

Building data pipelines with `tf.data`

Sayak Paul | Deep Learning Associate at [PyImageSearch](#)

Machine Learning Weekend, November 2 - 3, 2019

Turkey



Agenda

- Why care about data pipelines?
- What is `tf.data`?
- How it is different from the existing options?
- Building data pipelines with `tf.data`
 - Simpler ones
 - More complex ones (w/ `ImageDataGenerator`)
- Performance comparisons

Acknowledgements

- The entire team at [PyImageSearch](#)
- Picsou Balthazar

Why care about data pipelines?

Why care about data pipelines?

- Machine learning models are data-hungry.

Why care about data pipelines?

- Machine learning models are data-hungry.
- Before data is fed to an ML model it should be:

Why care about data pipelines?

- Machine learning models are data-hungry
- Before data is fed to an ML model it should be:
 - Shuffled

Why care about data pipelines?

- Machine learning models are data-hungry
- Before data is fed to an ML model it should be:
 - Shuffled
 - Batched (Online learning folks, I am sorry!)

Why care about data pipelines?

- Machine learning models are data-hungry
- Before data is fed to an ML model it should be:
 - Shuffled
 - Batched (Online learning folks, I am sorry!)
 - Batches to be available before the current epoch is finished

What is `tf.data`?

`tf.data` is a module by TensorFlow that:

What is `tf.data`?

`tf.data` is a module by TensorFlow that:

- Helps us to build data input pipelines which are:

What is `tf.data`?

`tf.data` is a module by TensorFlow that:

- Helps us to build data input pipelines which are:
 - Scalable

What is `tf.data`?

`tf.data` is a module by TensorFlow that:

- Helps us to build data input pipelines which are:
 - Scalable
 - Simple

What is `tf.data`?

`tf.data` is a module by TensorFlow that:

- Helps us to build data input pipelines which are:
 - Scalable
 - Simple
 - Reusable

What is `tf.data`?

`tf.data` is a module by TensorFlow that:

- Helps us to build data input pipelines
- Allows us to define data input pipelines with a lot of dynaminicity:

What is `tf.data`?

`tf.data` is a module by TensorFlow that:

- Helps us to build data input pipelines
- Allows us to define data input pipelines with a lot of dynaminicity:
 - For example, the data input pipelines for image and text can be completely different.

What is `tf.data`?

`tf.data` is a module by TensorFlow that:

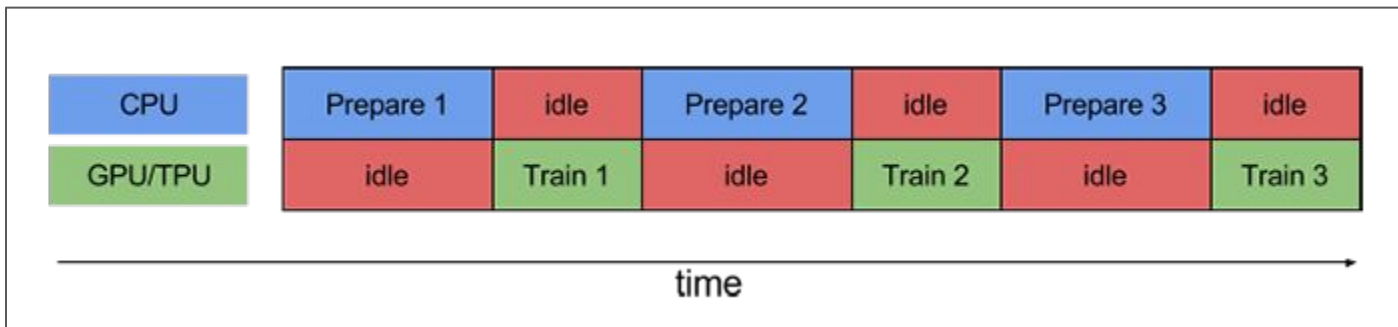
- Helps us to build data input pipelines
- Allows us to define the data input pipelines with a lot of dynaminicity
 - For example, the data input pipelines for image and text can be completely different.
 - `tf.data` gives you *programmable interfaces* to aid your use cases.

Salient features of `tf.data`

- Efficient *pipelining* to reduce any additional time it takes to stream your data to the model

Salient features of `tf.data`

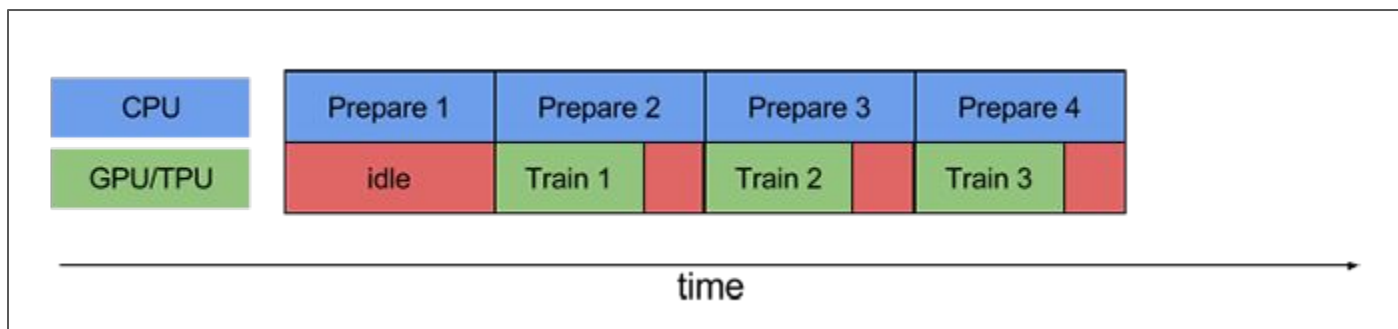
- Efficient *pipelining* to reduce any additional time it takes to stream your data to the model



Processes running without pipelines

Salient features of `tf.data`

- Efficient *pipelining* to reduce any additional time it takes to stream your data to the model



Processes running with pipelines

Salient features of `tf.data`

- Efficient *pipelining* to reduce any additional time it takes to stream your data to the model ← `tf.data.Dataset.prefetch()`

Salient features of `tf.data`

- Efficient *pipelining* to reduce any additional time it takes to stream your data to the model \leftarrow `tf.data.Dataset.prefetch()`
- *Parallelizable* function mapping to your data

Salient features of `tf.data`

- Efficient *pipelining* to reduce any additional time it takes to stream your data to the model ← `tf.data.Dataset.prefetch()`
- *Parallelizable* function mapping to your data ← `tf.data.Dataset.map(preproc_fn)`

Salient features of `tf.data`

- Efficient *pipelining* to reduce any additional time it takes to stream your data to the model ← `tf.data.Dataset.prefetch()`
- *Parallelizable* data transformation ← `tf.data.Dataset.map(preproc_fn)`

TLDR; *parallelization* made easier!

What makes `tf.data` different from others?



What makes `tf.data` different from others?

- `tf.data` can *dynamically* decide the level of parallelism to use (`tf.data.experimental.AUTOTUNE`).

What makes `tf.data` different from others?

- `tf.data` can *dynamically* decide the level of parallelism to use (`tf.data.experimental.AUTOTUNE`).
- For small datasets, you can `cache` the subsequent batches to be available after the current epoch.

What makes `tf.data` different from others?

- `tf.data` can *dynamically* decide the level of parallelism to use (`tf.data.experimental.AUTOTUNE`).
- For small datasets, you can `cache` the subsequent batches to be available after the current epoch.

Know more here: https://www.tensorflow.org/guide/data_performance.

Enough talking! Show me some `code`!




Building data pipelines with `tf.data`

- Use `tf.data.Dataset`.

Building data pipelines with `tf.data`

- Use `tf.data.Dataset`.



```
# FashionMNIST data along with images and labels
(train, test) = tf.keras.datasets.fashion_mnist.load_data()

# Create tf.data.Dataset!
X_train, y_train = train
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
X_test, y_test = test
test_dataset = tf.data.Dataset.from_tensor_slices((X_test, y_test))
```

Building data pipelines with `tf.data`

- Use `tf.data.Dataset`.

```
# FashionMNIST data along with images and labels
(train, test) = tf.keras.datasets.fashion_mnist.load_data()

# Create tf.data.Dataset!
X_train, y_train = train
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
X_test, y_test = test
test_dataset = tf.data.Dataset.from_tensor_slices((X_test, y_test))
```


```
train_dataset
<TensorSliceDataset shapes: ((28, 28), ()), types: (tf.uint8, tf.uint8)>
```


Building data pipelines with `tf.data`

- Use `tf.data.Dataset`.
- Shuffle, repeat, batch and prefetch the data.

Building data pipelines with `tf.data`

- Use `tf.data.Dataset`.
- Shuffle, repeat, batch and prefetch the data.



```
train_dataset = train_dataset.\n    shuffle(buffer_size=1000).\n    repeat().\n    batch(256).\n    prefetch(buffer_size=1000)
```

Building data pipelines with `tf.data`

- Use `tf.data.Dataset`.
- Shuffle, repeat, batch and prefetch the data.

```
train_dataset = train_dataset.\n    shuffle(buffer_size=1000).\n    repeat().\n    batch(256).\n    prefetch(buffer_size=1000)
```

Verify shapes!



```
for (images, labels) in train_dataset.take(1):\n    pass\n\nprint(images.shape) # TensorShape([256, 28, 28])
```

Building data pipelines with `tf.data`

- Use `tf.data.Dataset`.
- Shuffle, repeat, batch and prefetch the data.
- Define, compile and train your model! (optional)

Building data pipelines with `tf.data`


- Use `tf.data.Dataset`.
- Shuffle, repeat, batch and prefetch the data.
- Define, compile and train your model! (optional)

```
# Define and compile a model
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Building data pipelines with `tf.data`


- Use `tf.data.Dataset`.
- Shuffle, repeat, batch and prefetch the data.
- Define, compile and train your model! (optional)



```
model.fit(train_dataset,  
          steps_per_epoch=len(X_train)//256,  
          epochs=5,  
          validation_data=test_dataset.batch(256))
```

Building data pipelines with `tf.data`

- Use `tf.data.Dataset`.
- Shuffle, repeat, batch and prefetch the data.
- Define, compile and train your model! (optional)



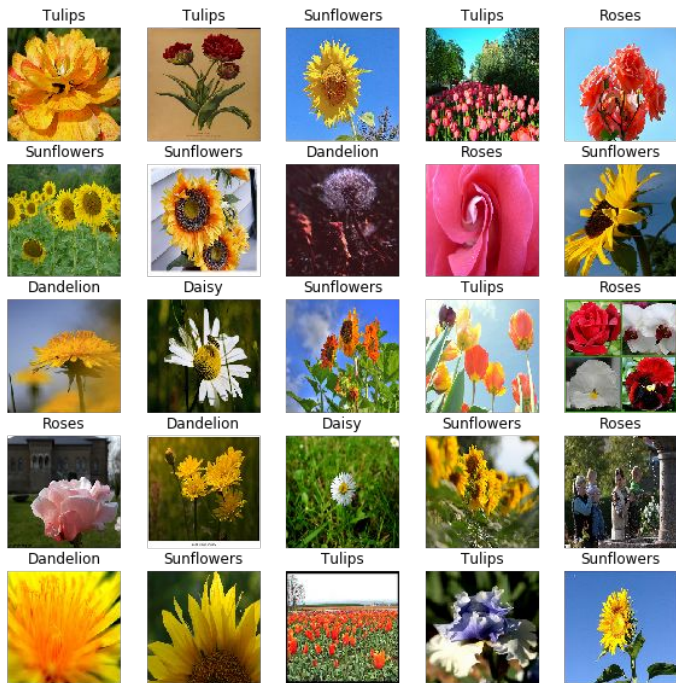
```
model.fit(train_dataset,  
          steps_per_epoch=len(X_train)//256,  
          epochs=5,  
          validation_data=test_dataset.batch(256))
```

You can play with the notebook here: <http://bit.ly/tfdata1>

`tf.data + ImageDataGenerator` =



We will use the **Flowers** dataset.



`tf.data + ImageDataGenerator` = 

- Initialize `ImageDataGenerator` with the augmentations.

tf.data + ImageDataGenerator = 

- Initialize `ImageDataGenerator` with the augmentations.

```
train_aug = ImageDataGenerator(  
    rotation_range=30,  
    zoom_range=0.15,  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    shear_range=0.15,  
    horizontal_flip=True,  
    fill_mode="nearest")
```

`tf.data + ImageDataGenerator` = 

- Initialize `ImageDataGenerator` with the augmentations.
- Wrap the generator with `tf.data`.

`tf.data + ImageDataGenerator` = 

- Initialize `ImageDataGenerator` with the augmentations.
- Wrap the generator with `tf.data`.

```
train_set = tf.data.Dataset.from_generator(  
    lambda: train_aug.flow_from_directory(flowers,  
        class_mode="categorical",  
        target_size=(224, 224),  
        color_mode="rgb",  
        shuffle=True),  
    output_types=(tf.float32, tf.float32),  
    output_shapes=( [None,224,224,3],[None,5])  
)
```

`tf.data + ImageDataGenerator` = 

- Initialize `ImageDataGenerator` with the augmentations.
- Wrap the generator with `tf.data`.
- And voila!

```
model = get_me_a_good_model()  
model.fit(train_set,  
          steps_per_epoch=train_data//32,  
          epochs=5)
```

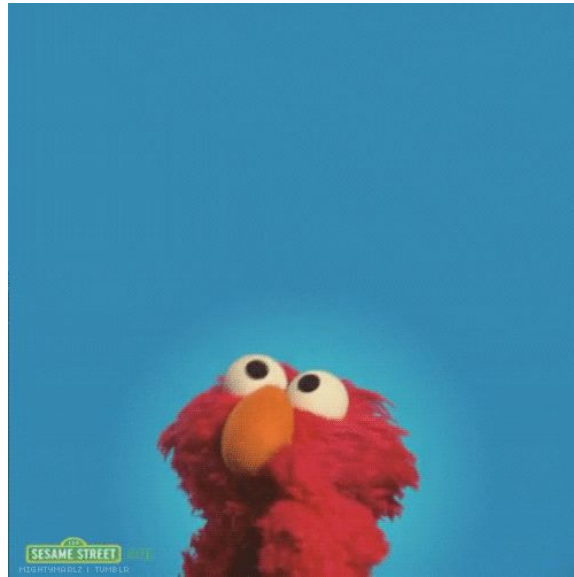
`tf.data + ImageDataGenerator` = 

- Initialize `ImageDataGenerator` with the augmentations.
- Wrap the generator with `tf.data`.
- And voila!

```
model = get_me_a_good_model()  
model.fit(train_set,  
          steps_per_epoch=train_data//32,  
          epochs=5)
```

You can play with the notebook here: <http://bit.ly/tfdata2>

In case you were wondering ...



`tf.data` is (quite) *fast*

`tf.data` is (quite) *fast*

- It drastically speeds up the data loading time.

`tf.data` is (quite) *fast*

- It drastically speeds up the data loading time.
 - Here's a comparison on the **FashionMNIST** dataset:

`tf.data` is (quite) *fast*

- It drastically speeds up the data loading time.
 - Here's a comparison on the **FashionMNIST** dataset:
 - Data loading with `ImageDataGenerator`:

```
1000 batches: 4.16782808303833 s  
61422.87899 Images/s
```

`tf.data` is (quite) *fast*

- It drastically speeds up the data loading time.
 - Here's a comparison on the **FashionMNIST** dataset:
 - Data loading with `ImageDataGenerator`:

```
1000 batches: 4.16782808303833 s  
61422.87899 Images/s
```

- Data loading with `tf.data`:

```
1000 batches: 0.6213550567626953 s  
412002.76269 Images/s
```

`tf.data` is (quite) *fast*

- It drastically speeds up the data loading time.
- Fast data loading indeed speeds up model training.

`tf.data` is (quite) *fast*

- It drastically speeds up the data loading time.
- Fast data loading indeed speeds up model training.
 - `ImageDataGenerator` on the **Flowers dataset**:

tf.data is (quite) *fast*

- It drastically speeds up the data loading time.
- Fast data loading indeed speeds up model training.
 - `ImageDataGenerator` on the **Flowers dataset**:

```
114/114 [=====] - 144s 1s/step - loss: 5.5306 - accuracy: 0.5962
Epoch 2/5
114/114 [=====] - 135s 1s/step - loss: 3.3431 - accuracy: 0.7526
Epoch 3/5
114/114 [=====] - 135s 1s/step - loss: 2.7142 - accuracy: 0.7974
Epoch 4/5
114/114 [=====] - 135s 1s/step - loss: 2.4073 - accuracy: 0.8120
Epoch 5/5
114/114 [=====] - 135s 1s/step - loss: 2.0779 - accuracy: 0.8378
It took 685.5545630455017 seconds
```

`tf.data` is (quite) *fast*

- It drastically speeds up the data loading time.
- Fast data loading indeed speeds up the model training.
 - `ImageDataGenerator` on the **Flowers dataset**:
 - `tf.data` on the **Flowers dataset**:

`tf.data` is (quite) *fast*

- It drastically speeds up the data loading time.
- Fast data loading indeed speeds up the model training.
 - `ImageDataGenerator` on the **Flowers dataset**:
 - `tf.data` on the **Flowers dataset**:

```
Epoch 1/5
Found 3670 images belonging to 5 classes.
114/114 [=====] - 80s 704ms/step - loss: 5.2687 - accuracy: 0.6757
Epoch 2/5
114/114 [=====] - 82s 715ms/step - loss: 0.9821 - accuracy: 0.7883
Epoch 3/5
114/114 [=====] - 79s 695ms/step - loss: 0.6616 - accuracy: 0.8219
Epoch 4/5
114/114 [=====] - 79s 695ms/step - loss: 0.4871 - accuracy: 0.8535
Epoch 5/5
114/114 [=====] - 80s 699ms/step - loss: 0.3722 - accuracy: 0.8821
It took 399.8591330051422 seconds
```

`tf.data` is (quite) *fast*

- It drastically speeds up the data loading time.
- Fast data loading indeed speeds up model training.

Comparison notebooks available here: <http://bit.ly/tf-data>

Explore more about `tf.data`

- <https://www.tensorflow.org/guide/data>
- https://www.tensorflow.org/guide/data_performance
- PyImageSearch future blogs
- PyImageSearch's book [DL4CV](#) will be updated with `tf.data` in a future release
- [Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow](#) (2nd ed.)

Slides available here:
<http://bit.ly/MLWeekend19>

See you next time



Find me here:
sayak.dev

Thank you very much :)



Experts

