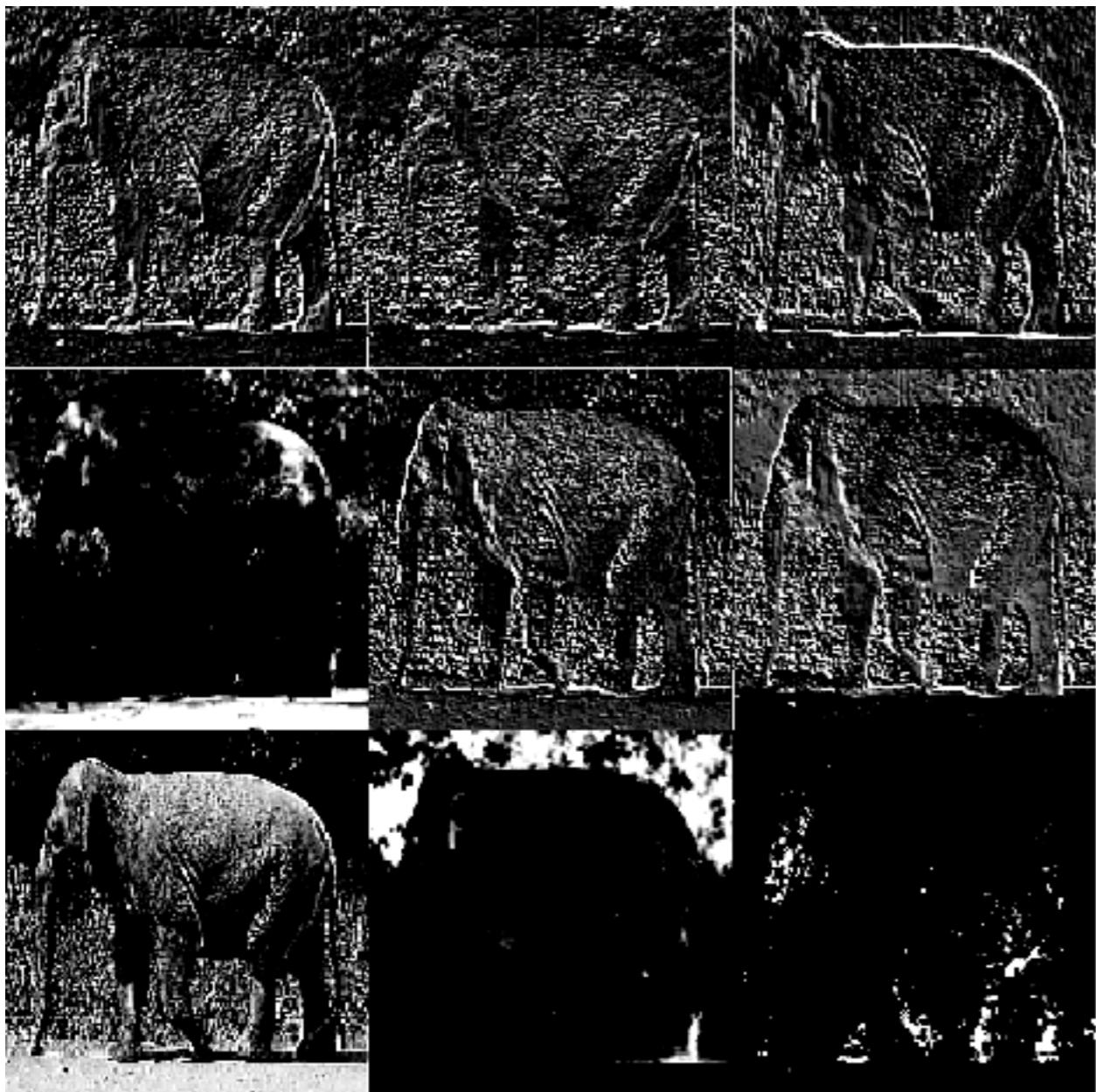
Si, pero es interesante K.function

```
478 layers = [l.output for l in model.layers[1:9] if 'conv' in l.name]
479 model_activation = tf.keras.models.Model(inputs=model.input, outputs=layers)
480 activations = model_activation.predict(x)
```

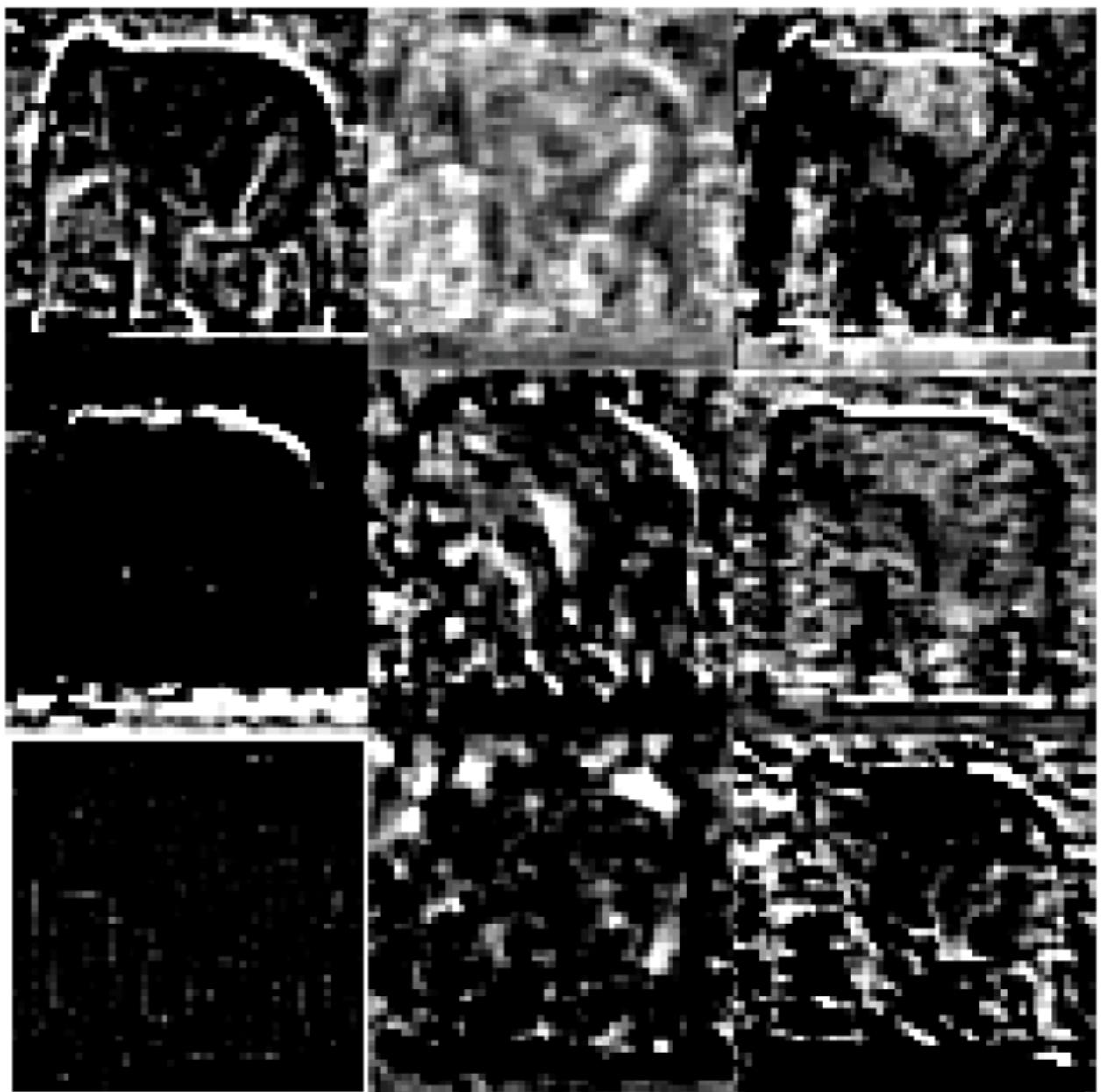
```
481 idl=4
482 nfigs=3
483 fig, ax = plt.subplots(nfigs,nfigs)
484 for i in range(nfigs):
485     for j in range(nfigs):
486         Ii = activations[idl][0,...,i+j*nfigs]
487         Ii -= Ii.mean()
488         Ii /= Ii.std()
489         Ii *= 64
490         Ii += 128
491         Ii = np.clip(Ii, 0, 255).astype('uint8')
492         ax[i,j].set_axis_off()
493         ax[i,j].imshow(Ii, 'gray')
494 fig.tight_layout()
```

Visualización de datos intermedios

layer 1 primeros 9 filtros



layer 4 primeros 9 filtros



Visualización de datos intermedios

¿Estamos viendo lo aprendido un filtro?

No, lo que prendió el filtro puede no ser ni de cerca lo que muestra su activación.

¿Qué significa lo que aprendió el filtro?

¿Qué estamos viendo entonces?

¿Cómo hacemos entonces para ver que aprendió la red?

Visualización de datos intermedios

Mapa de activación: Gradient ascendent input space

Armamos la imagen que maximiza la activación del filtro,
entonces, ese filtro aprendió a buscar esos detalles en la imagen

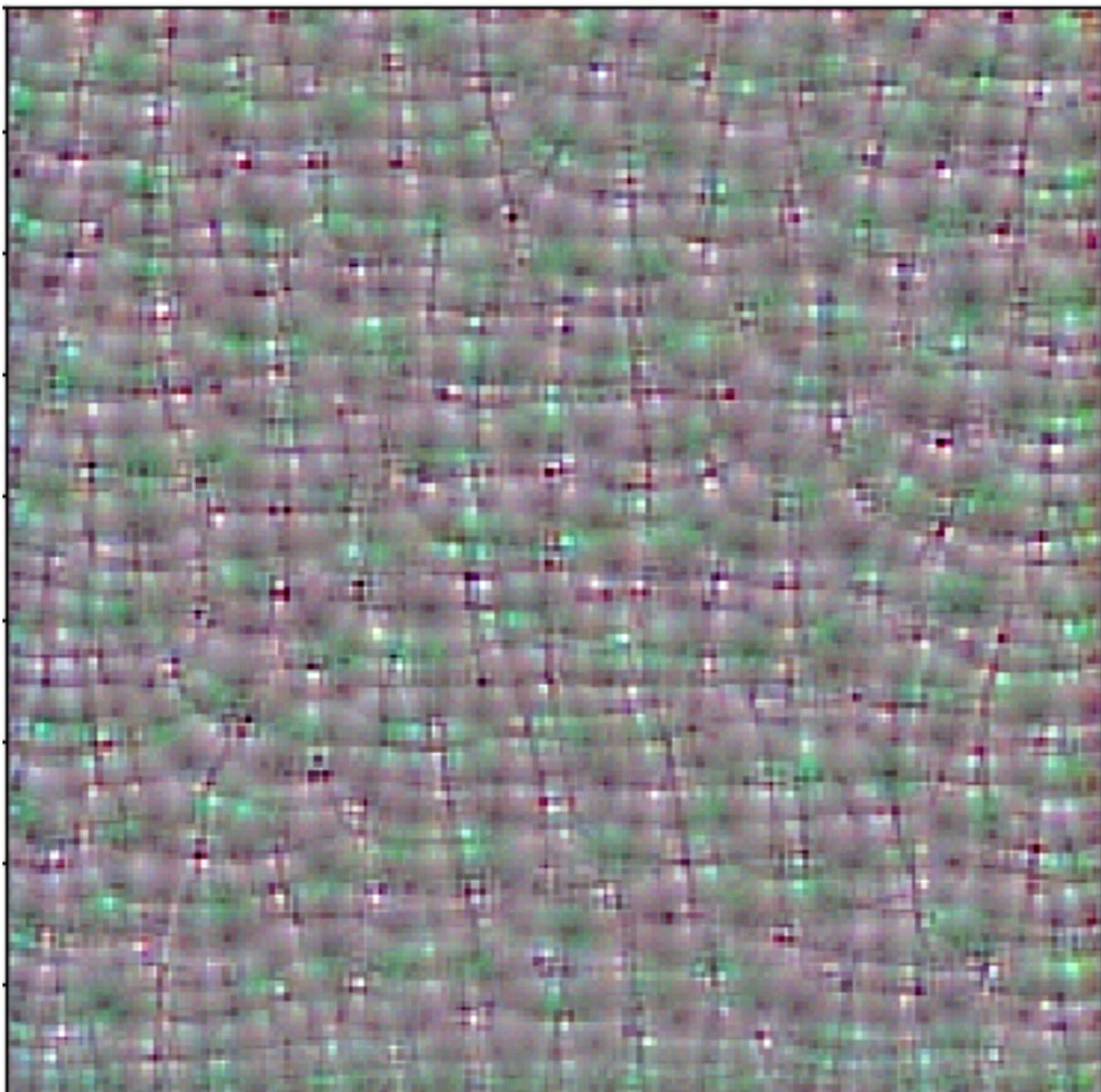
```
2 filt_index = 10
3 loss = K.mean(activation[..., filt_index])
4 grads = K.gradients(loss, model.input)[0]
5 # Truco de normalizacion
6 grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-6)
7 run = K.function([model.input], [loss, grads])
8
9 img_data = np.random.random((1,224,224,3)) * 20 + 128
10 step = 1
11 for i in range(40):
12     loss_val, grads_val = run([img_data])
13     img_data += grads_val * step
14     print('step:{:03d}\t loss:{:.3f}'.format(i, loss_val))
```

Arancamos con una imagen
sintética (gris con algo de ruido)

Maximizamos con
gradiente descendente

Visualización de datos intermedios

Mapa de activación: Gradient ascent input space

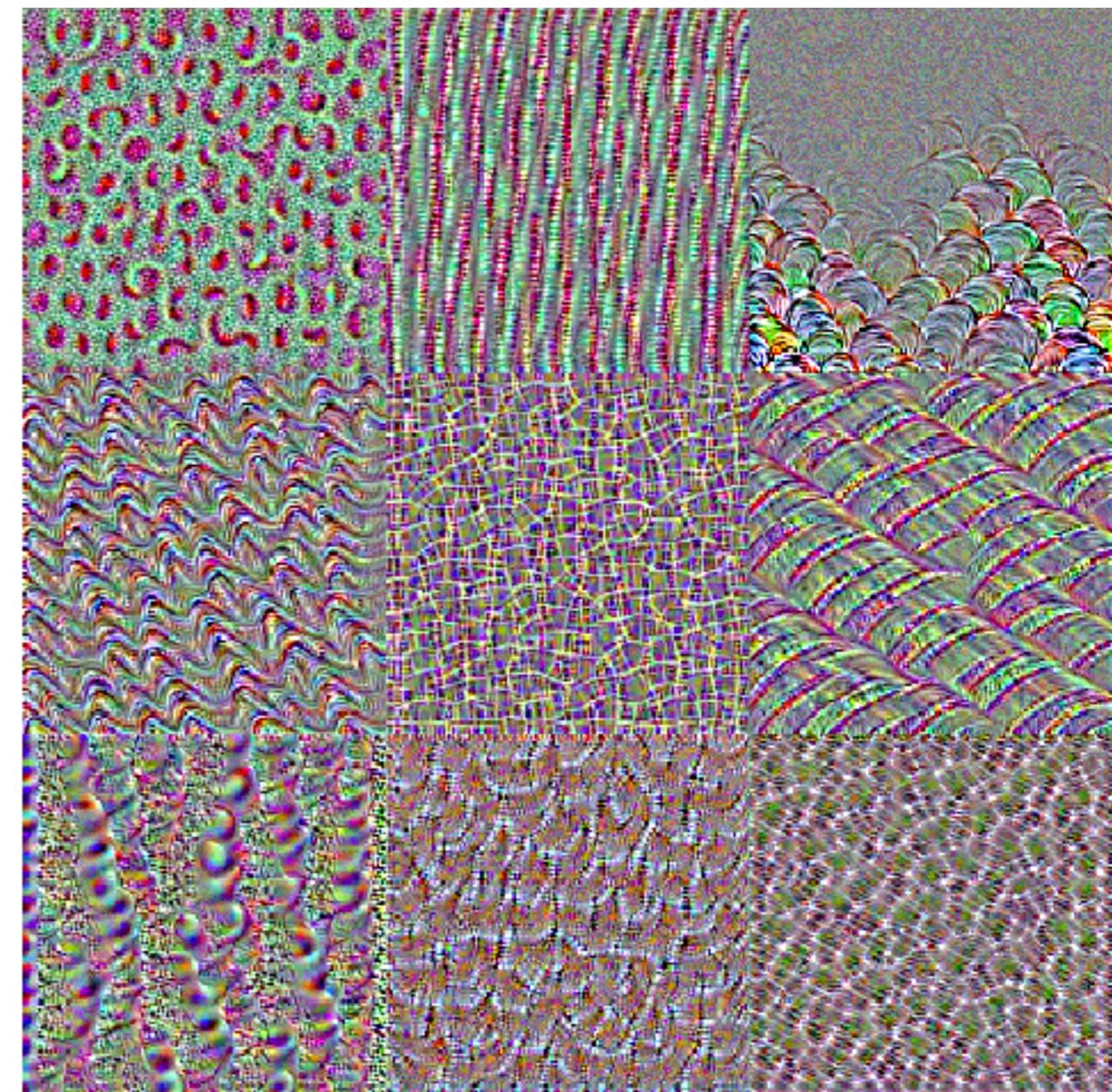


```
2 def deprocess_img(x):
3     x -= x.mean()
4     x /= (x.std() +1e-7)
5     x *= .1
6     x += .5
7     x = np.clip(x, 0, 1)
8     x *= 255
9     x = np.clip(x, 0, 255).astype('uint8')
10    return x
12 I2 = deprocess_img(img_data[0])
13 plt.figure()
14 plt.imshow(I2)
```

Visualización de datos intermedios

Mapa de activación: Gradient ascendent input space

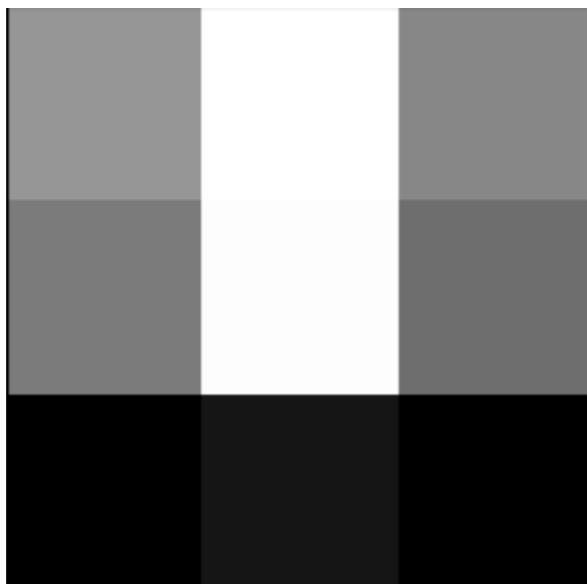
```
1 nfigs=3
2 fig, ax = plt.subplots(nfigs,nfigs)
3 for i in range(nfigs):
4     for j in range(nfigs):
5         img_data = learned_image(layer_name,
6                               i+j*nfigs)
7         Ii = img_data[0]
8         Ii -= Ii.mean()
9         Ii /= Ii.std()
10        Ii *= 64
11        Ii += 128
12        Ii = np.clip(Ii, 0, 255).astype('uint8')
13        ax[i,j].set_axis_off()
14        ax[i,j].imshow(Ii, 'gray')
15 fig.tight_layout()
```



Visualización de datos intermedios

¿Qué pinta tienen los filtros?

```
506 lname1 = 'block1_conv1'  
507 l1_weights = model.get_layer(lname1).get_weights()  
  
3 channel_id = 0  
4 filt_id = 10  
  
6 plt.figure()  
7 plt.imshow(l1_weights[0][...,channel_id,filt_id], 'gray')
```



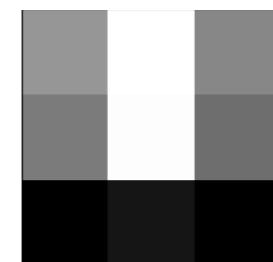
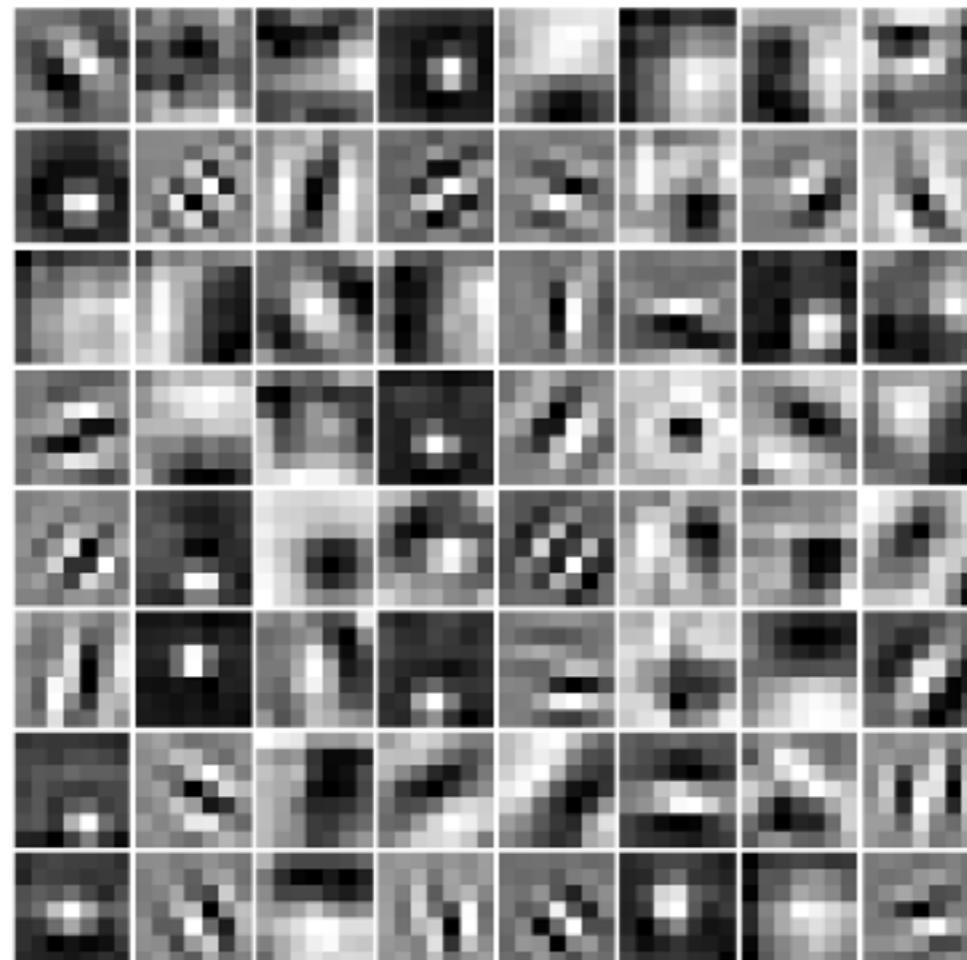
3x3 no nos aporta mucho

Visualización de datos intermedios

¿Qué pinta tienen los filtros?

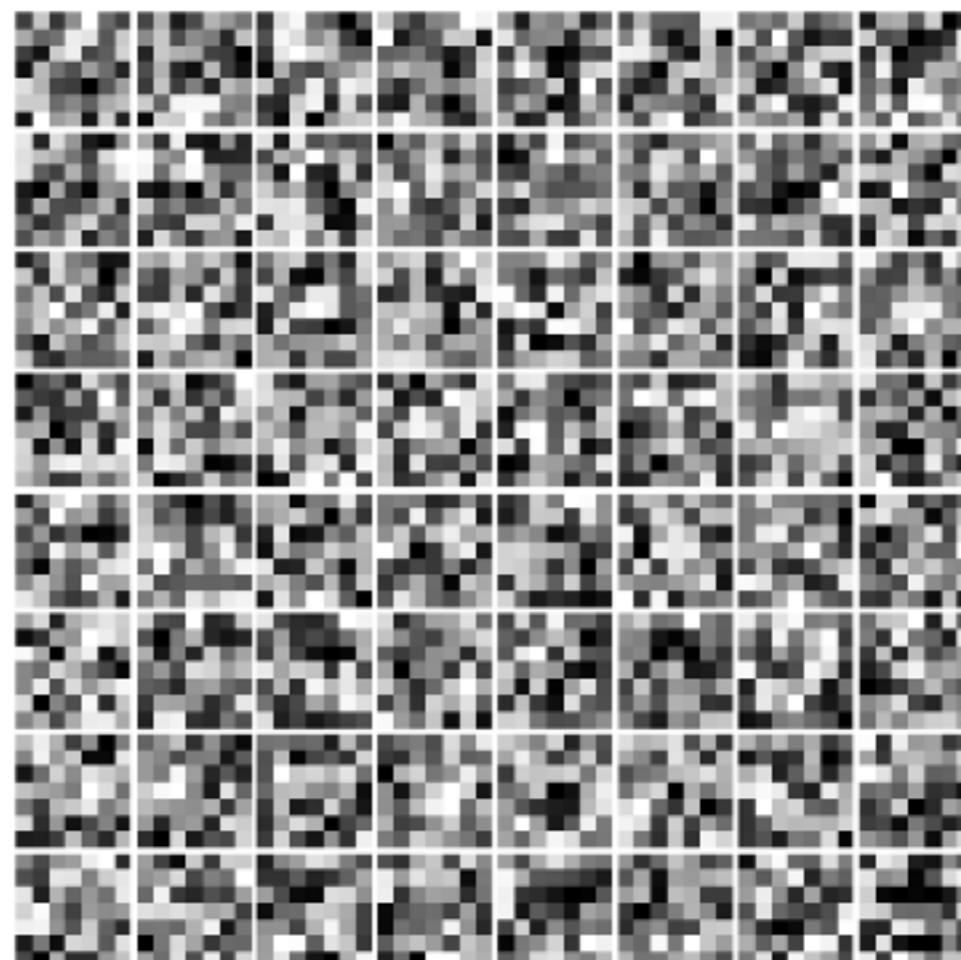
```
506 lname1 = 'block1_conv1'  
507 l1_weights = model.get_layer(lname1).get_weights()  
3 channel_id = 0  
4 filt_id = 10  
  
6 plt.figure()  
7 plt.imshow(l1_weights[0][...,channel_id,filt_id], 'gray')
```

ResNet50
Conv 1



3x3 no nos
aporta mucho

Unet
Conv 1



¿Alguno obtuvo dos veces el mismo resultado al utilizar TF o Keras?

Reproducibilidad de los datos

- ❖ Ejecutar varias veces y usar estadísticas al momento de describir el rendimiento de nuestros modelo
- ❖ Fijar la semilla de los generadores aleatorios.
 - ❖ En GPU hay instrucciones que NO son deterministas
 - ❖ En CPU se puede lograr pero hay que fijar la variable de entorno PYTHONHASHSEED a 0 antes de ejecutar el código. Por ejemplo así:

```
CUDA_VISIBLE_DEVICES="" PYTHONHASHSEED=0 python your_program.py
```

Reproducibilidad de los datos

- ❖ Las semillas de los generadores de números aleatorios se tienen que definir antes que otra cualquier cosa

Reproducibilidad de los datos

```
1 #----- Código para obtener reproducibilidad
2 import numpy as np
3 import tensorflow as tf
4 import random as rn
5 # The below is necessary for starting Numpy generated random numbers
6 # in a well-defined initial state.
7 np.random.seed(42)
8 # The below is necessary for starting core Python generated random numbers
9 # in a well-defined state.
10 rn.seed(12345)
11 # Force TensorFlow to use single thread.
12 # Multiple threads are a potential source of non-reproducible results.
13 # For further details, see: https://stackoverflow.com/questions/42022950/
14 session_conf = tf.ConfigProto(intra_op_parallelism_threads=1,
15                               inter_op_parallelism_threads=1)
16 from tensorflow.keras import backend as K
17 # The below tf.set_random_seed() will make random number generation
18 # in the TensorFlow backend have a well-defined initial state.
19 # For further details, see:
20 # https://www.tensorflow.org/api\_docs/python/tf/set\_random\_seed
21 tf.set_random_seed(1234)
22 sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
23 K.set_session(sess)
```