

TensorFlow.js Cheat Sheet

Tensors

Dimensions are given as **height**, **width**, and **depth**, OR **number of channels**. This format is used in many TFJS methods. These values are an example of the dimensions of a common desktop image size

```
const dims = [1080, 1920, 3]
```

Create a tensor with given dimensions.

```
tf.tensor(myArray, dimensions)
```

This example creates a 3-high, 2-wide, 1-deep tensor

```
tf.tensor([1, 2, 7, 49, 733, 29760], [3, 2, 1])
```

Special creator function for 2d tensors

```
tf.tensor2d([[4, 832, 22708], [312956, 2716096, 17117832]])
```

Create a tensor of ones, any dimensions

```
tf.ones(dims)
```

Create a tensor filled with a given value

```
tf.fill(dims, value)
```

Example - creates a 2-high, 4-wide tensor filled with 5s (but what about Rex?!)

```
tf.fill([2, 4], 5)
```

Create a tensor where each value is randomly assigned from a uniform distribution over the interval [minValue, maxValue]

```
tf.randomUniform(dims, minValue, maxValue)
```



```
tf.randomUniform([108, 192, 3], 0, 1)
```

Join Tensors, stackingDimension 0=y, 1=x
You wouldn't typically stack on channels.

```
tf.concat([tensor1, tensor2], stackingAxis)
```

Note: This is a special case of concat that always stacks along the y axis

```
tf.stack([tensor1, tensor2, tensor3, ...])
```

Reverse the order of elements along a given axis

```
tf.reverse(myTensor, reversingAxis)
```

Introspection. Prints values of tensor in proper shape

```
myTensor.print()
```

Output the shape of a tensor (height, width, channels).

```
myTensor.shape // => [ 3, 3, 1 ] (for example)
```

TensorFlow.js Cheat Sheet

Models: Training and Prediction

Create a sequential model

```
const model = tf.sequential()
```

Create and add a layer to the model. Can use other kinds of layers besides dense.

```
const layer = tf.layers.dense(  
  units: 10, inputShape: [2], activation:  
  'relu' ) model.add(layer)
```

Compile

```
model.compile({optimizer: "sgd",  
  loss: "meanSquaredError"})
```

Train

```
model.fit(xs, ys, { epochs: 500 })
```

Load and predict

```
const model =  
tf.loadLayersModel(modelURL)  
const result = model.predict(input)
```

Convert image to tensor

```
const myTensor =  
tf.browser.fromPixels(image,  
  numberOfChannels)
```

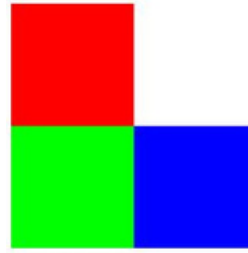
Convert tensor back to image

```
const printCanvas =  
document.getElementById("#dom-element")  
const image = tf.browser.toPixels(tensor,  
  printCanvas)
```

Resize without interpolating data (

```
tf.image.resizeNearestNeighbor(image,  
  size, true)
```

fig. a



resizeNearestNeighbor

fig. b



resizeBilinear

Resize without interpolating data. `alignCorners` should usually be true ()

```
tf.image.resizeBilinear(image, size,  
  alignCorners)
```

Use for tensors with values in 0-255

```
myTensor.asType('int32')
```

(Example)

```
const myTensor =  
tf.randomUniform(dimensions, 0, 255))
```

Use for tensors with values in 0-1

```
myTensor.asType('float32')
```

(Example)

```
const myTensor =  
tf.randomUniform(dimensions, 0, 1)
```

Coordinates

```
const startPosition = [y, x, z]
```

Crop an image

```
const cropped =  
myImageTensor.slice(startPosition, dims)
```