Build TensorFlow input pipelines tf.data

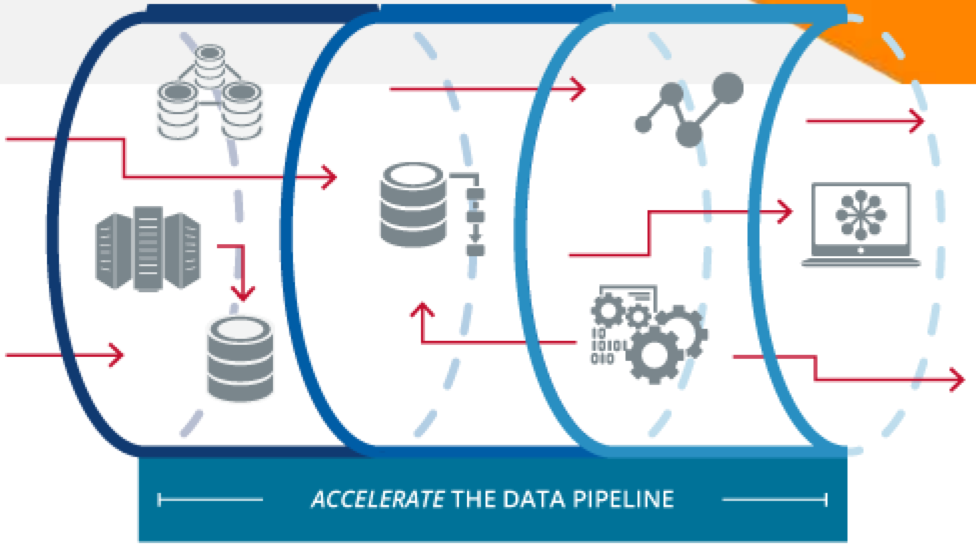
Alireza Akhavanpour

Thursday - 2020 02 April

Akhavanpour.ir
CLASS.VISION

tf.data
Build TensorFlow input pipelines

TensorFlow



ImageDataGenerator (Keras) VS tf.data

The **keras.preprocessing** method is convenient, but has three downsides:

- It's **slow**.
- It **lacks fine-grained control**.
- It is not well **integrated with** the rest of **TensorFlow**.

https://www.tensorflow.org/tutorials/load_data/images#load_using_keraspreprocessing

ImageDataGenerator (Keras) VS tf.data

```
# `keras.preprocessing`
timeit(train_data_gen)
```

```
..............................................................................
1000 batches: 79.98093152046204 s
400.09537 Images/s
```

```
# `tf.data`
timeit(train_ds)
```

```
..............................................................................
1000 batches: 5.851868878741455 s
5468.33853 Images/s
```

https://www.tensorflow.org/tutorials/load_data/images#load_using_keraspreprocessing

tf.data

Input pipelines for TensorFlow should be:

- **Fast...** to keep up with GPUs and TPUs
- **Flexible...** to handle diverse data sources and use cases
- **Easy to use...** to democratize machine learning

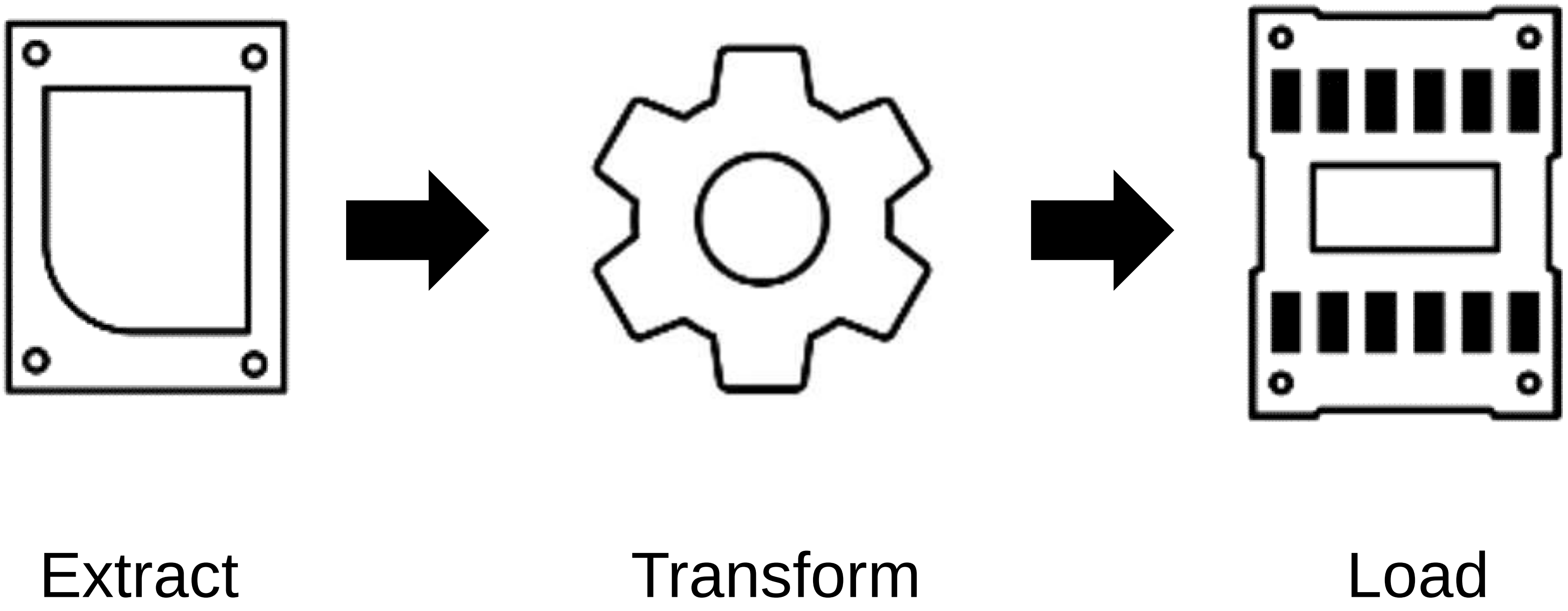
tf.data

tf.data is

~~Input pipelines for TensorFlow should be:~~

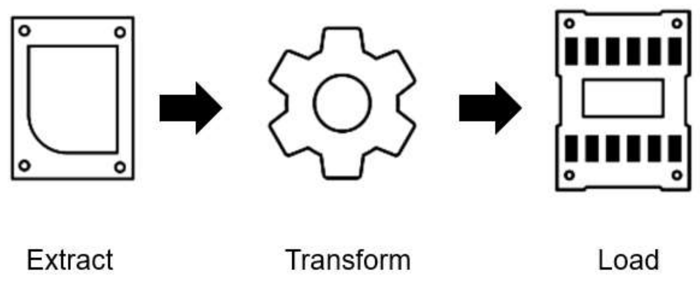
- **Fast...** to keep up with GPUs and TPUs
- **Flexible...** to handle diverse data sources and use cases
- **Easy to use...** to democratize machine learning

ETL for Tensorflow

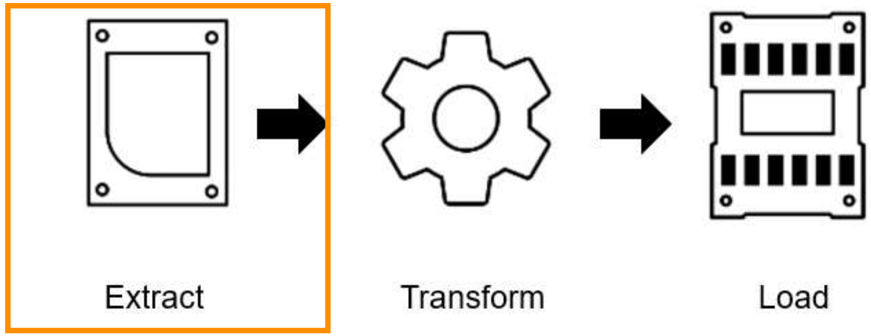


TensorFlow input pipeline

- Extract:
  - read data from memory / storage
  - parse file format
- Transform:
  - text vectorization
  - image transformations
  - video temporal sampling
  - shuffling, batching, ...
- Load:
  - transfer data to the accelerator



Extract Data



ETL: 1-Extract (from_tensor_slices)
From python lists or numpy arrays

- Create a source dataset from your input data.

```
dataset = tf.data.Dataset.from_tensor_slices([1, 2, 3])
for element in dataset:
    print(element)
```

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

ETL: 1-Extract (from_tensor_slices and from_tensors)

From python lists or numpy arrays

- to construct a Dataset from data in **memory**, you can use **tf.data.Dataset.from_tensors()** or **f.data.Dataset.from_tensor_slices()**.
- Create a source dataset from your input data.

```
dataset = tf.data.Dataset.from_tensor_slices([1, 2, 3])
for element in dataset:
    print(element)
```

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

ETL: 1-Extract (from_tensor_slices and from_tensors)

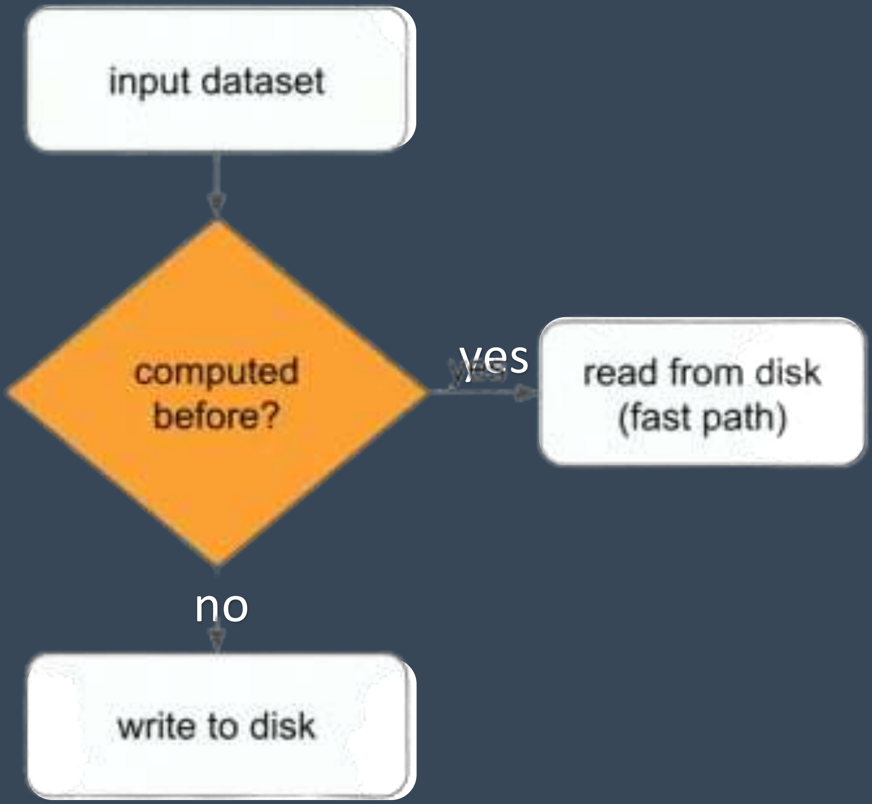
From python lists or numpy arrays

What is the difference between Dataset.from_tensors and Dataset.from_tensor_slices?

- https://stackoverflow.com/questions/49579684/what-is-the-difference-between-dataset-from-tensors-and-dataset-from-tensor-slic

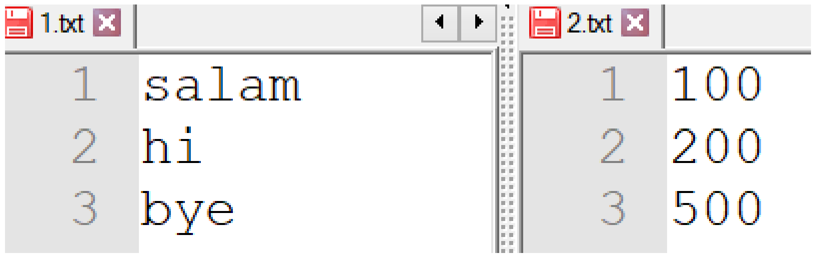
Improve single host performance

Reuse computation



ETL: 1-Extract (TextLineDataset)

- process lines from files



ETL: 1-Extract (TextLineDataset)

- process lines from files

```
dataset = tf.data.TextLineDataset(["1.txt", "2.txt"])
for element in dataset:
    print(element)
```

```
tf.Tensor(b'\xef\xbb\xbfsalam', shape=(), dtype=string)
tf.Tensor(b'hi', shape=(), dtype=string)
tf.Tensor(b'bye', shape=(), dtype=string)
tf.Tensor(b'100', shape=(), dtype=string)
tf.Tensor(b'200', shape=(), dtype=string)
tf.Tensor(b'500', shape=(), dtype=string)
```

ETL: 1-Extract (TFRecordDataset)

To process records written in the TFRecord format, use TFRecordDataset:

```
dataset = tf.data.TFRecordDataset(["file1.tfrecords", "file2.tfrecords"])
```

ETL: 1-Extract (TFRecordDataset)

```
#https://storage.googleapis.com/download.tensorflow.org/data/fs

dataset = tf.data.TFRecordDataset(filenames = ["fsns.tfrec"])
```

```
raw_example = next(iter(dataset))
parsed = tf.train.Example.FromString(raw_example.numpy())
parsed.features.feature['image/text']
```

```
bytes_list {
  value: "Rue Perreyon"
}
```

ETL: 1-Extract (from_generator)

From python generator

```python
def fib(n):
    a, b = 0, 1
    for _ in range(n):
        yield a
        a, b = b, a + b
```

```python
for e in fib(4):
    print (e)
```

```
0
1
1
2
```

ETL: 1-Extract (from_generator)

From python generator

```
dataset = tf.data.Dataset.from_generator(
    fib, args=[8], output_types=tf.int32, output_shapes = (), )
for element in dataset:
    print(element)
```

```
tf.Tensor(0, shape=(), dtype=int32)
tf.Tensor(1, shape=(), dtype=int32)
tf.Tensor(1, shape=(), dtype=int32)
tf.Tensor(2, shape=(), dtype=int32)
tf.Tensor(3, shape=(), dtype=int32)
tf.Tensor(5, shape=(), dtype=int32)
tf.Tensor(8, shape=(), dtype=int32)
tf.Tensor(13, shape=(), dtype=int32)
```

ETL: 1-Extract (from_generator)

From python generator

```python
dataset = tf.data.Dataset.from_generator(
    fib, args=[8], output_types=tf.int32, output_shapes = (), )
for element in dataset:
    print(element)
```

```
tf.Tenso                    int32)
tf.Tenso                    int32)
tf.Tenso                    int32)
tf.Tenso                    int32)
tf.Tensor(3, shape=(), dtype=int32)
tf.Tensor(5, shape=(), dtype=int32)
tf.Tensor(8, shape=(), dtype=int32)
tf.Tensor(13, shape=(), dtype=int32)
```

Callable!

ETL: 1-Extract (from_generator)

From python generator

```python
dataset = tf.data.Dataset.from_generator(
    fib, args=[8], output_types=tf.int32, output_shapes = (), )
for element in dataset:
    print(element)
```

```python
def fib(n):
    a, b = 0, 1
    for _ in range(n):
        yield a
        a, b = b, a + b
```

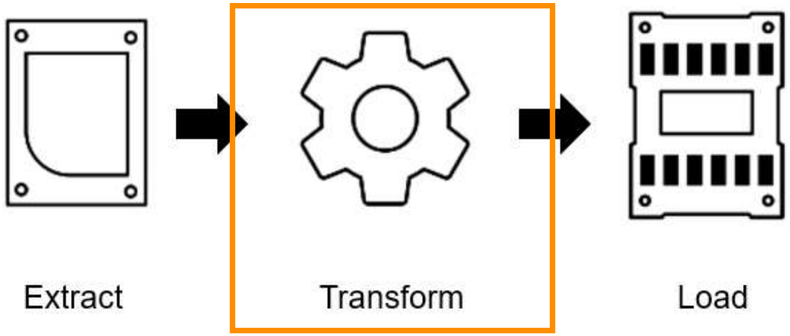
```
tf          int3
tf          int3
tf          int3
tf          int3
tf          int32)
tf          int32)
tf          int32)
tf.Tensor(13, shape=(), dtype=int32)
```

Optional arguments, if necessary!

ETL: 1-Extract
More…

- **tf.data.FixedLengthRecordDataset**
- https://www.tensorflow.org/api_docs/python/tf/data/FixedLengthRecordDataset

Transformations



ETL: 2-Transformations

```
dataset = tf.data.Dataset.from_tensor_slices([1, 2, 3])

dataset = dataset.map(lambda x: x*2)
for e in dataset:
    print (e)
```

```
tf.Tensor(2, shape=(), dtype=int32)
tf.Tensor(4, shape=(), dtype=int32)
tf.Tensor(6, shape=(), dtype=int32)
```

✓ See the documentation for tf.data.Dataset for a complete list of transformations.See the documentation for tf.data.Dataset for a complete list of transformations.

transformations

- **per-element** transformations
  - Dataset.map()
- **multi-element** transformations
  - Dataset.batch()

Batching dataset elements

```
dataset = tf.data.Dataset.range(100)
batched_dataset = dataset.batch(5)

for batch in batched_dataset.take(2):
    print(batch.numpy())
```

?

Batching dataset elements

```
dataset = tf.data.Dataset.range(100)
batched_dataset = dataset.batch(5)

for batch in batched_dataset.take(2):
    print(batch.numpy())
```

```
[0 1 2 3 4]
[5 6 7 8 9]
```

Batching dataset elements

```
inc_dataset = tf.data.Dataset.range(100)
dec_dataset = tf.data.Dataset.range(0, -100, -1)
dataset = tf.data.Dataset.zip((inc_dataset, dec_dataset))
batched_dataset = dataset.batch(4)

for batch in batched_dataset.take(4):
    print([arr.numpy() for arr in batch])
```

?

Batching dataset elements

```
inc_dataset = tf.data.Dataset.range(100)
dec_dataset = tf.data.Dataset.range(0, -100, -1)
dataset = tf.data.Dataset.zip((inc_dataset, dec_dataset))
batched_dataset = dataset.batch(4)

for batch in batched_dataset.take(4):
    print([arr.numpy() for arr in batch])
```

```
[array([0, 1, 2, 3], dtype=int64), array([ 0, -1, -2, -3], dtype=int64)]
[array([4, 5, 6, 7], dtype=int64), array([-4, -5, -6, -7], dtype=int64)]
[array([ 8,  9, 10, 11], dtype=int64), array([ -8,  -9, -10, -11], dtype
=int64)]
[array([12, 13, 14, 15], dtype=int64), array([-12, -13, -14, -15], dtype
=int64)]
```

Batching dataset elements

```
dataset = tf.data.Dataset.from_tensor_slices(
    ['Ali', 'Hassan', 'Hanieh', 'Sara', 'Omid'])
```

```
dataset = dataset.batch(2)
```

```
for e in dataset:
    print (e.numpy())
```

?

Batching dataset elements

```
dataset = tf.data.Dataset.from_tensor_slices(
    ['Ali', 'Hassan', 'Hanieh', 'Sara', 'Omid'])
```

```
dataset = dataset.batch(2)
```

```
for e in dataset:
    print (e.numpy())
```

```
[b'Ali' b'Hassan']
[b'Hanieh' b'Sara']
[b'Omid']
```

Batching dataset elements

```
dataset = tf.data.Dataset.from_tensor_slices(
    ['Ali', 'Hassan', 'Hanieh', 'Sara', 'Omid'])
```

```
dataset = dataset.batch(2, drop_remainder=True)
```

```
for e in dataset:
    print (e.numpy())
```

```
[b'Ali' b'Hassan']
[b'Hanieh' b'Sara']
```

Repeat

```
dataset = tf.data.Dataset.from_tensor_slices(
    ['Ali', 'Hassan', 'Hanieh', 'Sara', 'Omid'])
dataset = dataset.repeat(3)
dataset = dataset.batch(2, drop_remainder=True)
for e in dataset:
    print (e.numpy())
```

?

Repeat

```
dataset = tf.data.Dataset.from_tensor_slices(
    ['Ali', 'Hassan', 'Hanieh', 'Sara', 'Omid'])
dataset = dataset.repeat(3)
dataset = dataset.batch(2, drop_remainder=True)
for e in dataset:
    print (e.numpy())
```

```
[b'Ali' b'Hassan']
[b'Hanieh' b'Sara']
[b'Omid' b'Ali']
[b'Hassan' b'Hanieh']
[b'Sara' b'Omid']
[b'Ali' b'Hassan']
[b'Hanieh' b'Sara']
```

Repeat

```
dataset = tf.data.Dataset.from_tensor_slices(
    ['Ali', 'Hassan', 'Hanieh', 'Sara', 'Omid'])
dataset = dataset.repeat(3)
dataset = dataset.batch(2, drop_remainder=True)
for e in dataset:
    print (e.numpy())
```

```
[b'Ali' b'Hassan']
[b'Hanieh' b'Sara']
[b'Omid' b'Ali']
[b'Hassan' b'Hanieh']
[b'Sara' b'Omid']
[b'Ali' b'Hassan']
[b'Hanieh' b'Sara']
```

Repeat

```
dataset = tf.data.Dataset.from_tensor_slices(
    ['Ali', 'Hassan', 'Hanieh', 'Sara', 'Omid'])
dataset = dataset.batch(2, drop_remainder=True)
dataset = dataset.repeat(3)
for e in dataset:
    print (e.numpy())
```

?

Repeat

```
dataset = tf.data.Dataset.from_tensor_slices(
    ['Ali', 'Hassan', 'Hanieh', 'Sara', 'Omid'])
dataset = dataset.batch(2, drop_remainder=True)
dataset = dataset.repeat(3)
for e in dataset:
    print (e.numpy())
```

```
[b'Ali' b'Hassan']
[b'Hanieh' b'Sara']
[b'Ali' b'Hassan']
[b'Hanieh' b'Sara']
[b'Ali' b'Hassan']
[b'Hanieh' b'Sara']
```

Repeat

```
dataset = tf.data      taset.fr      tensor_slices(
    ['Ali', 'H      'Han      ara', 'Omid'])
dataset = datas      h(2      emainder=True)
dataset = datase
for e in dataset:
    print (e.numpy
```

```
[b'Ali' b'Hassan']
[b'Hanieh' b'Sara'
[b'Ali' b'Hassan'
[b'Hanieh' b'Sar
[b'Ali' b'Hassa
[b'Hanieh' b'S
```

Shuffle

```
dataset = tf.data.Dataset.from_tensor_slices(
    ['Ali', 'Hassan', 'Hanieh', 'Sara', 'Omid'])
dataset = dataset.shuffle(5)
dataset = dataset.batch(2, drop_remainder=True)
for e in dataset:
    print (e.numpy())
```

```
[b'Ali' b'Hanieh']
[b'Hassan' b'Omid']
```

Shuffle

```
dataset = tf.data.Dataset.from_tensor_slices(
    ['a', 'b', 'c', 'd', 'e', 'f'])
dataset = dataset.shuffle(6)
dataset = dataset.repeat(2)
dataset = dataset.batch(2, drop_remainder=True)
for e in dataset:
    print (e.numpy())
```

?

Shuffle

```
dataset = tf.data.Dataset.from_tensor_slices(
    ['a', 'b', 'c', 'd', 'e', 'f'])
dataset = dataset.shuffle(6)
dataset = dataset.repeat(2)
dataset = dataset.batch(2, drop_remainder=True)
for e in dataset:
    print (e.numpy())
```

```
[b'f' b'a']
[b'b' b'd']
[b'c' b'e']
[b'a' b'f']
[b'c' b'd']
[b'e' b'b']
```

Shuffle

```
dataset = tf.data.Dataset.from_tensor_slices(
    ['a', 'b', 'c', 'd', 'e', 'f'])
dataset = dataset.shuffle(6)
dataset = dataset.repeat(2)
dataset = dataset.batch(2, drop_remainder=True)
for e in dataset:
    print (e.numpy())
```

```
[b'f' b'a']
[b'b' b'd']
[b'c' b'e']
[b'a' b'f']
[b'c' b'd']
[b'e' b'b']
```

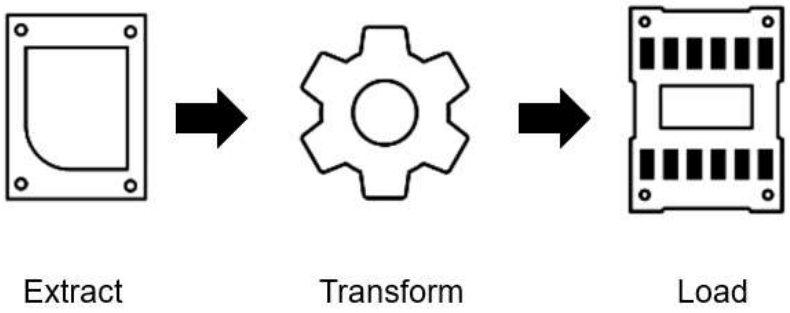
Shuffle

```
dataset = tf.data.Dataset.from_tensor_slices(
    ['a', 'b', 'c', 'd', 'e', 'f'])
dataset = dataset.shuffle(buffer_size=6, reshuffle_each_iteration=False)
dataset = dataset.repeat(2)
dataset = dataset.batch(2, drop_remainder=True)
for e in dataset:
    print (e.numpy())
```

```
[b'c' b'a']
[b'd' b'f']
[b'b' b'e']
[b'c' b'a']
[b'd' b'f']
[b'b' b'e']
```

=

ETL



ETL

E

```
files = tf.data.Dataset.list_files(file_pattern)
dataset = tf.data.TFRecordDataset(files)
```

T

```
dataset = dataset.shuffle(10000)
dataset = dataset.repeat(NUM_EPOCHS)
dataset = dataset.map(lambda x: tf.parse_single_example(x, features))
dataset = dataset.batch(BATCH_SIZE)
```

L

```
iterator = dataset.make_one_shot_iterator()
features = iterator.get_next()
```

ETL

E

```
files = tf.data.Dataset.list_files(file_pattern)
dataset = tf.data.TFRecordDataset(files)
```

T

```
dataset = dataset.shuffle(10000)
dataset = dataset.repeat(NUM_EPOCHS)
dataset = dataset.map(lambda x: tf.parse_single_example(x, features))
dataset = dataset.batch(BATCH_SIZE)
```

L

```
iterator = dataset.make_one_shot_iterator()
features = iterator.get_next()
```

ETL

E

```
files = tf.data.Dataset.list_files(file_pattern)
dataset = tf.data.TFRecordDataset(files)
```

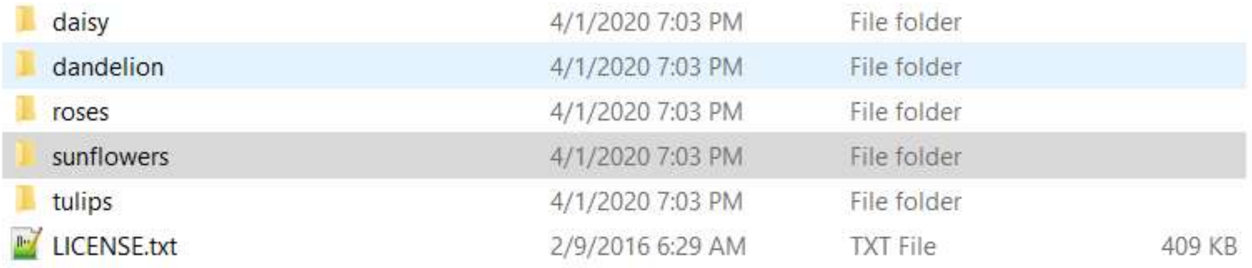
T

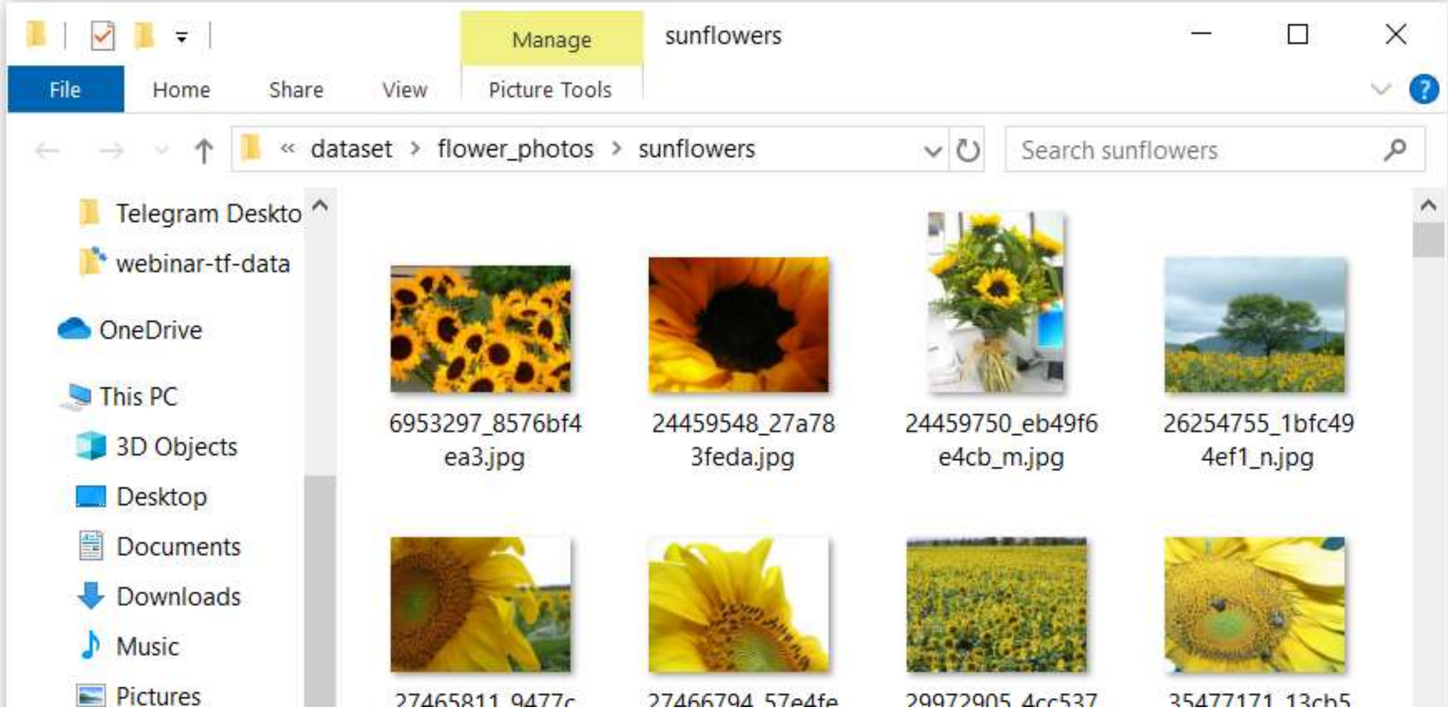
```
dataset = dataset.shuffle(10000)
dataset = dataset.repeat(NUM_EPOCHS)
dataset = dataset.map(lambda x: tf.parse_single_example(x, features))
dataset = dataset.batch(BATCH_SIZE)
```

L

```
iterator = dataset.make_one_shot_iterator()
features = iterator.get_next()
```

Load image with tf.data





Load image with tf.data

```
flowers_root = "D:/dataset/flower_photos"
```

```
list_ds = tf.data.Dataset.list_files(str(flowers_root/'*/*'))
```

```
for e in list_ds.take(3):
    print(e)
```

```
tf.Tensor(b'D:\\dataset\\flower_photos\\tulips\\2220085701_896054d263_n.
jpg', shape=(), dtype=string)
tf.Tensor(b'D:\\dataset\\flower_photos\\sunflowers\\4933230247_a0432f01d
a.jpg', shape=(), dtype=string)
tf.Tensor(b'D:\\dataset\\flower_photos\\tulips\\14254839301_ffb19c6445_
n.jpg', shape=(), dtype=string)
```

Load image with tf.data

```
def process_path(file_path):
    #label = tf.strings.split(file_path, '/')[-2]
    label = tf.strings.split(file_path, '\\')[-2]
    return tf.io.read_file(file_path), label
```

```
labeled_ds = list_ds.map(process_path)
```

```
for image, label in labeled_ds.take(3):
    print(label)
```

```
tf.Tensor(b'roses', shape=(), dtype=string)
tf.Tensor(b'roses', shape=(), dtype=string)
tf.Tensor(b'roses', shape=(), dtype=string)
```

Load image with tf.data

```
In [10]: image

Out[10]: <tf.Tensor: shape=(), dtype=string, numpy=b'\xff\xd8\xff\xe0\x00\x10JFI
F\x00\x01\x01\x00\x00\x01\x00\x01\x00\x00\xff\xe2\x0cXICC_PROFILE\x00\x
01\x01\x00\x00\x0cHLino\x02\x10\x00\x00mntrRGB XYZ \x07\xce\x00\x02\x0
0\t\x00\x06\x001\x00\x00acspMSFT\x00\x00\x00\x00IEC sRGB\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xf6\xd6\x00\x01\x00\x00\x00\x
00\xd3-HP  \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
```

so, what is this?

Load image with tf.data

TensorFlow In

- tf.image
- tf.io
  - Overview
  - decode_and_crop_jpeg
  - decode_base64
  - decode_bmp
  - decode_compressed
  - decode_csv
  - decode_gif
  - decode_image
  - decode_jpeg
  - decode_json_example
  - decode_png
  - decode_proto

tf.io.decode_jpeg

See Stable   See Nightly

TensorFlow 1 version

Decode a JPEG-encoded image to a uint8 tensor.

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

Load image with tf.data

```
def process_path(file_path):
    #label = tf.strings.split(file_path, '/')[-2]
    label = tf.strings.split(file_path, '\\')[-2]
    image = tf.io.read_file(file_path)
    image = tf.image.decode_jpeg(image)
    return image, label
```

```
labeled_ds = list_ds.map(process_path)
for image, label in labeled_ds.take(3):
    print(image.shape)
    print(label)
```

```
(240, 180, 3)
tf.Tensor(b'tulips', shape=(), dtype=string)
(240, 320, 3)
tf.Tensor(b'daisy', shape=(), dtype=string)
(213, 320, 3)
tf.Tensor(b'roses', shape=(), dtype=string)
```

Load image with tf.data

```
flowers_root = "D:/dataset/flower_photos"
list_ds = tf.data.Dataset.list_files(str(flowers_root+'*/*'))
labeled_ds = list_ds.map(process_path)
batched_ds = labeled_ds.batch(32)
for image, label in batched_ds.take(3):
    print(image.shape)
```

?

Load image with tf.data

```
flowers_root = "D:/dataset/flower_photos"
list_ds = tf.data.Dataset.list_files(str(flowers_root+'*/*'))
labeled_ds = list_ds.map(process_path)
batched_ds = labeled_ds.batch(32)
for image, label in batched_ds.take(3):
    print(image.shape)
```

```
InvalidArgumentError: Cannot add tensor to the batch: numb
er of elements does not match. Shapes are: [tensor]: [333,
500,3], [batch]: [240,240,3]
```

Load image with tf.data

✓ **Add resize**

```
# Reads an image from a file, decodes it into a dense tensor
# to a fixed shape.
def parse_image(filename):
    parts = tf.strings.split(filename, '\\') # or replace \\
    label = parts[-2]

    image = tf.io.read_file(filename)
    image = tf.image.decode_jpeg(image)
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = tf.image.resize(image, [128, 128])
    return image, label
```

Load image with tf.data

```
flowers_root = "D:/dataset/flower_photos"
list_ds = tf.data.Dataset.list_files(str(flowers_root+'*/*'))
labeled_ds = list_ds.map(parse_image)
batched_ds = labeled_ds.batch(32)
for image, label in batched_ds.take(3):
    print(image.shape)
```

?

Load image with tf.data

```
flowers_root = "D:/dataset/flower_photos"
list_ds = tf.data.Dataset.list_files(str(flowers_root+'*/*'))
labeled_ds = list_ds.map(parse_image)
batched_ds = labeled_ds.batch(32)
for image, label in batched_ds.take(3):
    print(image.shape)
```

```
(32, 128, 128, 3)
(32, 128, 128, 3)
(32, 128, 128, 3)
```

Load image with tf.data
(Applying arbitrary Python logic)

tf.py_function

- For performance reasons, use TensorFlow operations for preprocessing your data whenever possible.
- It is sometimes useful to call external Python libraries when parsing your input data.

Load image with tf.data
(Applying arbitrary Python logic)

```python
import scipy.ndimage as ndimage
import numpy as np

def random_rotate_image(image):
    image = ndimage.rotate(image, np.random.uniform(-30, 30), reshape=False
    return image
```

```python
def tf_random_rotate_image(image, label):
    [image,] = tf.py_function(random_rotate_image, [image], [tf.float32])
    return image, label
```

Load image with tf.data
(Applying arbitrary Python logic)

```python
import scipy.ndimage as ndimage
import numpy as np

def random_rotate_image(image):
    image = ndimage.rotate(image, np.random.uniform(-30, 30), reshape=False
    return image
```

Python function

Load image with tf.data
(Applying arbitrary Python logic)

```
import scipy.ndimage as ndimage
import numpy as np

def random_rotate_image(image):
    image = ndimage.rotate(image, np.random.uniform(-30, 30), reshape=False
    return image
```

Python function

```
def tf_random_rotate_image(image, label):
    [image,] = tf.py_function(random_rotate_image, [image], [tf.float32])
    return image, label
```

tensorflow

Load image with tf.data
(Applying arbitrary Python logic)

★ **Note:** `tensorflow_addons` has a TensorFlow compatible `rotate` in `tensorflow_addons.image.rotate`.

```
import scipy.ndimage as ndimage
import numpy as np

def random_rotate_image(image):
    image = ndimage.rotate(image, np.random.uniform(-30, 30), reshape=False
    return image
```

```
def tf_random_rotate_image(image, label):
    [image,] = tf.py_function(random_rotate_image, [image], [tf.float32])
    return image, label
```

Load image with tf.data (data_augmentation)

- flipped = tf.image.flip_left_right(image)
- grayscaled = tf.image.rgb_to_grayscale(image)
- saturated = tf.image.adjust_saturation(image, 3)
- bright = tf.image.adjust_brightness(image, 0.4)
- rotated = tf.image.rot90(image)
- cropped = tf.image.central_crop(image, central_fraction=0.5)

https://www.tensorflow.org/tutorials/images/data_augmentation

Load image with tf.data
(cast and normalize/standardization)

- **Cast and normalize the image to [0,1]**
  - image = tf.image.convert_image_dtype(image, tf.float32)
- **Only cast**
  - img = tf.cast(img, tf.float32)

Image classification – Version1

✓4-transfer_learning-VGG

tf.data
Performance

Recap...

```
import tensorflow as tf


dataset = tf.data.Dataset.list_files(PATH_GLOB)
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(get_bytes_and_label)
dataset = dataset.map(process_image)
dataset = dataset.batch(batch_size=32)
```

Recap...

```
import tensorflow as tf

dataset = tf.data.Dataset.list_files(PATH_GLOB)
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(get_bytes_and_label)
dataset = dataset.map(process_image)
dataset = dataset.batch(batch_size=32)
```

```
def process_image(image_bytes, label):
  image = tf.io.decode_jpeg(image_bytes)
  image = tf.image.resize(image, resolution)
  image.set_shape(input_shape)
  image = image / 255.0 - 0.5

  image = tf.image.random_flip_left_right(image)
  image = tf.image.random_flip_up_down(image)
  image += tf.random.normal(
      image.shape, mean=0, stddev=0.1)

  return image, tf.cast(label, tf.float32)
```

Prefetching

- Prefetching overlaps the preprocessing and model execution of a training step. While the model is executing training step s, the input pipeline is reading the data for step s+1. Doing so reduces the step time to the maximum (as opposed to the sum) of the training and the time it takes to extract the data.

Pipeline with prefetch

```
import tensorflow as tf

dataset = tf.data.Dataset.list_files(PATH_GLOB)
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(get_bytes_and_label)
dataset = dataset.map(process_image)
dataset = dataset.batch(batch_size=32)
dataset = dataset.prefetch(buffer_size=X) # Pipelining
```

Parallelize transformation

```
import tensorflow as tf


dataset = tf.data.Dataset.list_files(PATH_GLOB)
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(get_bytes_and_label)
dataset = dataset.map(process_image, num_parallel_calls=Y)  # Parallelize transformation
dataset = dataset.batch(batch_size=32)
dataset = dataset.prefetch(buffer_size=X)
```

Parallelize transformation

```
import tensorflow as tf

dataset = tf.data.Dataset.list_files(PATH_GLOB)
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(get_bytes_and_label, num_parallel_calls=Z)  # Parallelize extractio
dataset = dataset.map(process_image, num_parallel_calls=Y)
dataset = dataset.batch(batch_size=32)
dataset = dataset.prefetch(buffer_size=X)
```

Parallelize transformation (Autotune)

```
import tensorflow as tf
AUTOTUNE = tf.data.experimental.AUTOTUNE


dataset = tf.data.Dataset.list_files(PATH_GLOB)
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(get_bytes_and_label, num_parallel_calls=AUTOTUNE)
dataset = dataset.map(process_image, num_parallel_calls=AUTOTUNE)
dataset = dataset.batch(batch_size=32)
dataset = dataset.prefetch(buffer_size=AUTOTUNE)
```

Parallelize transformation (Autotune)

```
import tensorflow as tf
AUTOTUNE = tf.data.experimental.AUTOTUNE


dataset = tf.data.Dataset.list_files(PATH_GLOB)
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(get_bytes_and_label, num_parallel_calls=AUTOTUNE)
dataset = dataset.map(process_image, num_parallel_calls=AUTOTUNE)
dataset = dataset.batch(batch_size=32)
dataset = dataset.prefetch(buffer_size=AUTOTUNE)
```

Parallelize transformation (Autotune)

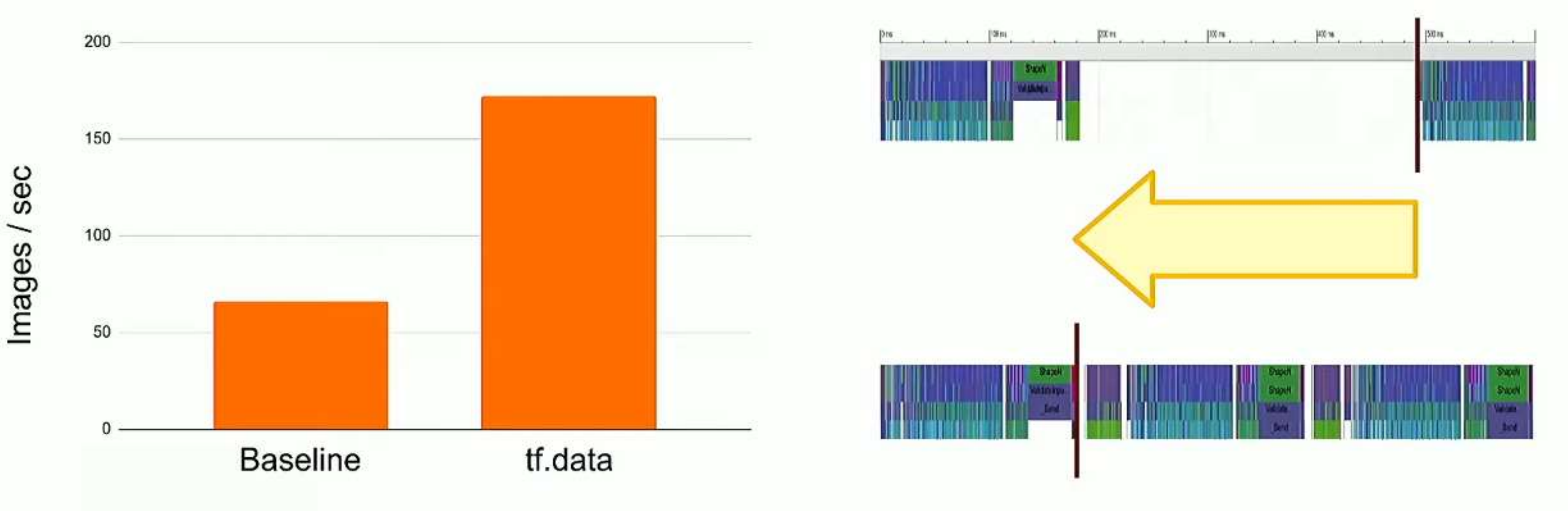
```
import tensorflow as tf
AUTOTUNE = tf.data.experimental.AUTOTUNE


dataset = tf.data.Dataset.list_files(PATH_GLOB)
dataset = dataset.shuffle(NUM_TOTAL_IMAGES)
dataset = dataset.map(get_bytes_and_label, num_parallel_calls=AUTOTUNE)
dataset = dataset.map(process_image, num_parallel_calls=AUTOTUNE)
dataset = dataset.batch(batch_size=32)
dataset = dataset.prefetch(buffer_size=AUTOTUNE)
```

Best practice summary

- Use the **prefetch** transformation to overlap the work of a producer and consumer.
- **Parallelize** the data reading transformation using the interleave transformation.
- **Parallelize** the map transformation by setting the **num_parallel_calls** argument.
- Use the **cache** transformation to cache data in memory during the first epoch
- **Vectorize** user-defined functions passed in to the map transformation
- Reduce memory usage when applying the interleave, prefetch, and shuffle transformations.

Performance..



The tf.function decorator

When you annotate a function with **tf.function**, you can still call it like any other function. But it will be compiled into a graph, which means you get the benefits of faster execution, running on GPU or TPU, or exporting to SavedModel.

https://www.tensorflow.org/guide/function

Using tf.function()

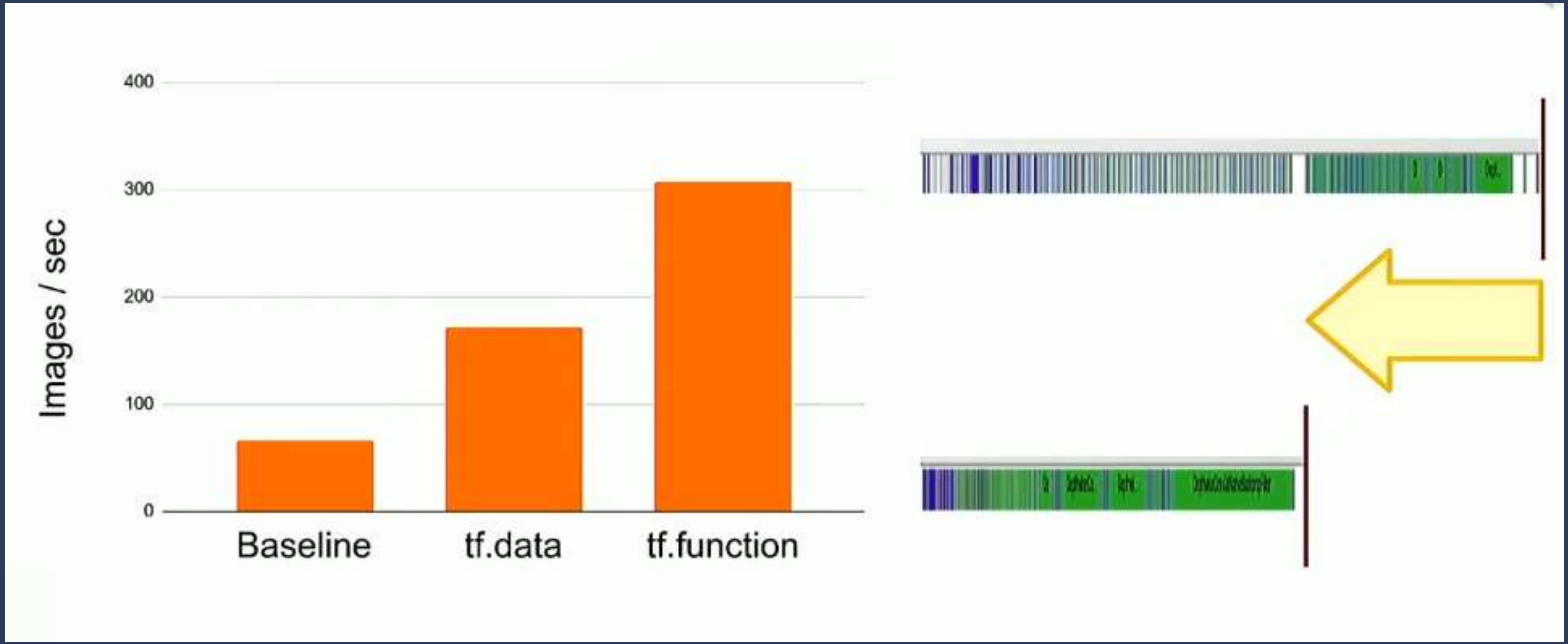
```
@tf.function
def step(features, labels):
  with tf.GradientTape() as tape:
    logits = model(features, training=True)
    loss = loss_fn(labels, logits)

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

for features, labels in data:
  loss = replica_step(features, labels)
```

https://www.tensorflow.org/guide/function

Performance..



XLA

TensorFlow > Resources > XLA

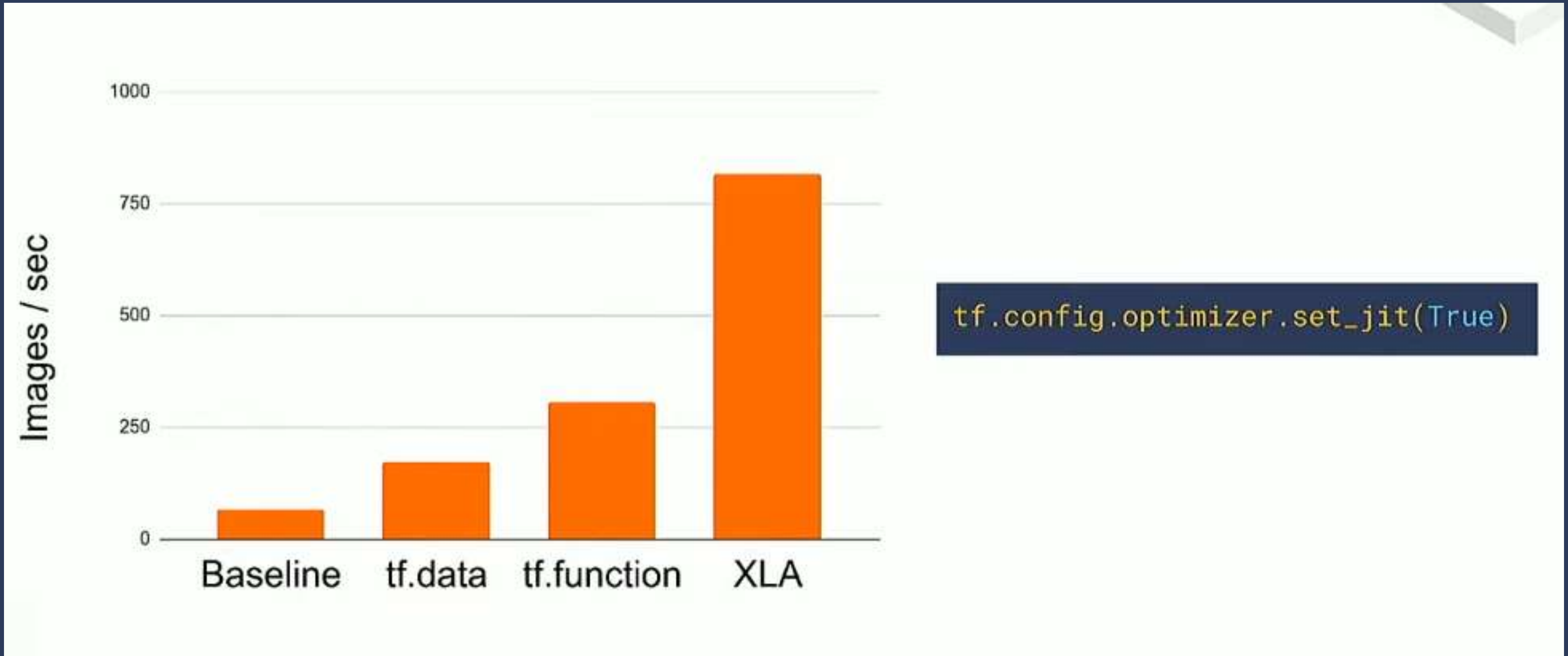
XLA: Optimizing Compiler for Machine Learning

XLA (Accelerated Linear Algebra) is a domain-specific compiler for linear algebra that can accelerate TensorFlow models with potentially no source code changes.

The results are improvements in speed and memory usage: most internal benchmarks run ~1.15x faster after XLA is enabled. The dataset below is evaluated on a single NVidia V100 GPU:

https://www.tensorflow.org/xla

Performance..



Performance..

The Keras mixed precision API is available in **TensorFlow 2.1.**

Overview

Mixed precision is the use of both 16-bit and 32-bit floating-point types in a model during training to make it run faster and use less memory. By keeping certain parts of the model in the 32-bit types for numeric stability, the model will have a lower step time and train equally as well in terms of the evaluation metrics such as accuracy. This guide describes how to use the experimental Keras mixed precision API to speed up your models. Using this API can improve performance by more than 3 times on modern GPUs and 60% on TPUs.

★ **Note:** The Keras mixed precision API is currently experimental and may change.

Performance..

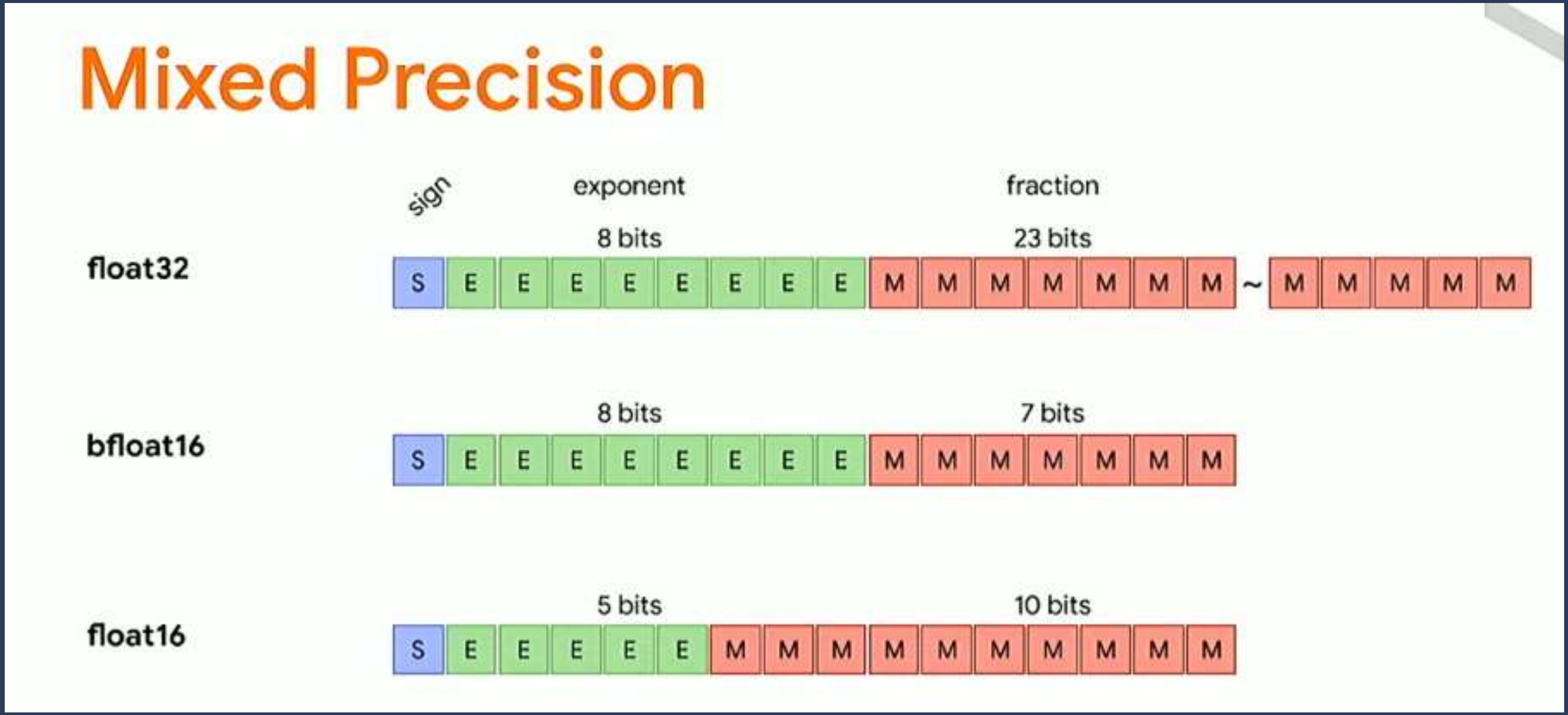
Supported hardware

While mixed precision will run on most hardware, it will only speed up models on recent NVIDIA GPUs and Cloud TPUs. NVIDIA GPUs support using a mix of float16 and float32, while TPUs support a mix of bfloat16 and float32.

Among NVIDIA GPUs, those with compute capability 7.0 or higher will see the greatest performance benefit from mixed precision because they have special hardware units, called Tensor Cores, to accelerate float16 matrix multiplications and convolutions. Older GPUs offer no math performance benefit for using mixed precision, however memory and bandwidth savings can enable some speedups. You can look up the compute capability for your GPU at NVIDIA's CUDA GPU web page. Examples of GPUs that will benefit most from mixed precision include RTX GPUs, the Titan V, and the V100.

https://www.tensorflow.org/guide/keras/mixed_precision

Performance..



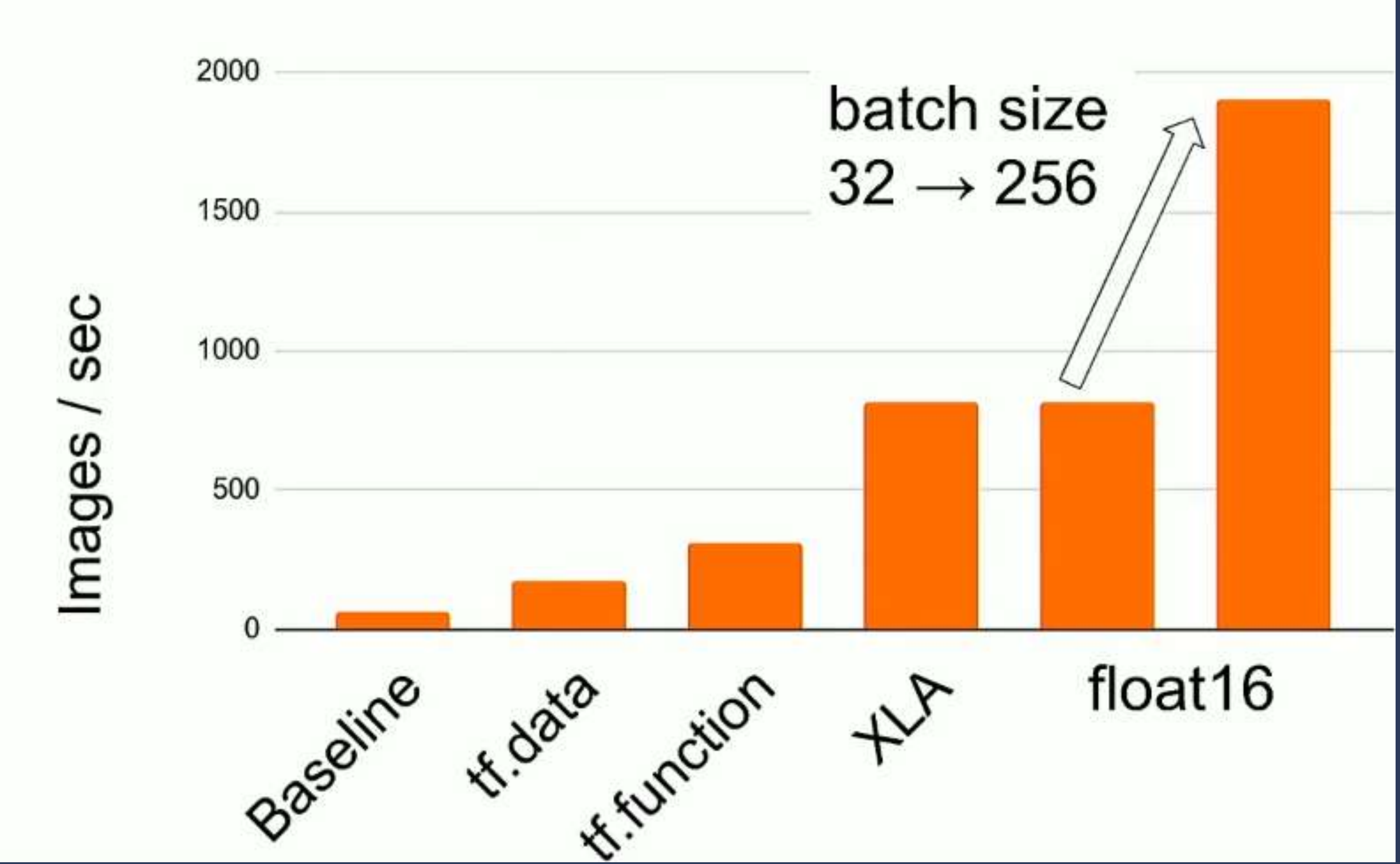
Performance..

```
loss_scale = "dynamic"  # This is the default.
policy = tf.keras.mixed_precision.experimental.Policy(
    "mixed_float16", loss_scale=loss_scale)
tf.keras.mixed_precision.experimental.set_policy(policy)

# Done automatically in Model.fit
optimizer = ...
optimizer = tf.keras.mixed_precision.experimental.LossScaleOptimizer(
    optimizer, loss_scale=loss_scale)

batch_size *= ... # e.g. 8
...
```

Performance..



Recap Performance..

- Use tf.data to build simple and performant data pipelines
- Use tf.function and XLA for improved model performance
- Use mixed precision for even faster training

More Performance..

- Use tf.data to build simple and performant data pipelines
- Use tf.function and XLA for improved model performance
- Use mixed precision for even faster training

More performance…

Improve single host performance

- Prefetch
- Parallel interleave
- Parallel map

https://www.tensorflow.org/guide/data_performance

Improve single host performance

tf.data snapshot

Materialize once, use many

- Experimenting with model architectures
- Hyperparameter tuning

Improve single host performance

```
import tensorflow as tf

def expensive_preprocess(record):
  ...


dataset = tf.data.TFRecordDataset(".../*.tfrecord")
dataset = dataset.map(expensive_preprocess)
dataset = dataset.snapshot("/path/to/snapshot_dir")
dataset = dataset.shuffle(buffer_size=1024)
dataset = dataset.batch(batch_size=32)

dataset = dataset.prefetch()

model = tf.keras.Model(...)
model.fit(dataset)
```

Available in TF 2.3

snapshot transformation

ما را دنبال کنید...

https://t.me/cvision

https://www.aparat.com/cvision

https://www.linkedin.com/company/class-vision/

http://class.vision

http://github.com/alireza-akhavan/

منابع

- https://www.tensorflow.org/api_docs/python/tf/data/Dataset
- https://www.tensorflow.org/guide/data
- https://www.tensorflow.org/tutorials/images/data_augmentation
- https://www.tensorflow.org/tutorials/load_data/images#load_using_keraspreprocessing
- https://www.tensorflow.org/guide/function

منابع

- **Scaling Tensorflow data processing with tf.data (TF Dev Summit '20)**
  - https://www.youtube.com/watch?v=n7byMbl2VUQ
- **tf.data: Fast, flexible, and easy-to-use input pipelines (TensorFlow Dev Summit 2018)**
  - https://www.youtube.com/watch?v=uIcqeP7MFH0
- **Performant, scalable models in TensorFlow 2 with tf.data, tf.function & tf.distribute (TF World '19)**
  - https://www.youtube.com/watch?v=yH1cF7GnoIo