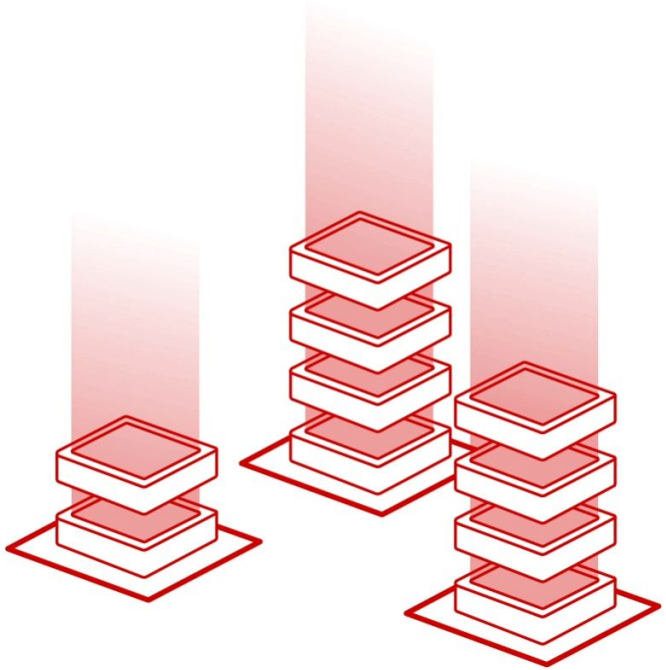
Google I/O 2019

Cutting Edge TensorFlow

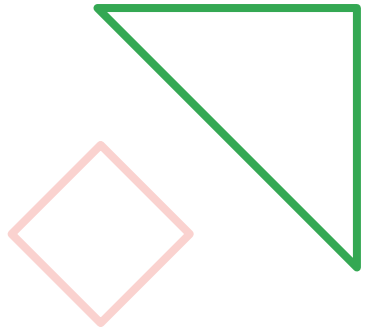
Keras Tuner: hypertuning for humans

Elie Bursztein
Google, @elie

How do I get the best TensorFlow model?



Getting the optimal model requires to tune many inter-dependent parameters

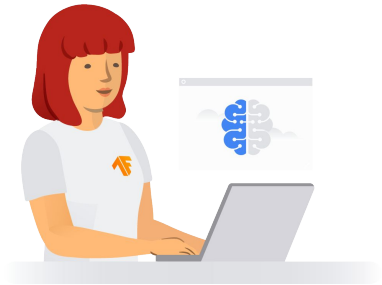


What if... hypertuning was as easy as 1, 2, 3?

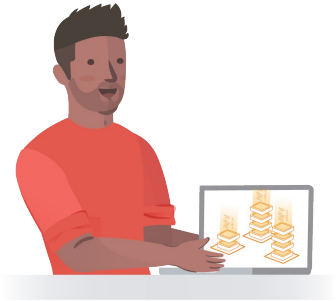
Introducing Keras Tuner

Hypertuning for humans

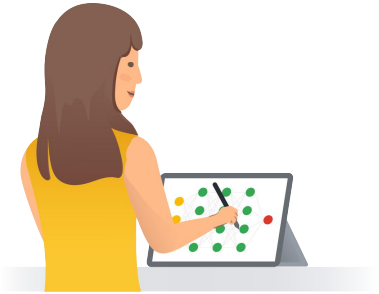
Keras Tuner is a framework designed for:



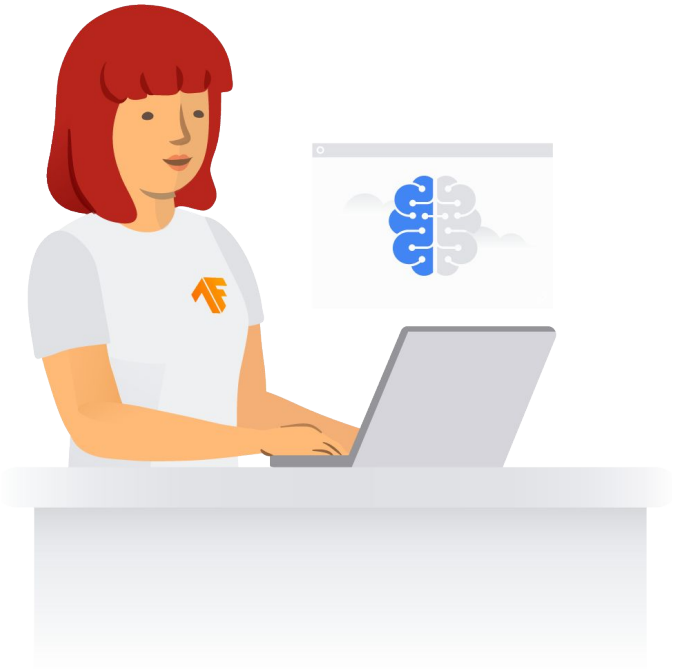
AI practitioners



Hypertuner algorithm creators



Model designers



With Keras Tuner hypertuning is just a few lines of code away

A basic MNIST model

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28,
28, 1)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(20, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer=Adam(0.001))
model.summary()
```

MNIST hypermodel is as easy as 1,2,3

1. Wrap model in a function

```
def model_fn():
```

MNIST hypermodel is as easy as 1,2,3

1. Wrap model in a function

2. Define hyper-parameters

```
def model_fn():

    LR = Choice('learning_rate', [0.001, 0.0005, 0.0001], group='optimizer')
    DROPOUT_RATE = Linear('dropout_rate', 0.0, 0.5, 5, group='dense')
    NUM_DIMS = Range('num_dims', 8, 32, 8, group='dense')
    NUM_LAYERS = Range('num_layers', 1, 3, group='dense')
    L2_NUM_FILTERS = Range('l2_num_filters', 8, 64, 8, group='cnn')
    L1_NUM_FILTERS = Range('l1_num_filters', 8, 64, 8, group='cnn')
```

MNIST hypermodel is as easy as 1,2,3

1. Wrap model in a function

2. Define hyper-parameters

3. Replace static value with hyper-parameters

```
def model_fn():

    LR = Choice('learning_rate', [0.001, 0.0005, 0.0001], group='optimizer')
    DROPOUT_RATE = Linear('dropout_rate', 0.0, 0.5, 5, group='dense')
    NUM_DIMS = Range('num_dims', 8, 32, 8, group='dense')
    NUM_LAYERS = Range('num_layers', 1, 3, group='dense')
    L2_NUM_FILTERS = Range('l2_num_filters', 8, 64, 8, group='cnn')
    L1_NUM_FILTERS = Range('l1_num_filters', 8, 64, 8, group='cnn')

    model = Sequential()
    model.add(Conv2D(L1_NUM_FILTERS, kernel_size=(3, 3), activation='relu'))
    model.add(Conv2D(L2_NUM_FILTERS, kernel_size=(3, 3), activation='relu'))
    model.add(Flatten())
    for _ in range(NUM_LAYERS):
      model.add(Dense(NUM_DIMS, activation='relu'))
      model.add(Dropout(DROPOUT_RATE))
    model.add(Dense(10, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=Adam(LR))
    return model
```

MNIST hypermodel is as easy as 1,2,3

1. Wrap model in a function

2. Define hyper-parameters

3. Replace static value with hyper-parameters

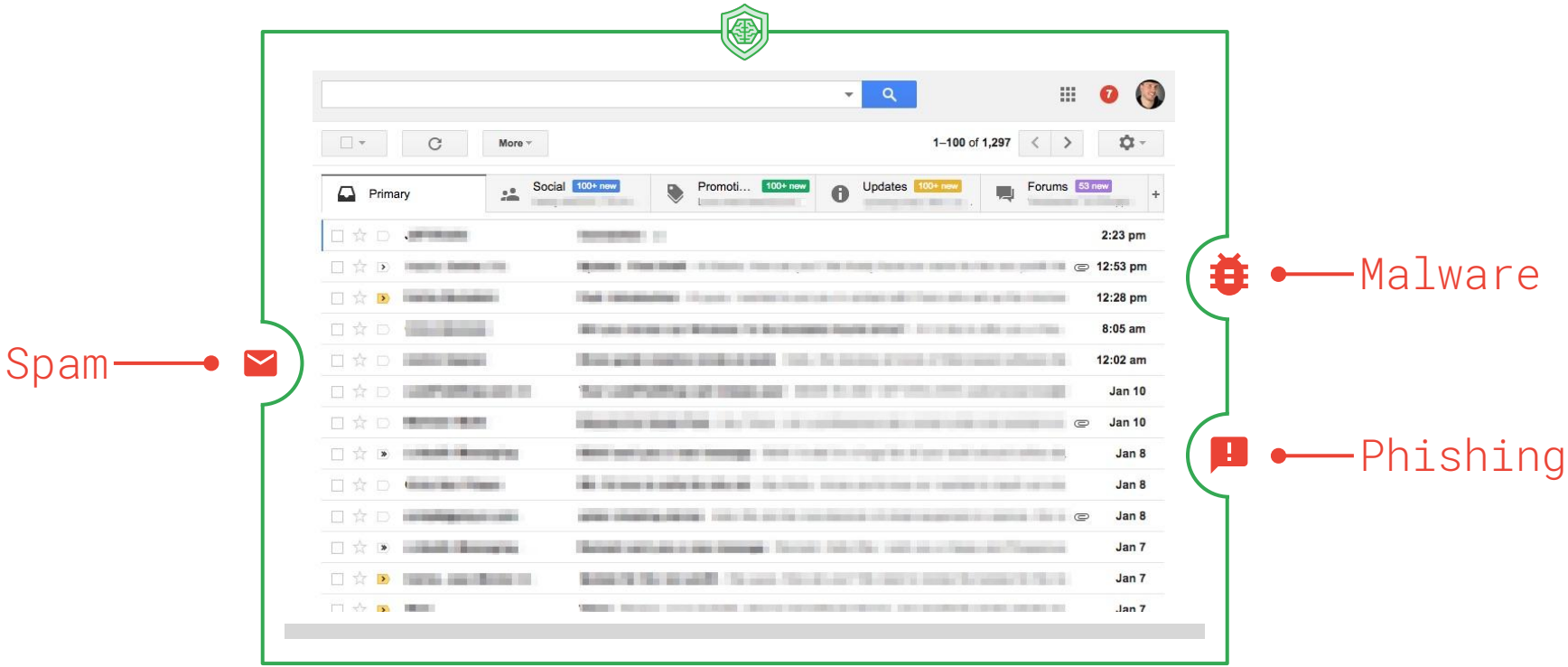
```
def model_fn():

    LR = Choice('learning_rate', [0.001, 0.0005, 0.0001], group='optimizer')
    DROPOUT_RATE = Linear('dropout_rate', 0.0, 0.5, 5, group='dense')
    NUM_DIMS = Range('num_dims', 8, 32, 8, group='dense')
    NUM_LAYERS = Range('num_layers', 1, 3, group='dense')
    L2_NUM_FILTERS = Range('l2_num_filters', 8, 64, 8, group='cnn')
    L1_NUM_FILTERS = Range('l1_num_filters', 8, 64, 8, group='cnn')


    model = Sequential()
    model.add(Conv2D(L1_NUM_FILTERS, kernel_size=(3, 3), activation='relu'))
    model.add(Conv2D(L2_NUM_FILTERS, kernel_size=(3, 3), activation='relu'))
    model.add(Flatten())
    for _ in range(NUM_LAYERS):
        model.add(Dense(NUM_DIMS, activation='relu'))
        model.add(Dropout(DROPOUT_RATE))
    model.add(Dense(10, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=Adam(LR))
    return model
```

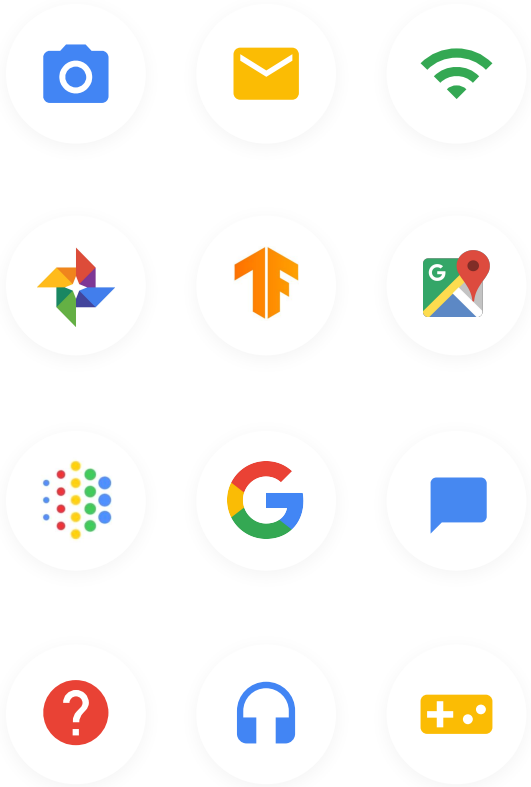
Intuitive
API

State of the art
hypertuner algorithms

Tunable architectures
ready to go

Seamless experiments
recording


Spam
Malware
Phishing
More
1–100 of 1,297
Primary
Social
100+ new
Promoti...
100+ new
Updates
100+ new
Forums
53 new
2:23 pm
12:53 pm
12:28 pm
8:05 am
12:02 am
Jan 10
Jan 10
Jan 8
Jan 8
Jan 8
Jan 7
Jan 7

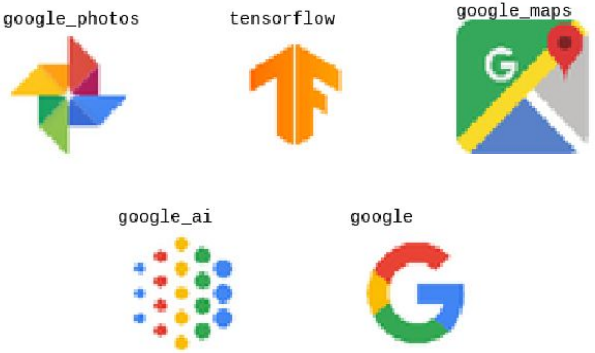


Let's build a **simple** logo classifier!

```python
from icongenerator.dataset import io19

ICON_SIZE = 100
NUM_EPOCHS = 5
BATCH_SIZE = 128
NUM_GEN_ICONS_PER_EPOCH = 50000
dataset = io19.download()
```

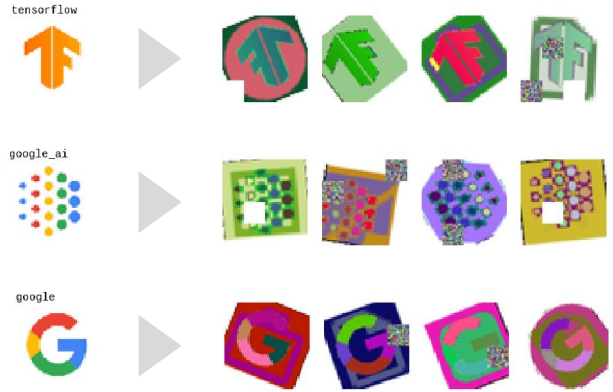
Loading the dataset

151 logo & icons



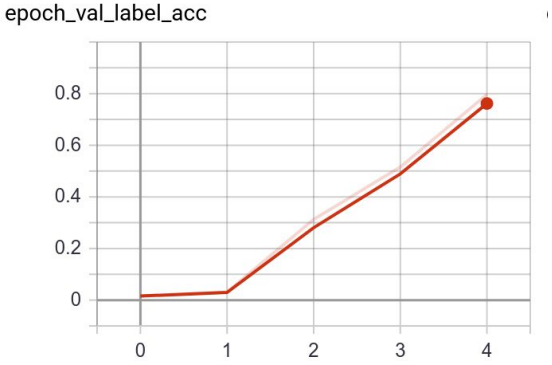
```python
from icongenerator import TFGenerator

AUGMENTATION_PARAMS = {
    "background_frequency": 0.5,
    "shape_frequency": 0.5,
    "min_logo_size": 0.50,
    "max_logo_size": 0.95,
    "use_flips": True,
    "occlusion":  "light",
    "use_logo_color_shift": True,
    "blur": 1.0,
    "max_rotation": 45,
    "max_affine_transformations": 2,
    "use_background_color_shift": True,
    "use_shape_color_shift": True,
    "min_logo_alpha": 230
}


tfg = TFGenerator(dataset, icon_size=ICON_SIZE,
**AUGMENTATION_CONFIG)
```

Adding data augmentation



Training ResNet101v2 as baseline

```
base_model = ResNet101v2(input_shape=(100, 100, 3), weights=None, include_top=False)
x = GlobalAveragePooling2D()(base_model.output)

y = Dense(100 * 100 * 3, activation='sigmoid')(x)
decoded = Reshape((100, 100, 3), name='reconstruction')(y)

x = Flatten()(x)
prediction = Dense(kg.num_classes, activation='softmax', name='label')(x)
model = Model(base_model.input, [prediction, decoded])

model.compile(optimizer='adam',
   loss=['categorical_crossentropy', 'mse']
   metrics=['accuracy'])

model.fit_generator(kg, epochs=NUM_EPOCHS, validation_data=kg.val_data, callbacks=callbacks)
```




Epoch 0


Epoch 0


Epoch 0


Epoch 0


Epoch 0

Training worked but model accuracy is low: 79.6% and it's big: 44M parameters

Let's use Keras Tuner to find a more accurate model and smaller model!

Keras Tuner TunableResNet

Use tunable ResNet as base

```
def model_fn():
  base_model = TunableResNet(input_shape=(100, 100, 3))
```

Customize it to support our multi-head output

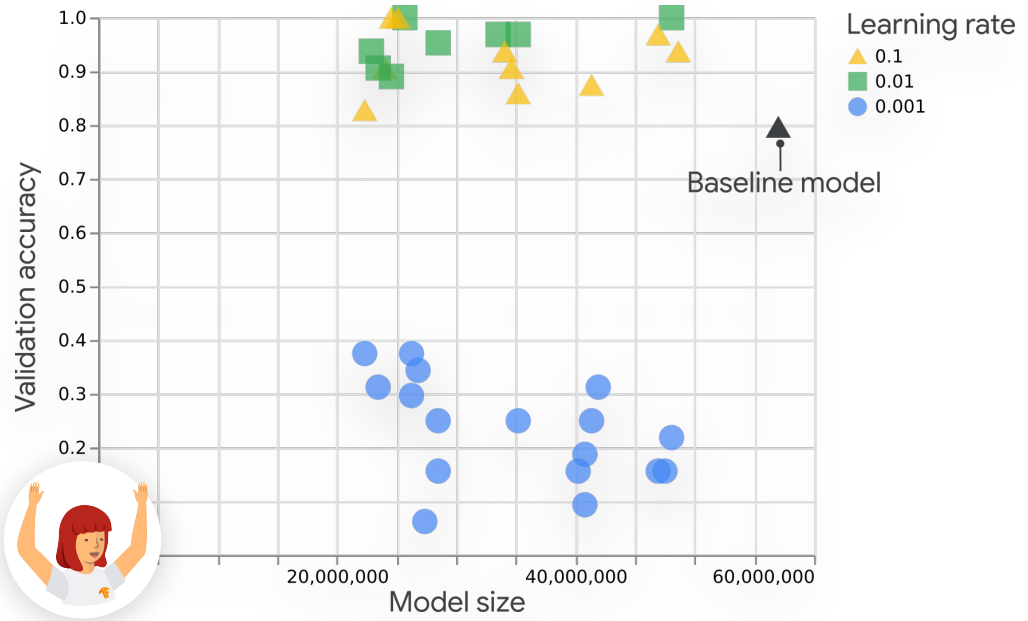
```
  x = base_model.output
  x = GlobalAveragePooling2D()(x)
  y = Dense(100 * 100 *3, activation='sigmoid')(x)
  decoded = Reshape((100, 100, 3), name='reconstruction')(y)
  prediction = Dense(generator.num_classes, activation='softmax')(x)
  model = Model(base_model.input, [prediction, decoded])

  model.compile(optimizer=optimizers.Adam(LEARNING_RATE),
       loss=['categorical_crossentropy', 'mse']
       metrics=['accuracy'])
  return model
```

Init hypertuner

```
tuner = Tuner(model_fn, 'val_accuracy' epoch_budget=500, max_epochs=5)
```

Tuning

```
tuner.search(tfg,validation_data=validation_data)
```



Yay! Keras Tuner found a better model with 100% accuracy (+20%) and only 24M parameters (-45%)

Dataset is small so there is a possibility of overfit despite using augmented icons in training

Ecosystem integration

TensorBoard

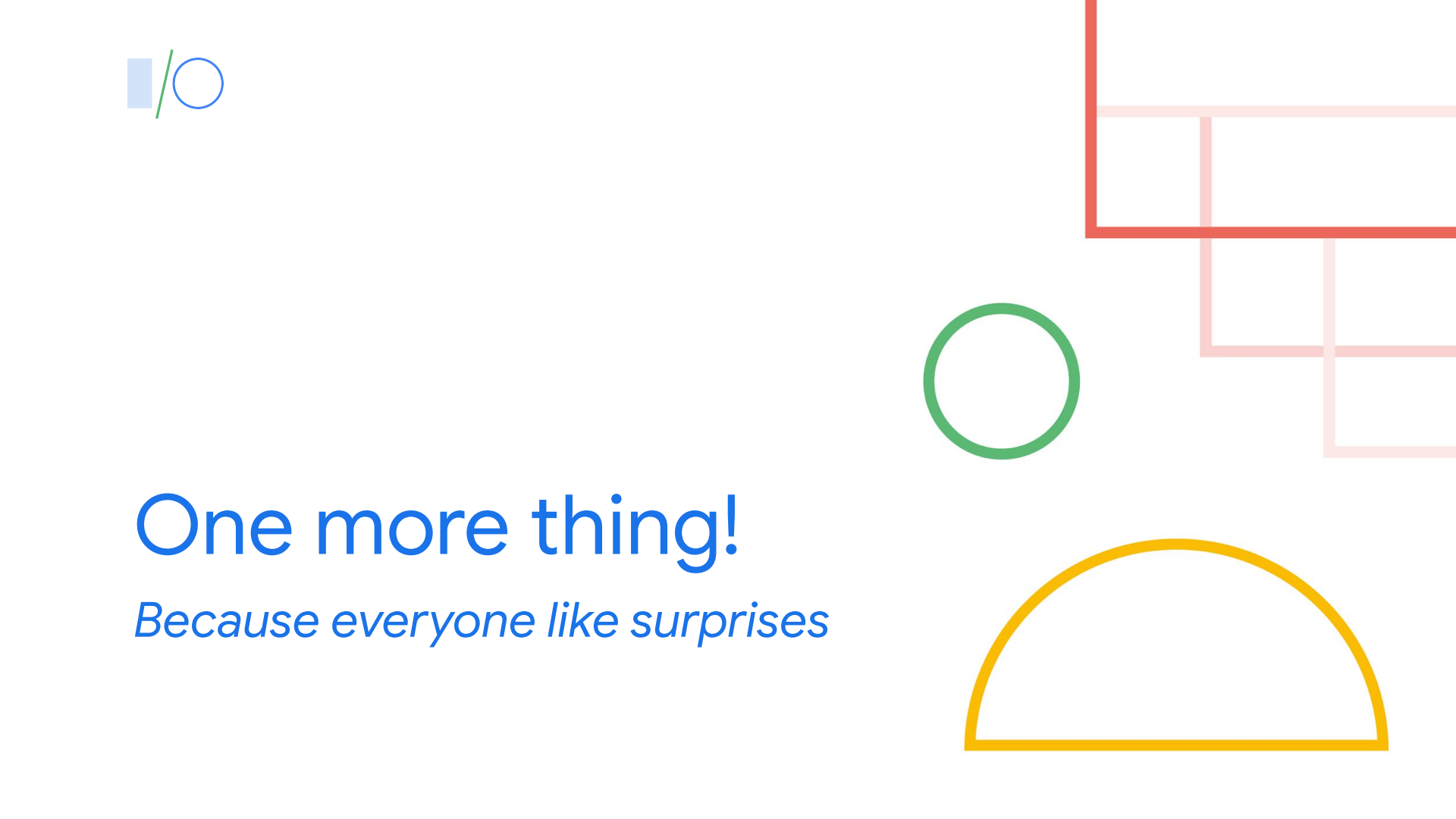
Colab

BigQuery

Command line

...

More to come!

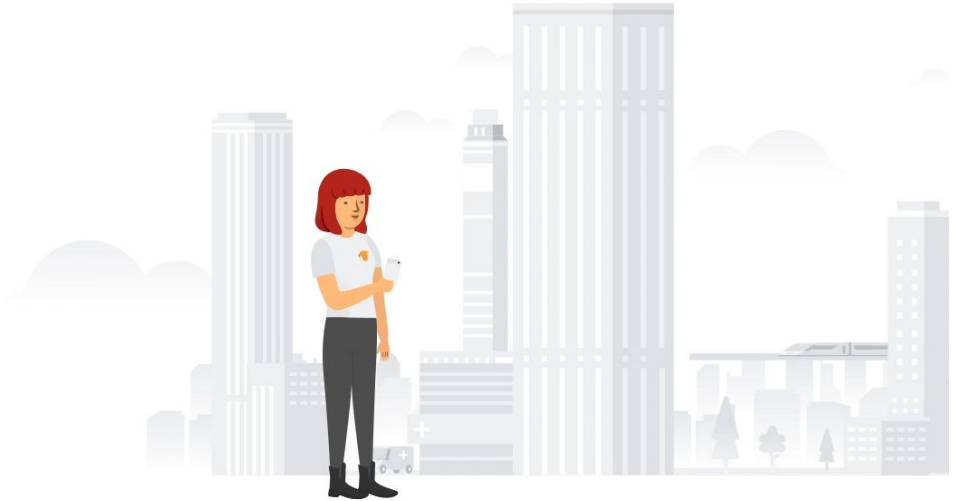


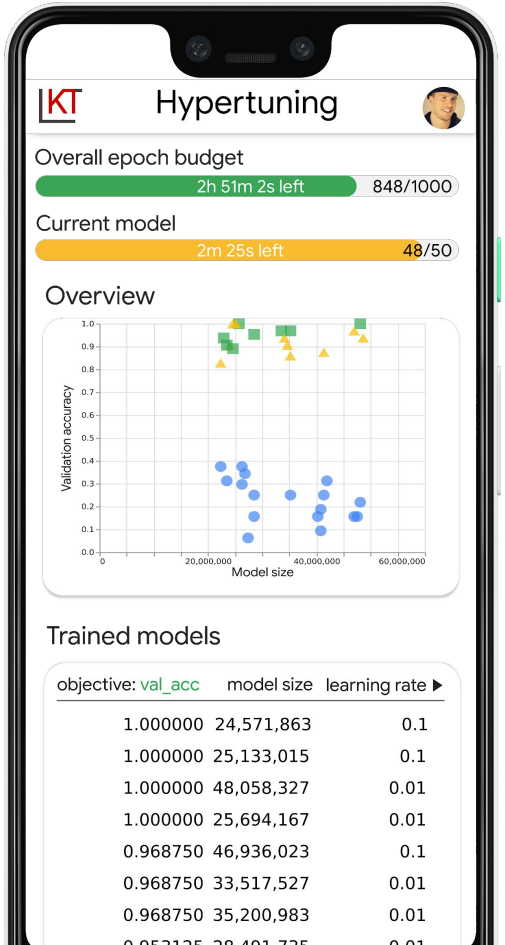
One more thing!

Because everyone like surprises

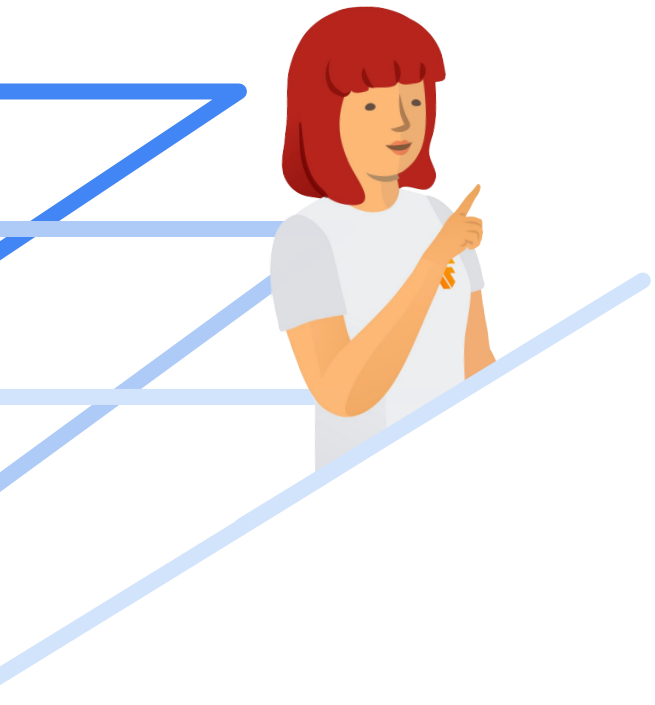
Online dashboard

Monitoring on the go!





Alpha version -- Non final UI



Sign up today for Keras Tuner alpha to help us making it awesome!

https://g.co/research/kerastunereap