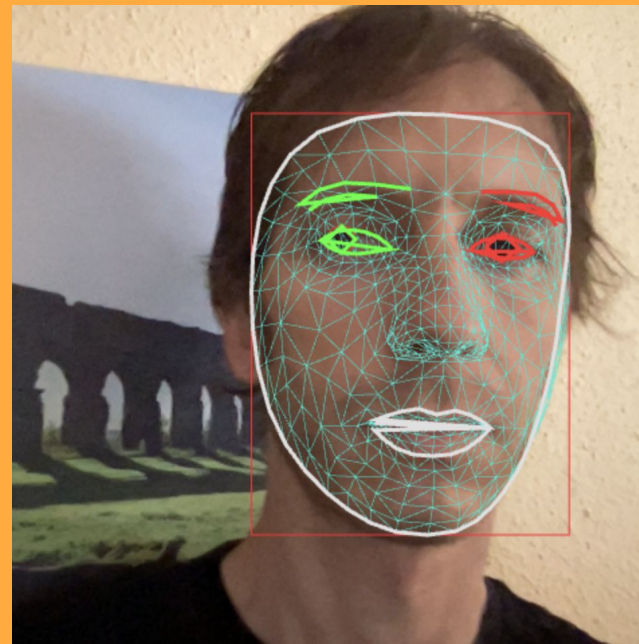


Inteligencia Ambiental

Máster Universitario en Inteligencia Artificial Aplicada

Modelos pre-entrenados y tensores
en TFJS



Contenido

1. Utilizar modelos pre-entrenados brutos (o sin procesar)
2. Detección de articulaciones del cuerpo con Movenet

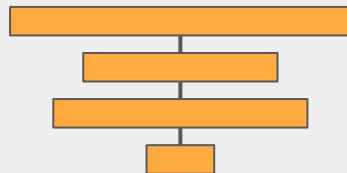
1. Utilizar modelos pre-entrenados brutos (o sin procesar)

Modelos brutos

- Necesario escribir el código JavaScript: se usan tensores para enviar datos al modelo y extraer los datos
- Útiles para encontrar y usar ejemplos en desarrollo o que resuelven un caso menos común
- Dos tipos de modelos brutos (o **sin procesar**)
 - **Modelo de capas (Layers Model):**
 - **Modelo de grafo (Graph Model):**
- Los modelos de capas y los modelos de grafo son dos formatos distintos: necesitamos dos funciones distintas para cargarlos
- Ejemplos en <https://github.com/tensorflow/tfjs-examples>

[tf.LayersModel](#)

Modelo de capas (Layers Model)



- Más fáciles de inspeccionar y entender en tiempo de ejecución
- Pueden funcionar más lentamente: no se han optimizado

[tf.GraphModel](#)

Modelo de grafo (Graph Model)



- No son fáciles de entender
- Altamente optimizados

Modelo de capas (Layers Model)

- Se crea:
 - [Convirtiendo](#) un modelo Keras (Python) utilizando el [conversor de línea de comando](#)
 - A través del API [tf.model](#) o [tf.sequential](#)
- Se guarda como:
 - Un archivo **model.json**: metadatos sobre tipo, arquitectura y configuración del modelo
 - Archivo(s) **.bin**: almacenan los pesos entrenados que ha aprendido el modelo
 - Divido en fragmentos: **shard1ofN.bin ... shardNofN.bin**
 - Cada fragmento ~ 4 MB: descarga simultánea para acelerar el tiempo de carga de la página
- Archivos alojados en servidor web o en una red de distribución de contenidos (CDN)

MODELS

Creation

tf.sequential

tf.model

API TFJS para modelos de capas

- Cargar modelos de capas
- Entender el modelo
- Realizar inferencias/predicciones

Ejemplo: utilizamos el modelo [iris v1](#) para clasificar la flor Iris de tres especies relacionadas (I. setosa, I. versicolor y I. virginica). [El conjunto de datos Iris](#) contiene 50 muestras de cada una de las tres especies y la información de cuatro rasgos de cada muestra: largo y ancho del sépalo y largo y ancho del pétalo, en centímetros

Código de ejemplo:

<https://aulaglobal.uc3m.es/mod/resource/view.php?id=5253079>

MODELSLoading

tf.loadGraphModel

tf.loadLayersModel

tf.io.browserDownloads

tf.io.browserFiles

tf.io.http

tf.loadGraphModelSync

Classes

tf.LayersModel

.summary

.compile

.evaluate

.evaluateDataset

.predict

.predictOnBatch

.fit

.fitDataset

.trainOnBatch

.save

.getLayer

Cargar el modelo

```
//1. Load the model
const model = await tf.loadLayersModel(
  "https://storage.googleapis.com/tfjs-models/tfjs/iris_v1/model.json"
);
```

tf.loadLayerModel es un método asíncrono

Acepta:

- Una ruta al fichero **model.json** alojado en un servidor web
- Otras fuentes como local storage, indexDB o fichero local

MODELSLoading`tf.loadGraphModel``tf.loadLayersModel``tf.io.browserDownloads``tf.io.browserFiles``tf.io.http``tf.loadGraphModelSync`Classes`tf.LayersModel``.summary``.compile``.evaluate``.evaluateDataset``.predict``.predictOnBatch``.fit``.fitDataset``.trainOnBatch``.save``.getLayer`

Entender el modelo

```
//2. Understand the model  
model.summary();
```

`tf.LayerModel.summary` muestra en la consola información sobre: (1) capas del modelo, (2) formas de salida en cada capa, (3) número de parámetros en cada capa, (4) número total de parámetros, (5) número de parámetros entrenables y no entrenables



Layer (type)	Input Shape	Output shape	Param #
input_1 (InputLayer)	[[null,4]]	[null,4]	0
Dense1 (Dense)	[[null,4]]	[null,10]	50
Dense2 (Dense)	[[null,10]]	[null,3]	33
Total params: 83			
Trainable params: 83			
Non-trainable params: 0			

La primera es una **capa de entrada** con **forma de salida (null, 4)** y **0 parámetros entrenables**. Las otras dos son capas completamente conectadas (**densas**). La última capa es la **capa de salida**, la **forma de salida (null, 3)** significa que el modelo tendrá tres posibles valores de salida, las tres clases de Iris

Layer (type)	Input Shape	Output shape	Param #
input_1 (InputLayer)	[[null,4]]	[null,4]	0
Dense1 (Dense)	[[null,4]]	[null,10]	50
Dense2 (Dense)	[[null,10]]	[null,3]	33
Total params: 83			
Trainable params: 83			
Non-trainable params: 0			

El parámetro **null** es el **tamaño de lote (batch)**: permite clasificar n ejemplos de entrada de una vez (en un lote). Pasar entradas como lotes requiere añadir otra dimensión al tensor de entrada. Un **tensor de entrada 1D (los valores de los cuatro rasgos)** debe almacenarse como un **tensor 2D** para usarse en un lote.

Layer (type)	Input Shape	Output shape	Param #
input_1 (InputLayer)	[[null,4]]	[null,4]	0
Dense1 (Dense)	[[null,4]]	[null,10]	50
Dense2 (Dense)	[[null,10]]	[null,3]	33
Total params: 83			
Trainable params: 83			
Non-trainable params: 0			

La **forma de salida (null, 3)** de la **capa de salida** significa que el modelo tendrá tres posibles valores de salida, **las tres clases de Iris**.

Layer (type)	Input Shape	Output shape	Param #
input_1 (InputLayer)	[[null,4]]	[null,4]	0
Dense1 (Dense)	[[null,4]]	[null,10]	50
Dense2 (Dense)	[[null,10]]	[null,3]	33
Total params: 83			
Trainable params: 83			
Non-trainable params: 0			

El **número total de parámetros** da una idea de la complejidad del modelo. Más parámetros generalmente significan que se necesitará más memoria y más potencia de procesamiento.

Layer (type)	Input Shape	Output shape	Param #
input_1 (InputLayer)	[[null,4]]	[null,4]	0
Dense1 (Dense)	[[null,4]]	[null,10]	50
Dense2 (Dense)	[[null,10]]	[null,3]	33
=====			
Total params: 83			
Trainable params: 83			
Non-trainable params: 0			

MODELSLoading

tf.loadGraphModel
 tf.loadLayersModel
 tf.io.browserDownloads
 tf.io.browserFiles
 tf.io.http
 tf.loadGraphModelSync

Classes

tf.LayersModel
 .summary
 .compile
 .evaluate
 .evaluateDataset
 .**predict**
 .predictOnBatch
 .fit
 .fitDataset
 .trainOnBatch
 .save
 .getLayer

Realizar inferencias/predicciones

```
//3. Make inferences or predictions
tf.tidy(() => {
  const inputValues = [5.0, 3.3, 1.4, 0.2];
  const inputTensor = tf.tensor2d([inputValues], [1, 4]);
  const prediction = model.predict(inputTensor);
  const predictionValues = prediction.dataSync();
  const argMaxIdx = prediction.argmax(-1).dataSync()[0];
  console.log(
    `Input: ${inputValues} - ${IRIS_CLASSES[argMaxIdx]} with
  );
  outputMessageEl.innerText = `Input: ${inputValues} - ${IRI
});
```

tf.LayerModel.**predict** genera predicciones de salida para las entradas: acepta un tf.**tensor2d**

En este ejemplo el segundo parámetro de tf.**tensor2d**, la forma del tensor, es [1,4]: 1 ejemplo que tiene 4 valores (los rasgos del iris)

tf.Tensor.**dataSync** convierte el tensor a un array con los valores

tf.Tensor.**argMax** devuelve los índices de los valores máximos de un eje

MODELSLoading

tf.loadGraphModel
tf.loadLayersModel
tf.io.browserDownloads
tf.io.browserFiles
tf.io.http
tf.loadGraphModelSync

Classes


tf.LayersModel
.summary
.compile
.evaluate
.evaluateDataset
.predict
.predictOnBatch
.fit
.fitDataset
.trainOnBatch
.save
.getLayer

Realizar inferencias/predicciones

```
//3. Make inferences or predictions  
tf.tidy(() => {  
  const inputValues = [5.0, 3.3, 1.4];  
  const inputTensor = tf.tensor2d([inputValues], [1, 4]);
```

Atención, si hubiera una discrepancia entre las formas esperadas y la forma proporcionada, TFJS lanzará un "error de forma" en la consola

Es un error **muy común** cuando se usan tensores



```
✖ ▶ Uncaught (in promise) Error: Based on the provided shape, [1,4], the tensor should have 4 values but has 3  
    at jv (util_base.js:153:11)  
    at Dk (tensor_ops_util.js:48:5)  
    at Object.oR [as tensor2d] (tensor2d.js:65:10)  
    at index.js:19:22  
    at engine.js:467:20  
    at t.scopedRun (engine.js:478:19)  
    at t.tidy (engine.js:465:17)  
    at Object.hN [as tidy] (globals.js:192:17)  
    at app (index.js:18:6)
```

Modelo de grafo (Graph Model)

- Se crea:
 - [Convirtiendo](#) un modelo TensorFlow SavedModel
- Se guarda como:
 - Un archivo **model.json**: metadatos sobre tipo, arquitectura y configuración del modelo
 - Archivo(s) **.bin**: almacenan los pesos entrenados que ha aprendido el modelo
 - Divido en fragmentos: **shard1ofN.bin ... shardNofN.bin**
 - Cada fragmento ~ 4 MB: descarga simultánea para acelerar el tiempo de carga de la página
- Archivos alojados en servidor web o en una red de distribución de contenidos (CDN)

API TFJS para modelos de grafo

- Cargar modelos de grafo
- Realizar inferencias/predicciones

Ejemplo: cargar y hacer una predicción ficticia con el modelo movenet/singlepose/lighting de TensorFlowHub

Código de ejemplo:

<https://aulaglobal.uc3m.es/mod/resource/view.php?id=4765392>

Trabajaremos con el mismo modelo de grafo en el tutorial para detectar poses con movenet

MODELSLoading**tf.loadGraphModel**

tf.loadLayersModel

tf.io.browserDownloads

tf.io.browserFiles

tf.io.http

tf.loadGraphModelSync

Classes

tf.GraphModel

.loadSync

.save

.predict

.execute

.executeAsync

.getIntermediateTensors

.disposeIntermediateTensors

.dispose

Cargar el modelo



```
const MODEL_PATH = "https://tfhub.dev/google/tfjs-model/movenet/singlepose/lightning/4";

async function app() {
  const model = await tf.loadGraphModel(MODEL_PATH, { fromTFHub: true });
}
```

tf.loadGraphModel es un método asíncrono

Acepta:

- Una ruta al fichero **model.json** alojado en un servidor web
- Otras fuentes como local storage, indexDB o fichero local
- Con la opción **fromTFHub:true**, permite pasar una URL de módulo TF-Hub, omitiendo el nombre de archivo del modelo estándar y los parámetros de consulta

MODELSLoading

tf.loadGraphModel
tf.loadLayersModel
tf.io.browserDownloads
tf.io.browserFiles
tf.io.http
tf.loadGraphModelSync

Classes

tf.GraphModel
 .loadSync
 .save
 .predict
 .execute
 .executeAsync
 .getIntermediateTensors
 .disposeIntermediateTensors
 .dispose

Realizar inferencias/predicciones

```
async function app() {  
  const model = await tf.loadGraphModel(MODEL_PATH, { fromTFHub: true });  
  tf.tidy(() => {  
    const exampleInputTensor = tf.zeros([1, 192, 192, 3], "int32");  
    const tensorOutput = model.predict(exampleInputTensor);  
    const arrayOutput = tensorOutput.arraySync();  
    console.log(arrayOutput);  
  });  
}
```

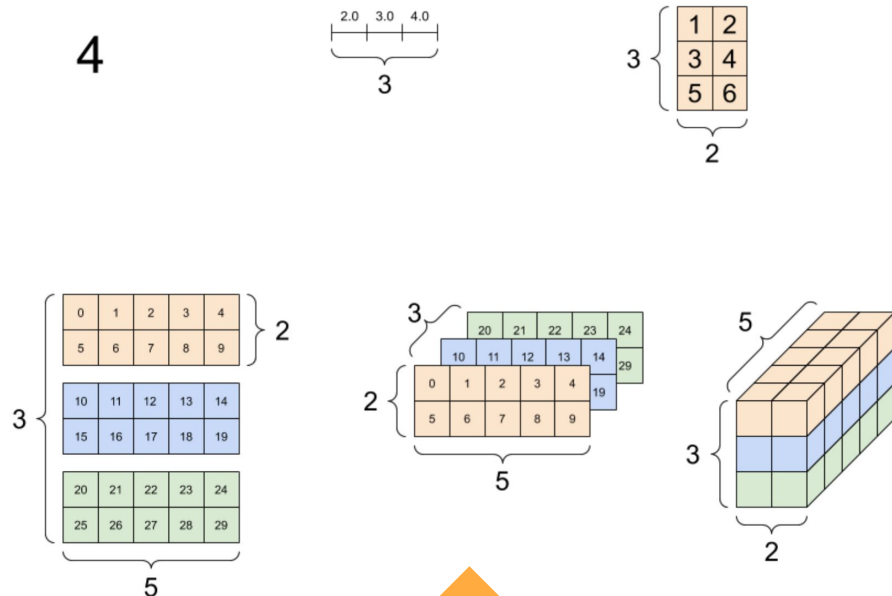
tf.GraphModel.**predict** funciona exactamente como su omologo tf.LayerModel.predict

tf.**zeros** genera un tensor de la forma especificada

tf.Tensor.**arraySync** convierte el tensor en un array anidado

Tensoros

- **Estructuras de datos primarias** en los programas de TensorFlow, similares a los arrays
- Pueden tener **múltiples dimensiones**
- Contienen **datos uniformes** (a diferencia de los arrays en JavaScript)
- Los modelos los usan como **entradas**, los **procesan** y los devuelven como **salidas**



Podemos visualizar un tensor 3d de distintas maneras...

API TFJS /1

https://www.tensorflow.org/js/guide/tensors_operations

- Un tensor es un objeto de la clase **tf.Tensor**
- TFJS proporciona funciones para:
 1. **Creación** de tensores: `tf.scalar`, `tf.tensor1d`, `tf.tensor2d`, ...
 2. **Obtener valores**: `tf.Tensor.data`, `tf.Tensor.array`
 3. **Operaciones** aritméticas y matemáticas básicas
 4. **Transformaciones**
 5. **Gestionar la memoria**
...
- Podemos utilizar el API de dos maneras:
 - Desde el objeto **tf** -> `tf.mul(a,b)`
 - Desde el objeto **tf.Tensor** -> `a.mul(b)`

Ejemplos: <https://aulaglobal.uc3m.es/mod/resource/view.php?id=4767948>

1 Creation

`tf.tensor`
`tf.scalar`
`tf.tensor1d`
`tf.tensor2d`
`tf.tensor3d`
`tf.tensor4d`
`tf.tensor5d`
`tf.tensor6d`
`tf.buffer`
`tf.clone`

Classes

tf.Tensor

`.buffer`
`.bufferSync`
`.array`
2 `.arraySync`
`.data`
`.dataToGPU`
`.dataSync`
`.dispose`
`.print`
`.clone`
`.toString`

3 Arithmetic

`tf.add`
`tf.sub`
`tf.mul`
`tf.div`
`tf.addN`
`tf.divNoNan`
`tf.floorDiv`
`tf.maximum`
`tf.minimum`
`tf.mod`
`tf.pow`

4 Transformations

`tf.batchToSpaceND`
`tf.broadcastArgs`
`tf.broadcastTo`
`tf.cast`
`tf.depthToSpace`
`tf.expandDims`
`tf.mirrorPad`
`tf.pad`
`tf.reshape`

5 Memory

`tf.tidy`
`tf.dispose`
`tf.keep`
`tf.memory`

API TFJS /2

- Podemos imprimir un tensor utilizando el método **tf.Tensor.print(verbose?)**
- Si el argumento **verbose** es **true**, se imprimirá también información adicional como el tipo de dato, el rango o la forma

```
Tensor
dtype: float32
rank: 3
shape: [192,192,3]
values:
[[[197      , 229      , 230      ],
  [196.6875 , 228      , 229.6875 ],
  [194.75    , 227.375  , 228.375  ],
  ...,
  [151      , 189.875   , 190.9375 ],
  [150.625   , 188      , 189.625   ],
  [150      , 188      , 189      ]],
 [[195.375   , 228      , 229      ],
  [195.375   , 228      , 228.9023438],
  [194.3203125, 227.1796875, 228.2578125],
  ...,
  [153.5429688, 190.2304688, 192.2304688],
  [152.4296875, 189.2578125, 191      ],
  [152.625    , 189.4296875, 190.7421875]],
```

API TFJS /3

- TFJS proporciona también funciones para obtener tensores desde:
 1. Fotogramas de la **cámara web**
 1. Audio del **micrófono**
 1. **Ficheros** csv
 2. Imágenes **estáticas** en el navegador

1 DATA

Creation

tf.data.array

tf.data.csv

tf.data.generator

tf.data.microphone

tf.data.webcam

2

BROWSER

tf.browser.fromPixels

tf.browser.fromPixelsAsync

tf.browser.toPixels

API TFJS /4

- Tenemos también funciones para manipular y realizar operaciones con imágenes:
 - Recortar y cambiar de tamaño
 - Dar la vuelta
 - Convertir de escala de gris a RGB
 - ...

Images

tf.image.cropAndResize
tf.image.flipLeftRight
tf.image.grayscaleToRGB
tf.image.nonMaxSuppression
tf.image.nonMaxSuppressionAsync
tf.image.nonMaxSuppressionPadded
tf.image.nonMaxSuppressionPaddedAsync
tf.image.nonMaxSuppressionWithScore
tf.image.nonMaxSuppressionWithScoreAsync
tf.image.resizeBilinear
tf.image.resizeNearestNeighbor
tf.image.rotateWithOffset
tf.image.transform

Importante!

- TensorFlow no elimina los tensores inutilizados automáticamente
- Es necesario limpiar la memoria de los tensores intermedios para evitar **memory leaks**
- `tf.Tensor.dispose` elimina el tensor
- `tf.tidy(fn)` ejecuta la función proporcionada `fn` y, una vez ejecutada, **limpia todos los tensores intermedios** declarados en `fn`, excepto los tensores devueltos por `fn`

```
const webcam = await tf.data.webcam(webcamEl, {
  centerCrop: true,
  resizeWidth: 192,
  resizeHeight: 192
});

while (true) {
  const img = await webcam.capture();
  img.dispose();
  await tf.nextFrame();
}
```

No ejecutar este código sin la llamada a `img.dispose()`, se generaría una pérdida de memoria

Términos

- **Tipo de datos** (dtype, datatype) se refiere al tipo de datos que almacenará el tensor. Pueden ser **float32**, **int32**, **bool**, **complex64**, **string**
- **Rango/Eje** (rank/axis): la dimensionalidad de un tensor se define con su **rango**. El número de **ejes** en un tensor también corresponde al número de dimensiones o rango del tensor. El rango máximo de un tensor en TFJS es 6
- **Forma** (shape): la forma de un tensor define el número de elementos presentes a lo largo de cada eje del tensor. Es la representación de la longitud de un tensor en cada dimensión
- **Tamaño** (size) es el número de elementos de un tensor

Rango 0

Escalar, un valor numérico

Ejemplo:

1



Uso: Cualquier caso en el que se necesite un único valor numérico

Rango 1

Un array de números

Ejemplo:

[4, 9, 2]



Uso: Representa las coordenadas x, y, z de un punto

Rango 2

Una matriz (array de arrays)

Ejemplo:

```
[  
  [4, 9, 2],  
  [3, 5, 7],  
  [8, 1, 6]  
]
```

4	9	2
3	5	7
8	1	6

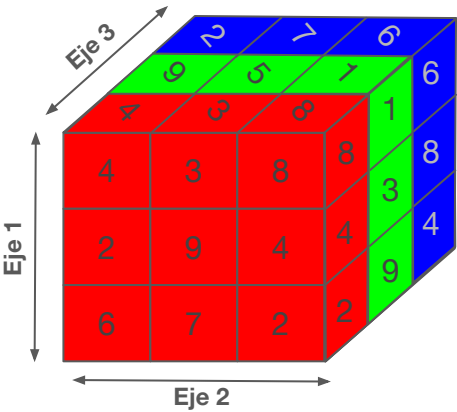
Uso: Representar una imagen en escala de gris, cada píxel es un valor de 0 a 255

Rango 3

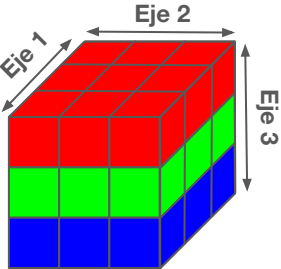
Una pila de matrices

Ejemplo:

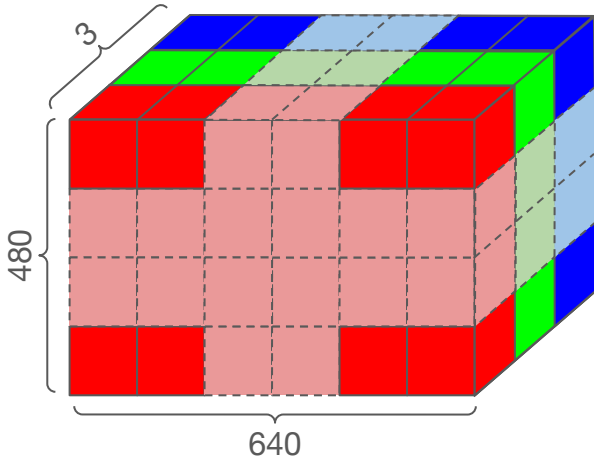
```
[  
  [  
    [4, 9, 2], [3, 5, 7], [8, 1, 6]  
  ],  
  [  
    [2, 7, 6], [9, 5, 1], [4, 3, 8]  
  ],  
  [  
    [6, 1, 8], [7, 5, 3], [2, 9, 4]  
  ],  
]
```



Podemos visualizar un tensor de distintas maneras...



Uso: Representar una imagen RGB



```

```

```
const imgEl = document.querySelector("#img");
const imageTensor = tf.browser.fromPixels(imgEl);
imageTensor.print(true);
```

```
Tensor
dtype: int32
rank: 3
shape: [480,640,3]
values:
[[[71 , 144, 225],
  [72 , 145, 226],
  [72 , 145, 226],
  ...,
  [66 , 137, 217],
  [65 , 136, 216],
  [66 , 137, 217]]],
```

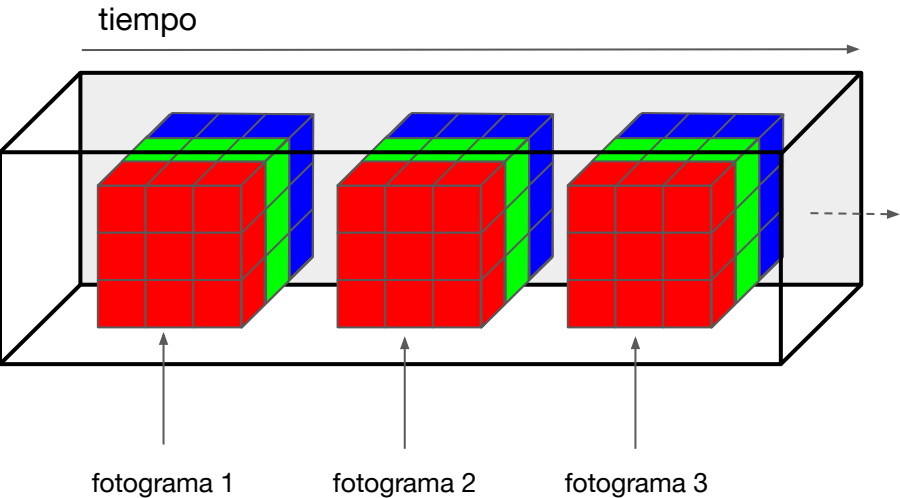
[R, G, B] del primer píxel
de la imagen

Rango 4

Una pila o cola de tensores de rango 3

Se pueden usar, por ejemplo, para representar un video: una secuencia (**tiempo**) de fotogramas como **imágenes RGB** (3 ejes)

O también un **lote de imágenes**

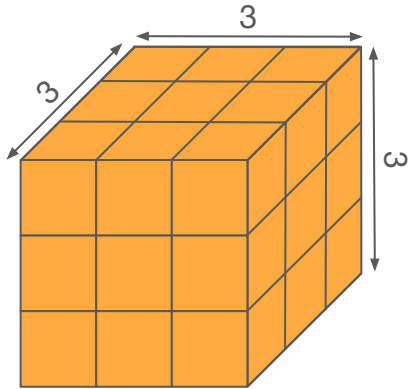


Rango 5

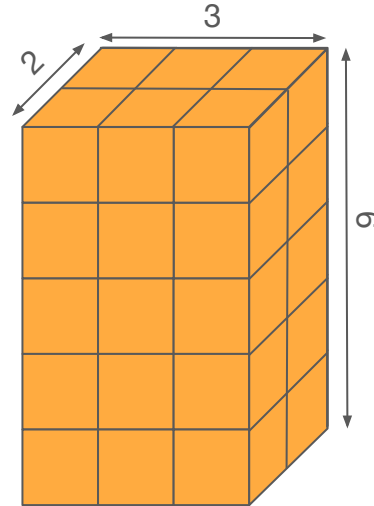
En un juego 3d como Minecraft, se puede utilizar 1 eje para representar el **color** de un voxel (pixel 3d), otros 3 ejes para las coordenadas **x, y, z** del voxel, y un último eje para el **tiempo**



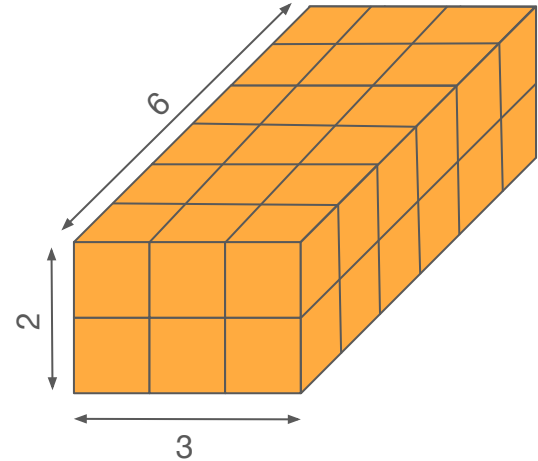
Forma y tamaño



Forma: [3, 3, 3]
Tamaño: 27



Forma: [2, 3, 6]
Tamaño: 36



2. Detección de articulaciones del cuerpo con Movenet

movenet/singlepose/lightning

<https://storage.googleapis.com/movenet/MoveNet.SinglePose%20Model%20Card.pdf>

<https://tfhub.dev/google/tfjs-model/movenet/singlepose/lightning/4>

- Modelo de red neuronal convolucional (CNN) que se ejecuta en imágenes RGB y predice la ubicación de las articulaciones humanas de **una sola persona**
- Puede **ejecutarse >50FPS** en la mayoría de los portátiles modernos
- Más adecuado para detectar la pose de una sola persona que está a una **distancia de entre 1 y 2 metros**
- Detecta la pose de la persona que está **más cerca del centro** de la imagen
- Predice **17 puntos clave** del cuerpo incluso cuando están ocluidos

Tutorial 3: Detección de articulaciones del cuerpo con Movenet

<https://docs.google.com/document/d/10FN0GmJuy1ew17YoUgQ-2RY0fp5Psg46tiWoZLDyz7M>

Aprender a:

- Cargar un modelo bruto, el modelo **movenet/singlepose/lighting**
- Normalizar los datos de entradas de acuerdo a las necesidades del modelo
- Ejecutar el modelo bruto movenet/singlepose/lighting para realizar estimaciones sobre fotogramas de la cámara web
- Visualizar la salida de MoveNet (17 puntos clave del cuerpo)

Resumen

- Existen dos tipos de modelos pre-entrenados brutos (o sin procesar): (1) Modelos de capas (Layers Model), (2) Modelo de grafo (Graph Model):
- TFJS proporciona modelos pre-entrenados brutos para detectar la ubicación de las articulaciones humanas de una sola persona: MoveNet
- Un tensor es un objeto de la clase `tf.Tensor`
- TFJS proporciona métodos para realizar operaciones con tensores