

# Intro to TensorFlow 2.0

Easier for beginners, more powerful for experts



Josh Gordon

[twitter.com/random\\_forests](https://twitter.com/random_forests)



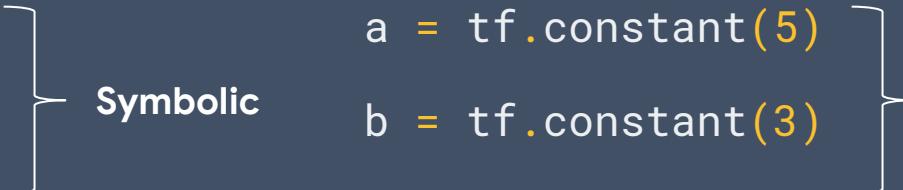
These are the slides from this talk

<https://www.youtube.com/watch?v=5ECD8J3dvDQ>

# Functions, not sessions

TF1

```
a = tf.constant(5)  
b = tf.constant(3)  
c = a * b
```



Symbolic

TF2

```
a = tf.constant(5)  
b = tf.constant(3)  
c = a * b
```

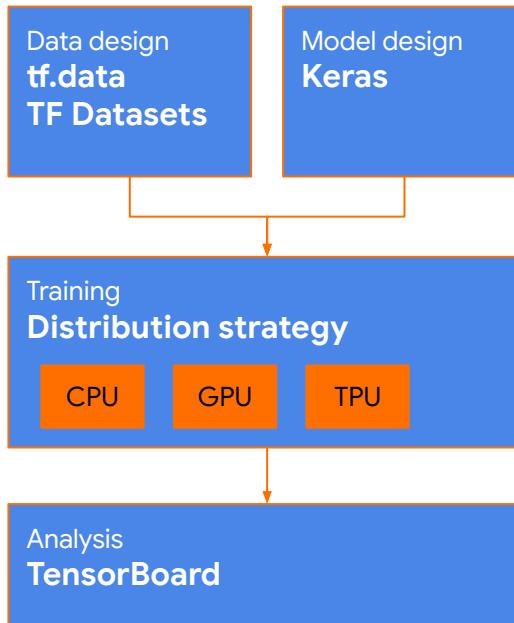


Concrete

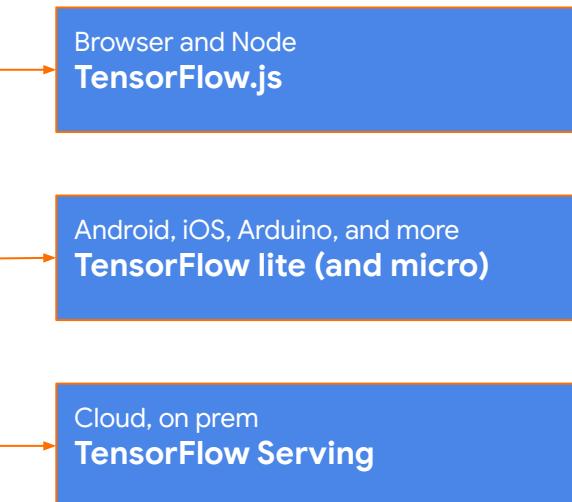
```
with tf.Session() as sess:  
    print(c)  
  
    print(sess.run(c))
```



## Training

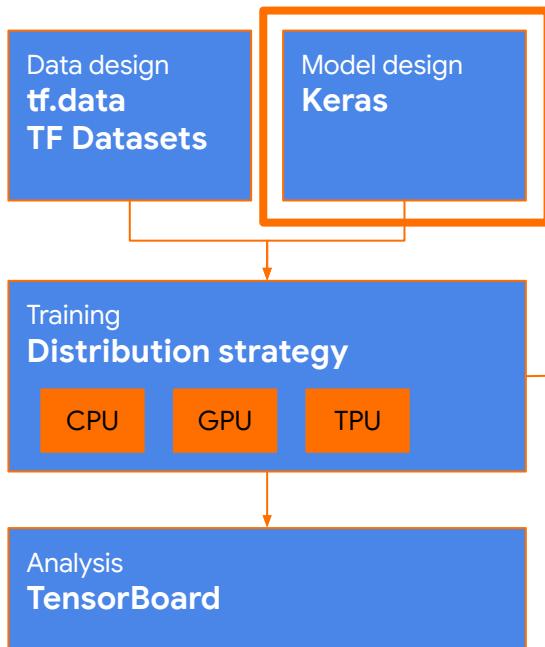


## Deployment

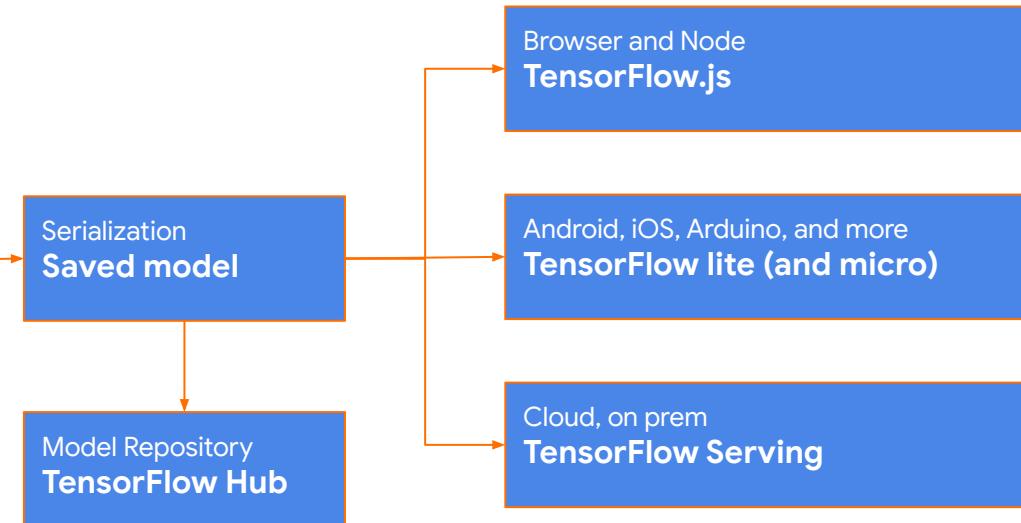




## Training



## Deployment





# Model building: from simple to arbitrarily flexible

Progressive disclosure of complexity

Sequential API  
+ built-in layers

Functional API  
+ built-in layers

Functional API  
+ Custom layers  
+ Custom metrics  
+ Custom losses

Subclassing: write  
everything yourself  
from scratch



New users,  
simple models

for building a  
stack of layers

by far the easiest and  
clearest way to develop a  
deep learning model today

Engineers with  
standard use  
cases

for building DAGs  
— Directed Graphs

Engineers  
requiring  
increasing  
control

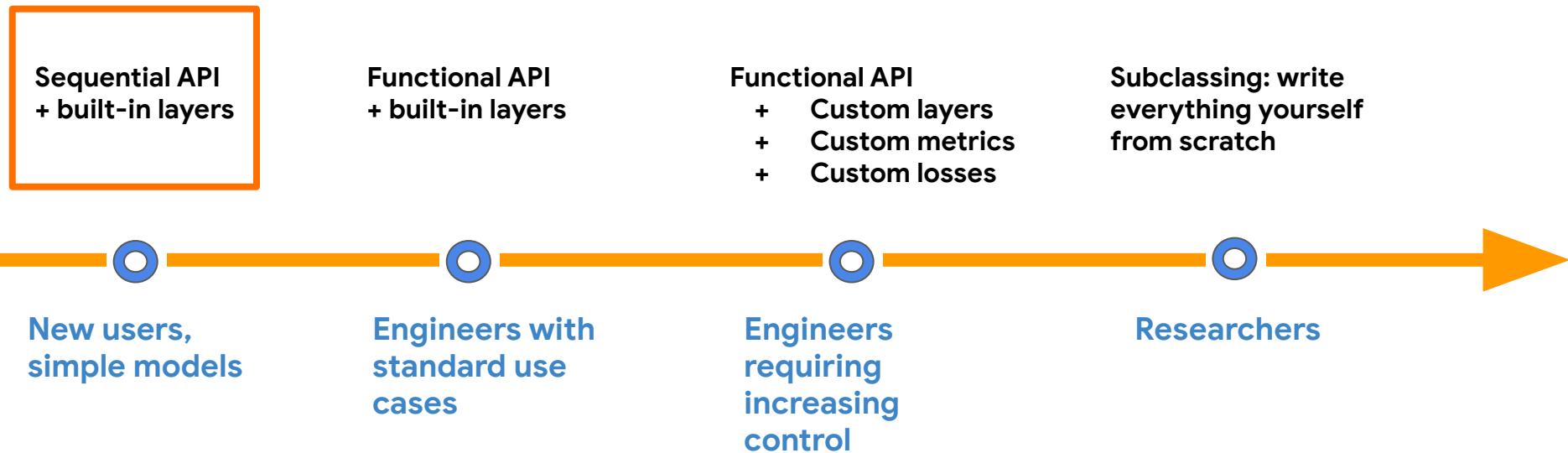
Researchers





# Model building: from simple to arbitrarily flexible

Progressive disclosure of complexity

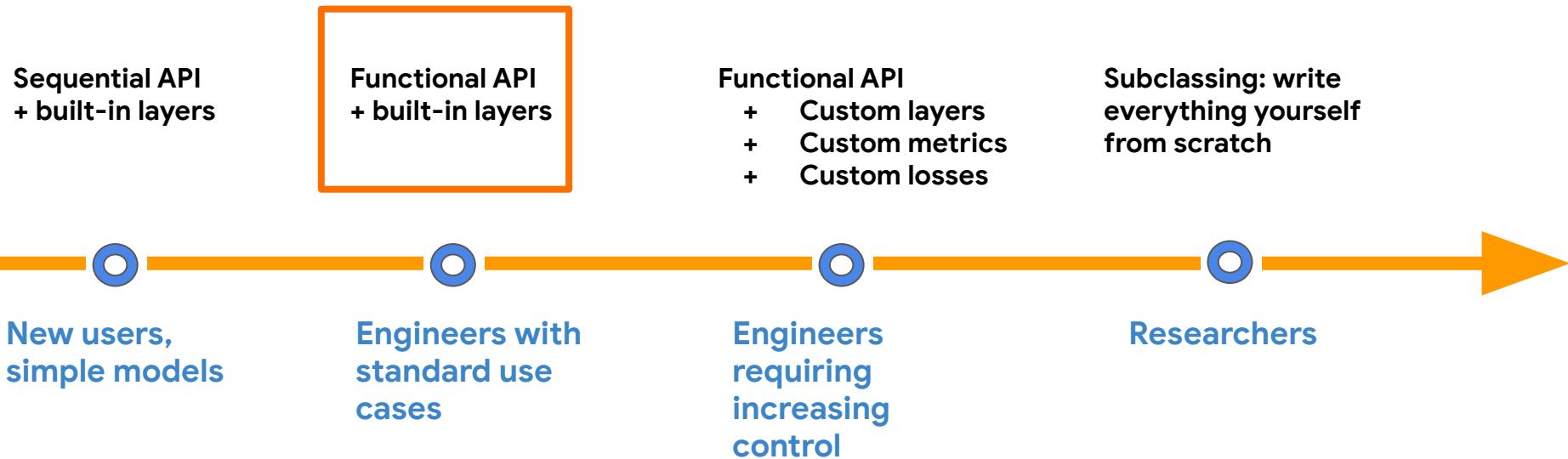


```
model = keras.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(784,)))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(32, activation='softmax'))
```



# Model building: from simple to arbitrarily flexible

Progressive disclosure of complexity





# Visual Question Answering



**Question:** What color is the dog on the right?

**Answer:** Golden

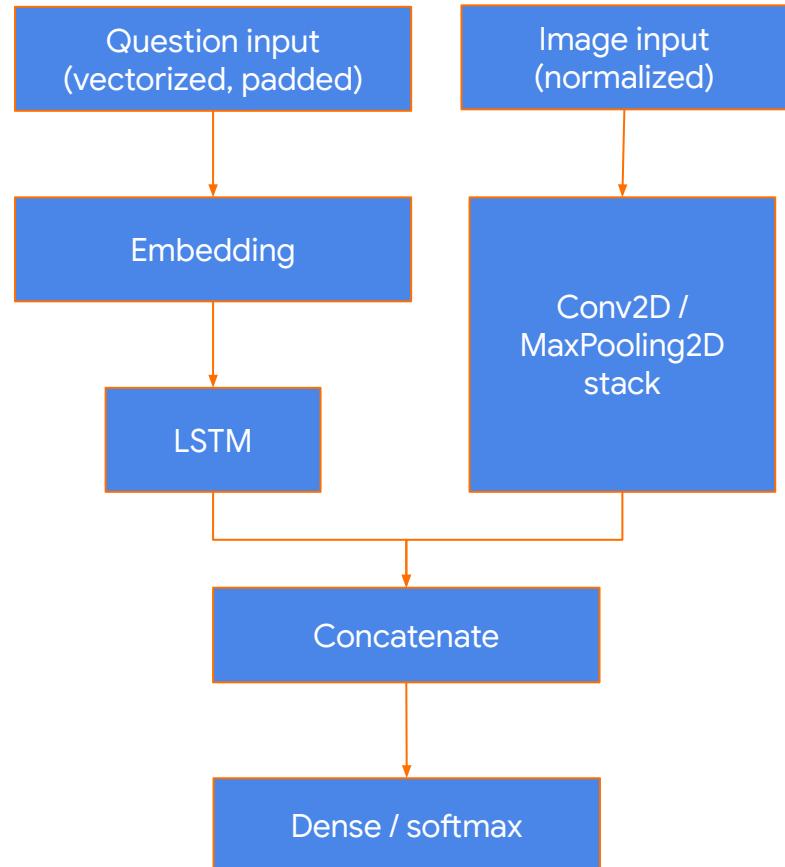


# Workflow

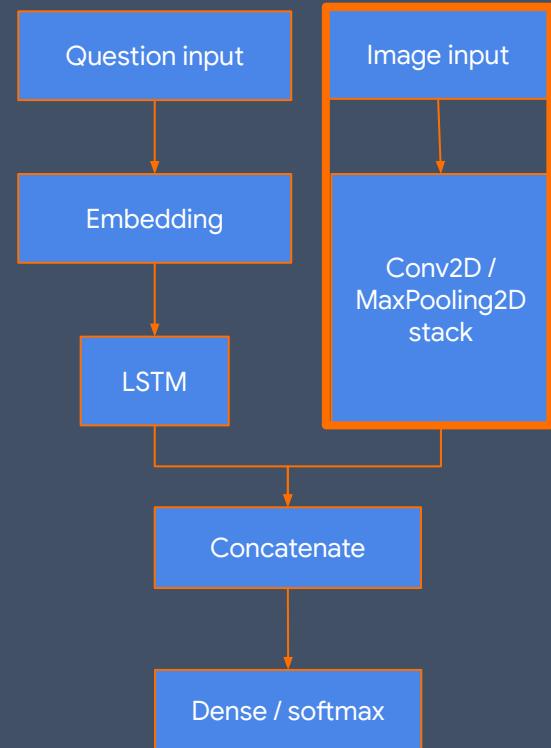
## A multi-input model

1. Use a CNN to embed the image
2. Use a LSTM to embed the question
3. **Concatenate**
4. Classify with Dense layers, per usual

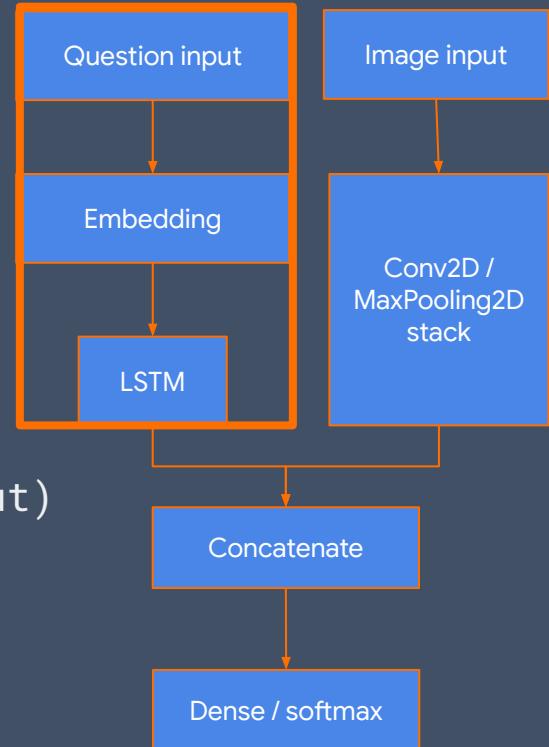
A wonderful thing about Deep Learning: ability to mix data types (text, images, timeseries, structured data) in a single model.



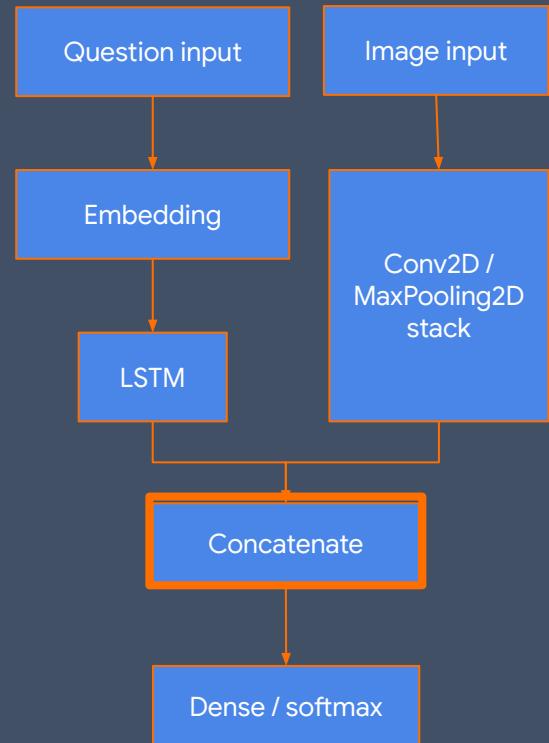
```
# A vision model.  
# Encode an image into a vector.  
vision_model = Sequential()  
vision_model.add(Conv2D(64, (3, 3),  
                      activation='relu',  
                      input_shape=(224, 224, 3)))  
vision_model.add(MaxPooling2D())  
vision_model.add(Flatten())  
  
# Get a tensor with the output of your vision model  
image_input = Input(shape=(224, 224, 3))  
encoded_image = vision_model(image_input)
```



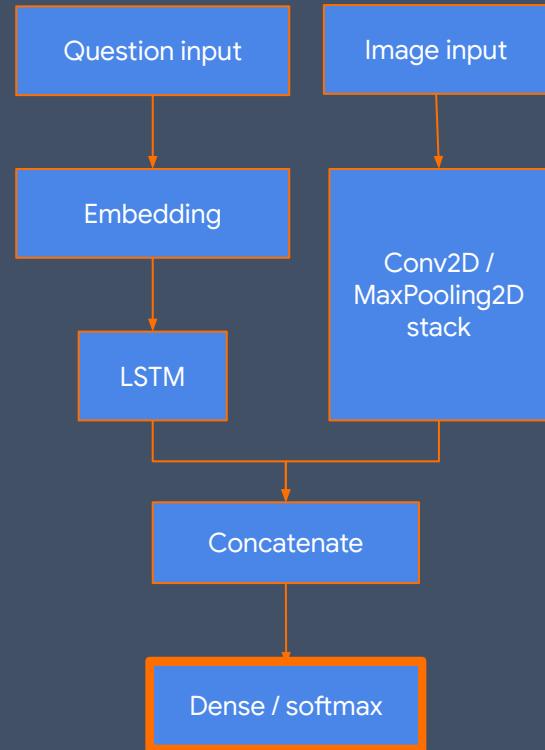
```
# A language model.  
# Encode the question into a vector.  
question_input = Input(shape=(100,),  
                      dtype='int32',  
                      name="Question")  
  
embedded = Embedding(input_dim=10000,  
                      output_dim=256,  
                      input_length=100)(question_input)  
  
encoded_question = LSTM(256)(embedded)
```



```
# Concatenate the encoded image and question  
merged = layers.concatenate([encoded_image,  
                            encoded_question])
```



```
# Train a classifier on top.  
output = Dense(1000,  
               activation='softmax')(merged)  
  
# You can train w/ .fit, .train_on_batch,  
# or with a GradientTape.  
vqa_model = Model(inputs=[image_input,  
                         question_input],  
                   outputs=output)
```



```
from tensorflow.keras.utils import plot_model  
plot_model(vqa_model, to_file='model.png')
```

Show the model graph that looks exactly  
like what I just showed you.



# Model building: from simple to arbitrarily flexible

Progressive disclosure of complexity

Sequential API  
+ built-in layers

Functional API  
+ built-in layers

Functional API  
+ Custom layers  
+ Custom metrics  
+ Custom losses

Subclassing: write  
everything yourself  
from scratch



New users,  
simple models

Engineers with  
standard use  
cases

Engineers  
requiring  
increasing  
control

Researchers



```
class MyModel(tf.keras.Model):                                in the constructor you define your layers
    def __init__(self, num_classes=10):
        super(MyModel, self).__init__(name='my_model')
        self.dense_1 = layers.Dense(32, activation='relu')
        self.dense_2 = layers.Dense(num_classes, activation='softmax')

    def call(self, inputs):
        # Define your forward pass here
        x = self.dense_1(inputs)
        return self.dense_2(x)
```

This feels a lot like Object-Oriented NumPy development.

```
class MyModel(tf.keras.Model):
    def __init__(self, num_classes=10):
        super(MyModel, self).__init__(name='my_model')
        self.dense_1 = layers.Dense(32)
        self.dense_2 = layers.Dense(num_classes, activation='softmax')

    def call(self, inputs):
        # Define your forward pass here
        x = self.dense_1(inputs)
        x = tf.nn.relu(x)                                in case you wanna to write your own activation function
        return self.dense_2(x)
```



# Helpful references

## Guides

[tensorflow.org/guide/keras/overview](https://tensorflow.org/guide/keras/overview)

[tensorflow.org/guide/keras/functional](https://tensorflow.org/guide/keras/functional)

[tensorflow.org/guide/keras/train\\_and\\_evaluate](https://tensorflow.org/guide/keras/train_and_evaluate)

[tensorflow.org/guide/keras/custom\\_layers\\_and\\_models](https://tensorflow.org/guide/keras/custom_layers_and_models)

## Examples

[tensorflow.org/tutorials/images/segmentation](https://tensorflow.org/tutorials/images/segmentation)

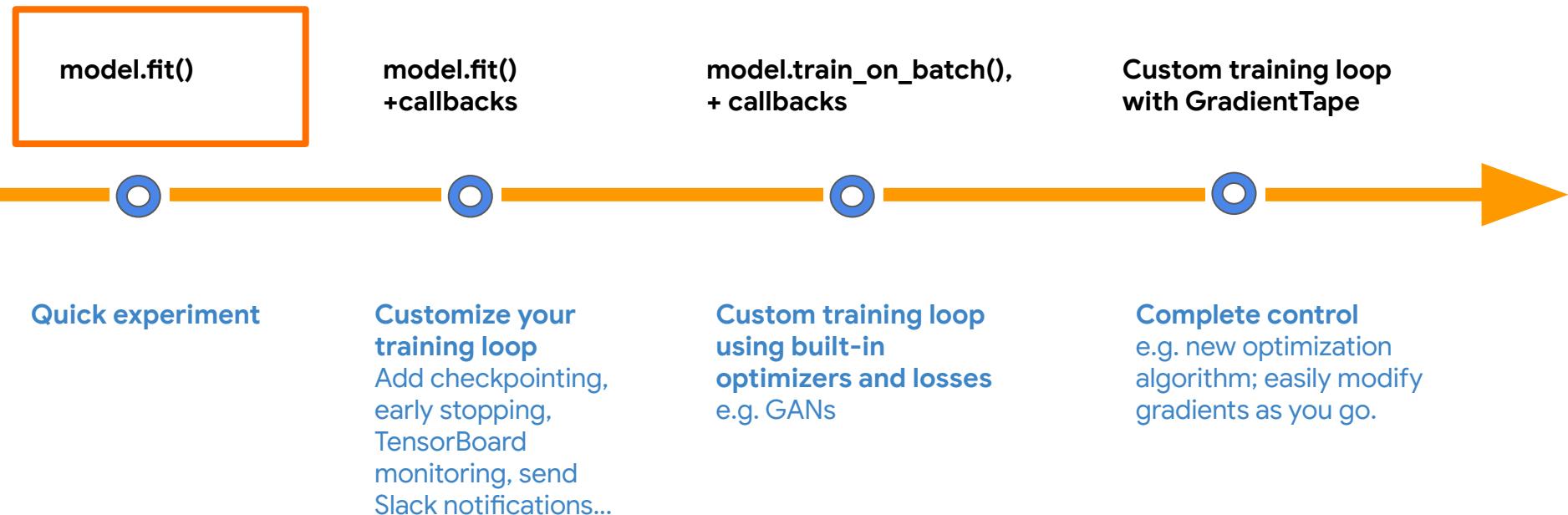
[tensorflow.org/tutorials/generative/pix2pix](https://tensorflow.org/tutorials/generative/pix2pix)

[tensorflow.org/tutorials/generative/adversarial\\_fgsm](https://tensorflow.org/tutorials/generative/adversarial_fgsm)



# Model training: from simple to arbitrarily flexible

Progressive disclosure of complexity



```
model.compile(optimizer=Adam(),  
              loss=BinaryCrossentropy(),  
              metrics=[AUC(), Precision(), Recall()])
```

```
model.fit(data,  
          epochs=10,  
          validation_data=val_data,  
          callbacks=[EarlyStopping(),  
                    TensorBoard(),  
                    ModelCheckpoint()])
```

Typically, when we're training models, we need to prevent overfitting. And a really wonderful way to do that is to make plots of your loss over time and so on and so forth. These callbacks can do things like that for you automatically.

**...or write your own callbacks!**



# Model training: from simple to arbitrarily flexible

Progressive disclosure of complexity

`model.fit()`

`model.fit()`  
+ callbacks

`model.train_on_batch()`,  
+ callbacks

**Custom training loop  
with GradientTape**



Quick experiment

**Customize your  
training loop**  
Add checkpointing,  
early stopping,  
TensorBoard  
monitoring, ...

**Custom training loop  
using built-in  
optimizers and losses**  
e.g. GANs

**Complete control**  
e.g. new optimization  
algorithm; easily modify  
gradients as you go.

powerful especially for students who are  
learning this for the first time and don't want a  
black box or for researchers.

# Graphs with one LOC

```
@tf.function
```

```
def train_step(features, labels):
    with tf.GradientTape() as tape:
        logits = model(features, training=True)
        loss = loss_fn(labels, logits)

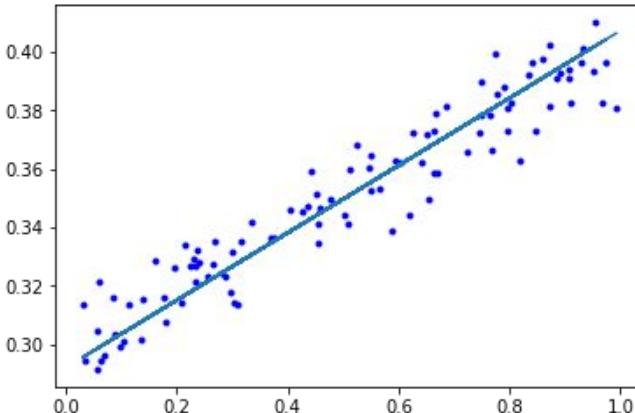
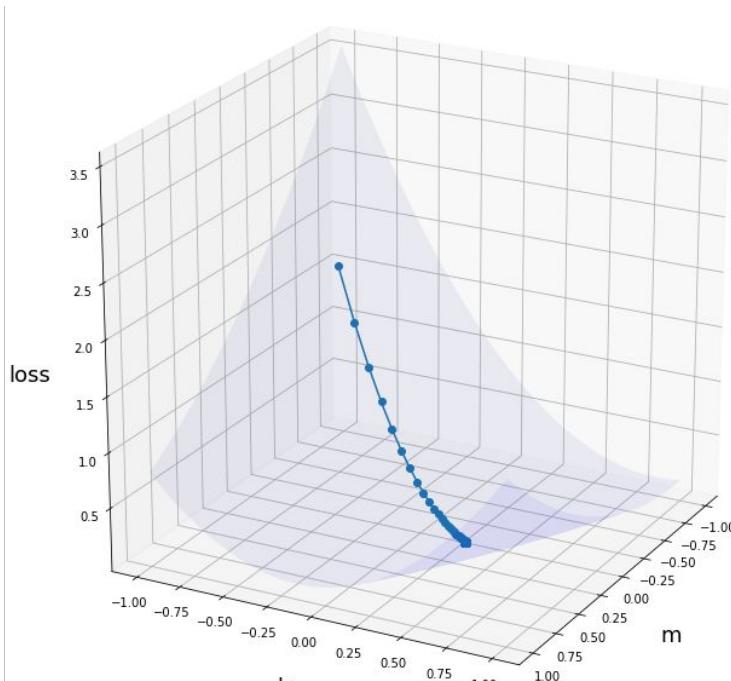
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))
    return loss
```

Here is a custom training loop. And I have an example of this for you in a minute, just with linear regression.

- we have some function that's taking features and labels as input
- whenever we're doing training in deep learning, we're doing gradient descent, the first step in doing gradient descent is getting the gradients.
- the way all frameworks do this is by backpropagation. And the implementation in TensorFlow is we start recording operations on a tape.



# Low-level details: Linear regression



# Constants

```
x = tf.constant([[5, 2], [1, 3]])
print(x)
```

```
tf.Tensor(
[[5 2]
 [1 3]], shape=(2, 2), dtype=int32)
```

You can also use TensorFlow 2.0 a lot like you would use Numpy.  
Basically whenever you see something like tensor, just replace  
that in your head with NumPy ndarray.

# Tensors to NumPy

```
x.numpy()
```

```
array([[5, 2],  
       [1, 3]], dtype=int32)
```

# Shape and dtype

```
print('shape:', x.shape)
print('dtype:', x.dtype)
```

shape: (2, 2)

dtype: <dtype: 'int32'>

# Distributions

```
print(tf.random.normal(shape=(2, 2), mean=0., stddev=1.))

<tf.Tensor: id=1652819, shape=(2, 2), dtype=float32, numpy=
array([[-0.5636922,  2.6130383],
       [ 0.2622066,  0.7773327]], dtype=float32)>
```

# Math in TF2 feels like NumPy

```
a = tf.random.normal(shape=(2, 2))
b = tf.random.normal(shape=(2, 2))
c = a + b
d = tf.square(c)
```

# Gradients

Here's how we get the gradients using the GradientTape.

```
x = tf.constant(3.0)
with tf.GradientTape() as g:
    g.watch(x)
    y = x * x
dy_dx = g.gradient(y, x) # 6.0
```

# Variables are automatically tracked

```
dense1 = tf.keras.layers.Dense(32)  
dense2 = tf.keras.layers.Dense(32)
```

Here we have a pair of dense layers.

```
with tf.GradientTape() as tape:  
    result = dense2(dense1(tf.zeros([1, 10])))  
    tape.gradient(result, dense1.variables)
```

As we saw previously, you can get the gradients for `model.trainable_variables` as well in one go.



# Quick demo

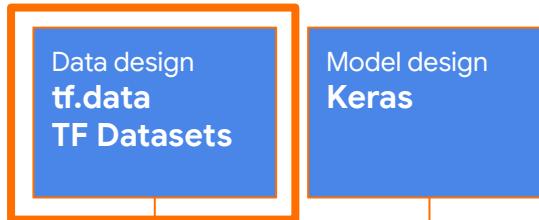
[Link](#)

```
m = tf.Variable(0.)
```

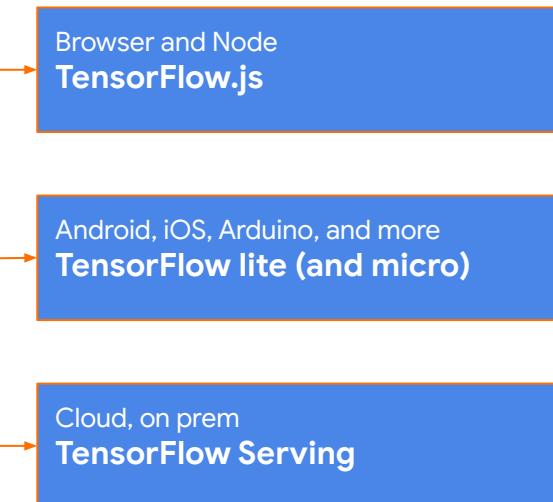
You almost never need to write code this low level. This is pretending that we don't have Keras. We don't have any built-in fit methods. We just want to do this from scratch.



## Training



## Deployment



```
# Keras datasets
from tensorflow.keras import datasets
(train_images, train_labels), \
(test_images, test_labels) = datasets.cifar10.load_data()
```

good to start with. They're in Numpy format. And they're usually really tiny. They fit into memory no problem.

```
# TensorFlow Datasets
import tensorflow_datasets as tfds
dataset, metadata = tfds.load('cycle_gan/horse2zebra',
                               with_info=True,
                               as_supervised=True)
```

```
# If you're using TensorFlow Datasets
# Either load your dataset into memory, or
# write a performant input pipeline to load it off disk.
dataset, metadata = tfds.load('mnist',
                               with_info=True,
                               as_supervised=True,
                               in_memory=True)
```

```
# Caching is important to avoid repeated work
# Use either an in-memory cache, or a cache file
def preprocess(img):
    img = tf.cast(image, tf.float32)
    img = (img / 127.5) - 1
    img = tf.image.resize(img, [286, 286])
    # ...
    return img
```

```
image_ds = image_ds.map(
    preprocess, num_parallel_calls=AUTOTUNE).cache()
```

Here is some code for `ft.data`. And maybe we have an image dataset, and we have some code to preprocess the images. And let's pretend that preprocessing code is expensive and we don't want to run it every time, on every epoch. What you can do is you can add this `'cache'` line at the end.

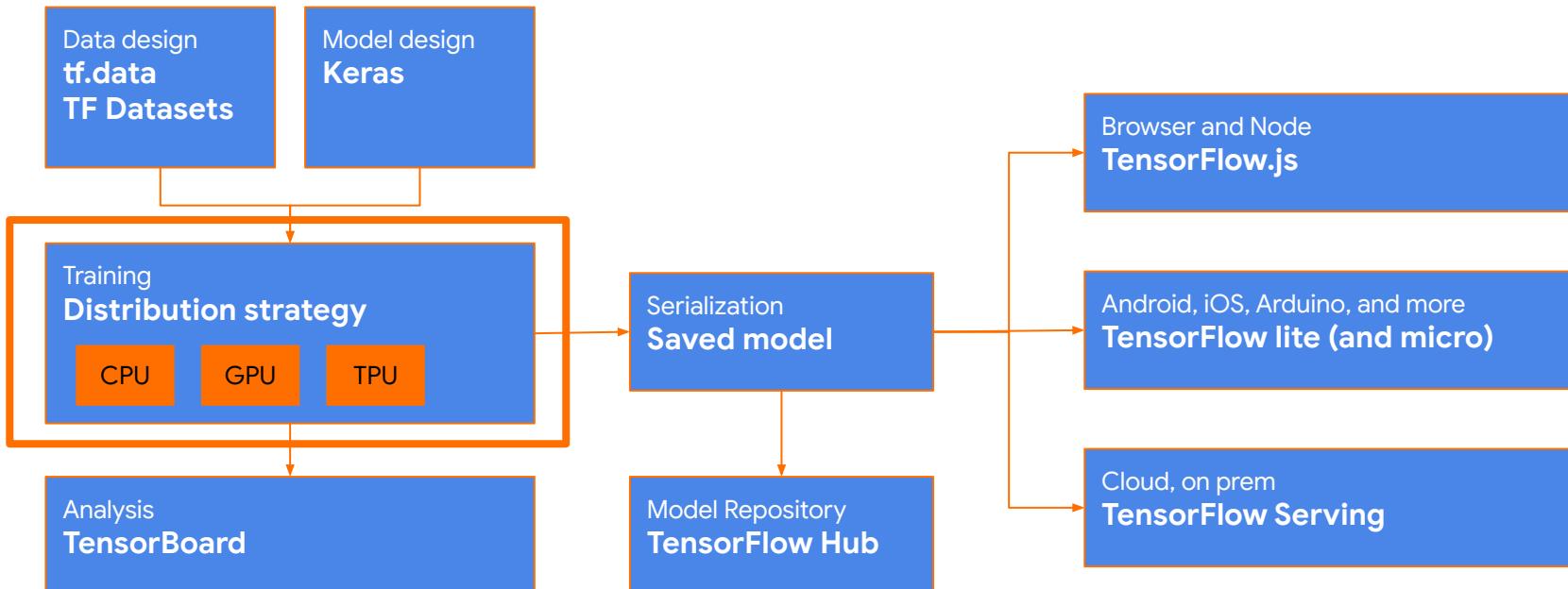
Note: order is important. Cache before shuffling and batching.

Helpful reference (on `tf.data`, loading images, and caching): [tensorflow.org/tutorials/load\\_data/images](https://tensorflow.org/tutorials/load_data/images)

List of TensorFlow Datasets: [tensorflow.org/datasets/catalog/overview](https://tensorflow.org/datasets/catalog/overview)



## Training



## Deployment



# Distribute, without changing your code

- No code changes for single machine, multi-GPU training
- No code changes for multi-machine, multi-GPU training

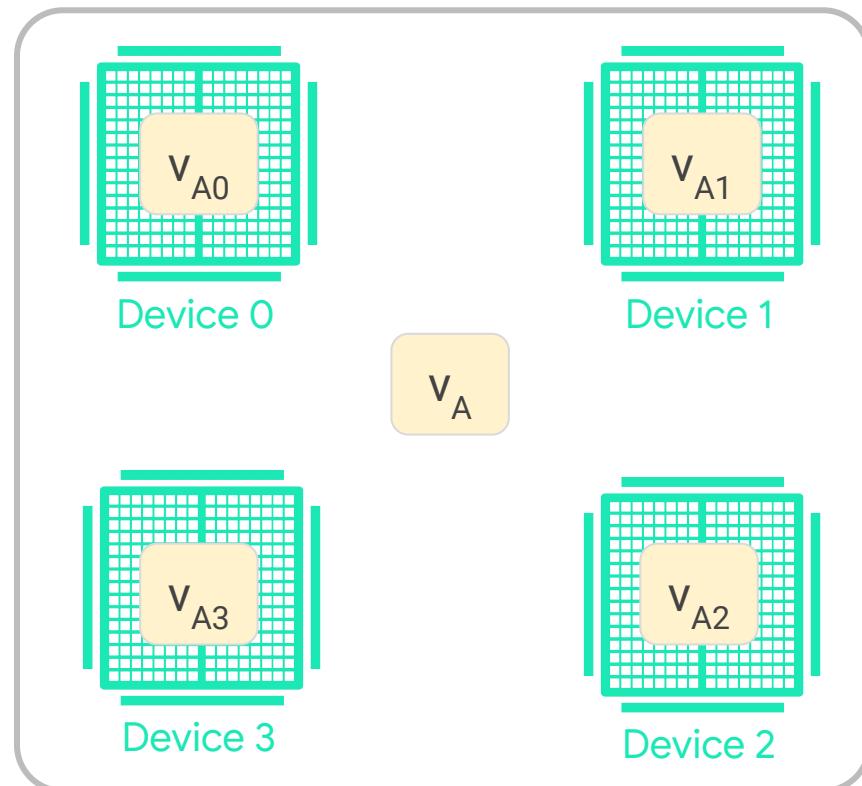
Synchronous data parallelism via efficient all reduce.



# MirroredStrategy

## Multi-GPU training

- Synchronous data parallelism.
- Variables mirrored on each GPU.
- Replicas are run in lock-step..
- All-reduce: network efficient way to aggregate gradients.



```
import tensorflow as tf

model = tf.keras.applications.ResNet50()
optimizer = tf.keras.optimizers.SGD(learning_rate=0.1)
model.compile(..., optimizer=optimizer)
model.fit(train_dataset, epochs=10)
```

```
import tensorflow as tf

strategy = tf.distribute.MirroredStrategy()
with strategy.scope():

    model = tf.keras.applications.ResNet50()
    optimizer = tf.keras.optimizers.SGD(learning_rate=0.1)
    model.compile(..., optimizer=optimizer)
    model.fit(train_dataset, epochs=10)
```

## Distribute-aware

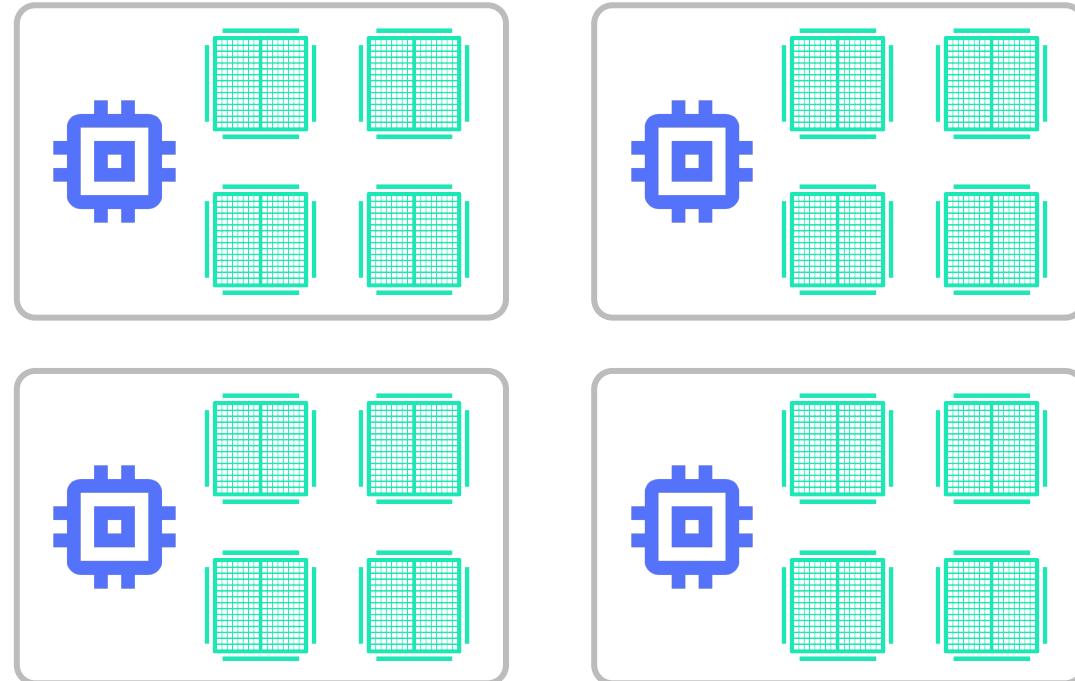
- Strategy
  - MirroredStrategy
  - MirroredMultiWorker
    - if you have a network with multiple machines on it.



# MultiWorkerMirroredStrategy

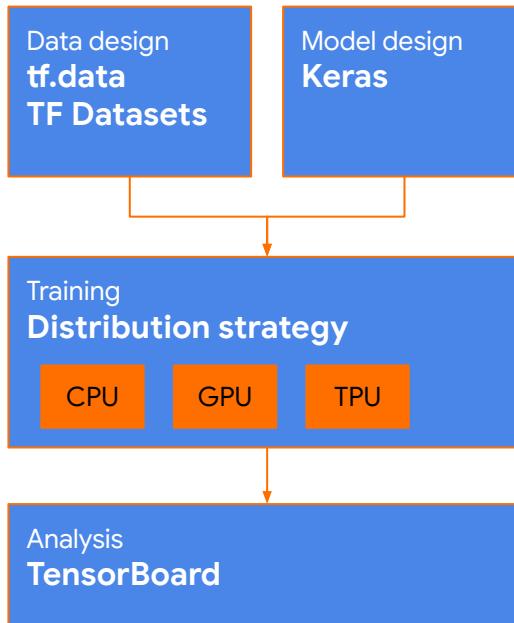
Multi-machine,  
multi-GPU training

- Efficient all reduce across multiple machines
- No additional code changes
- Cluster config environment variable



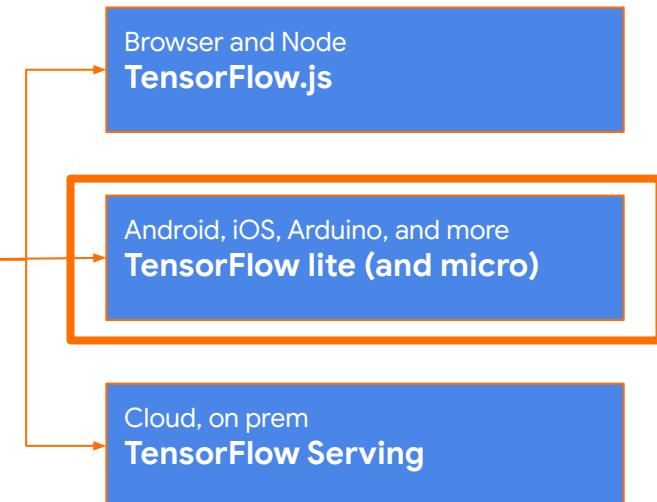


## Training



Let's look at low level details for a moment.

## Deployment



# Project suggestion: TinyML





# Train with Python; Deploy with Arduino

Gesture classification (on-device!)



[How-to Get Started with Machine Learning on Arduino, by Sandeep Mistry & Dominic Pajak](#)  
[tensorflow.org/lite/microcontrollers/overview](https://tensorflow.org/lite/microcontrollers/overview)

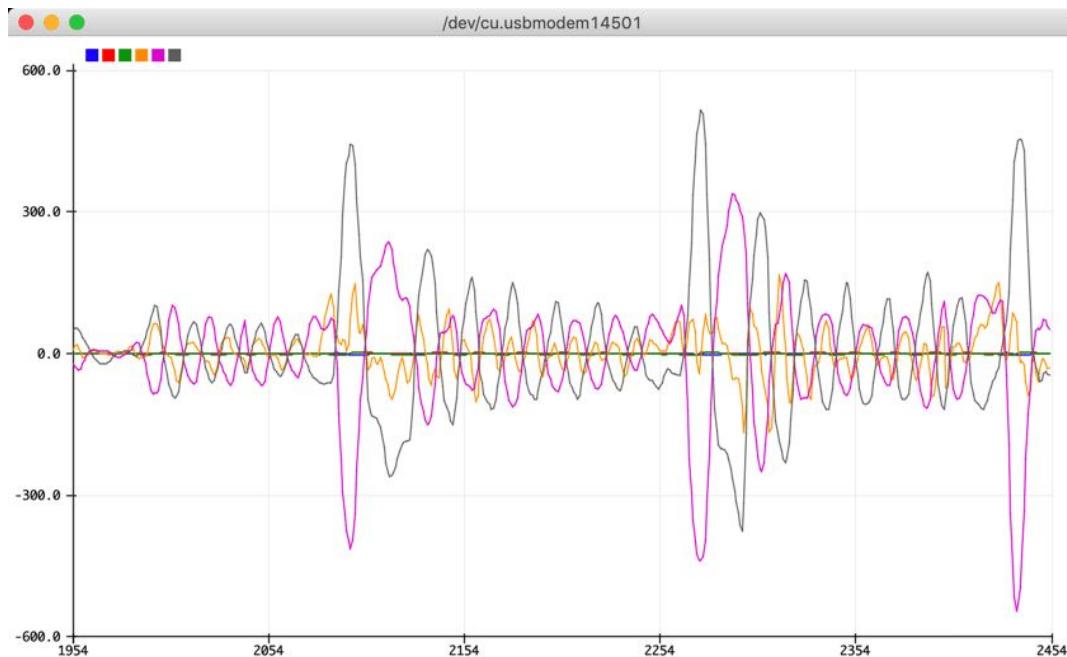


# Workflow

1. Capture data with the Arduino IDE for each gesture
2. Save CSVs (e.g. punch.csv, wave.csv)
3. Upload to Colab
4. Train a model with TF2 using Keras
5. Convert your model to TFLite
6. Install on device (walkthrough provided)



# Capturing data



```
08:23:46.849 -> 0.964,0.864,-0.691,108.643,36.438,-27.832
```

[How-to Get Started with Machine Learning on Arduino, by Sandeep Mistry & Dominic Pajak](#)

```
# Convert the model to the TFLite format without quantization
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model to disk
open("gesture_model.tflite", "wb").write(tflite_model)

# Check the size
import os
basic_model_size = os.path.getsize("gesture_model.tflite")
print("Model is %d bytes" % basic_model_size)
```



# Deploying on device

The screenshot shows the Arduino IDE interface with the title bar "IMU\_Classifier | Arduino 1.8.10". The tabs at the top are "IMU\_Classifier" (selected) and "model.h". The code editor contains the following code:

```
/*
IMU Classifier

This example uses the on-board IMU to start reading ac-
data from on-board IMU, once enough samples are read,
TensorFlow Lite (Micro) model to try to classify the m

Note: The direct use of C/C++ pointers, namespaces, an
discouraged in Arduino examples, and in the futu
might change to make the sketch simpler.

Compiling sketch...

```

At the bottom, it says "1" and "Arduino Nano 33 BLE on /dev/cu.usbmodem14501".

The screenshot shows the Arduino IDE interface with the title bar "IMU\_Classifier - model.h | Arduino 1.8.10". The tabs at the top are "IMU\_Classifier" and "model.h" (selected). The code editor contains the following code:

```
const unsigned char model[] = {
 0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00,
 0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x10,
 0x00, 0x00, 0x18, 0x00, 0x12, 0x00, 0x00, 0x00, 0x03,
 0xdc, 0x40, 0x02, 0x00, 0x10, 0x00, 0x00, 0x00, 0x1c,
 0x2c, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x00, 0x00, 0x01,
 0xe4, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0xac,
 0x0f, 0x00, 0x00, 0x00, 0x54, 0x4f, 0x43, 0x4f, 0x20,
 0x76, 0x65, 0x72, 0x74, 0x65, 0x64, 0x2e, 0x00, 0xd,
 0x80, 0x00, 0x00, 0x00, 0x74, 0x00, 0x00, 0x00, 0x68,
 0x5c, 0x00, 0x00, 0x00, 0x50, 0x00, 0x00, 0x00, 0x44,
};

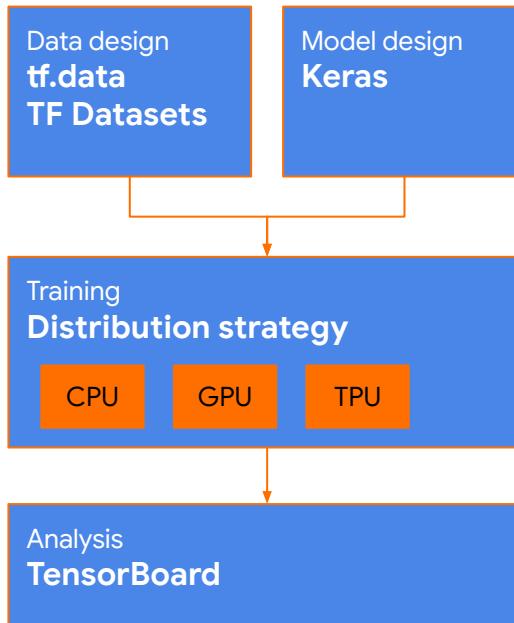
Compiling sketch...

```

At the bottom, it says "1" and "Arduino Nano 33 BLE on /dev/cu.usbmodem14501".

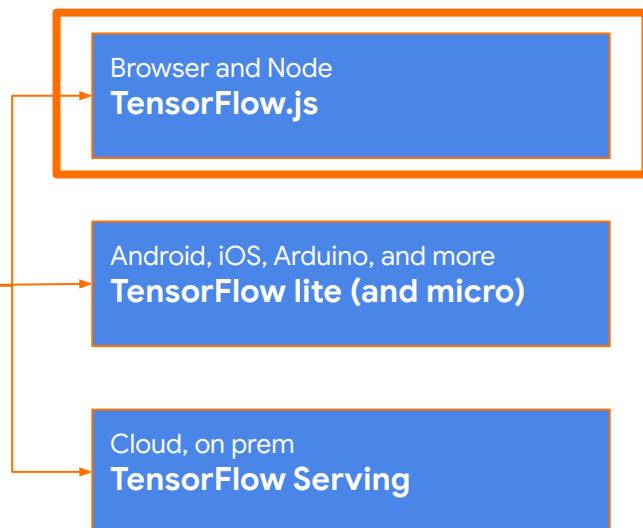


## Training



Let's look at low level details for a moment.

## Deployment



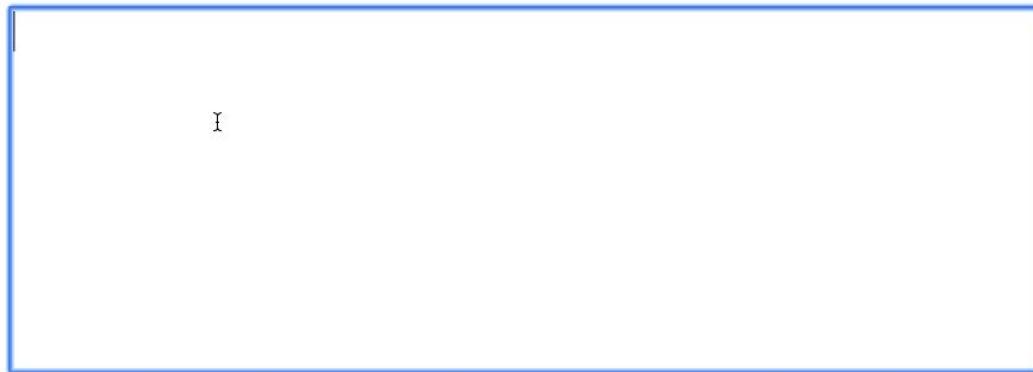
# Project suggestion: TF.js





# Train with Python; Deploy with JS

Sentiment analysis in the browser



## STATUS

Inference result (0 - negative; 1 - positive): 0.653573 (elapsed: 5.88 ms)



# Workflow

1. Train a model in Python
2. Save and convert your model (and metadata) to TF.js format
3. Upload to GitHub pages (or serve locally)
4. Run in the browser following the HTML and JS in the example

```
import tensorflowjs as tfjs

metadata = {
    'word_index': tokenizer.word_index,
    # ...
}

# Save metadata
metadata_json_path = os.path.join(FLAGS.artifacts_dir, 'metadata.json')
json.dump(metadata, open(metadata_json_path, 'wt'))

# Convert your model to TF.js format
tfjs.converters.save_keras_model(model, FLAGS.artifacts_dir)
```

Tip: you must preprocess text in the browser exactly as you do in Python.



## TensorFlow.js toxicity classifier demo

This is a demo of the TensorFlow.js toxicity model, which classifies text according to whether it exhibits offensive attributes (i.e. profanity, sexual explicitness). The samples in the table below were taken from this [Kaggle dataset](#).

text	identity attack	insult	obscene	severe toxicity	sexual explicit	threat	toxicity
We're dudes on computers, moron. You are quite astonishingly stupid.	false	true	false	false	false	false	true
Please stop. If you continue to vandalize Wikipedia, as you did to Kmart, you will be blocked from editing.	false	false	false	false	false	false	false
I respect your point of view, and when this discussion originated on 8th April I would have tended to agree with you.	false	false	false	false	false	false	false

Enter text below and click 'Classify' to add it to the table.

i.e. 'you suck'

CLASSIFY

## Applications

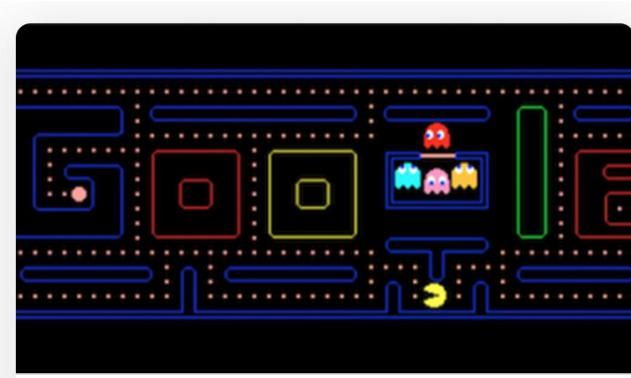
Imagine a tool to assist moderators.

Or, to prompt users.

All data stays client side.



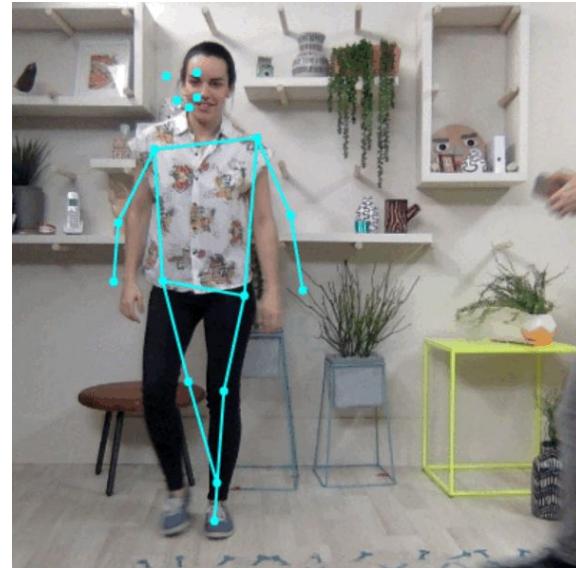
# Demos



## Webcam Controller

Play Pac-Man using images trained in your browser.

[Explore demo ↗](#) [View code](#)



[bit.ly/pose-net](http://bit.ly/pose-net)

[tensorflow.org/js/](http://tensorflow.org/js/)

# Notes





# Installing TF2

In Colab, run this command at the top of your notebook:

```
%tensorflow_version 2.x
```

To install locally, you can use pip.

- Visit [tensorflow.org/install](https://tensorflow.org/install)



# Keras vs tf.keras

In TF2, instead of writing “import keras” you write “from tensorflow import keras”.

In Colab, if you ever see the message “Using TensorFlow Backend”, you’ve imported the incorrect version.

# Learning more





# Learning more

## Practical books

- [Hands-on ML with Scikit-Learn, Keras and TensorFlow \(2nd edition\)](#)
- [Deep Learning with Python](#)
- [Deep Learning with JavaScript](#)
- [TinyML](#)

## Latest tutorials and guides

- [`tensorflow.org/tutorials`](#)
- [`tensorflow.org/guide`](#)

# Thank you!



Josh Gordon

[twitter.com/random\\_forests](https://twitter.com/random_forests)