深度之眼
deepshare.net

Dataset类详解

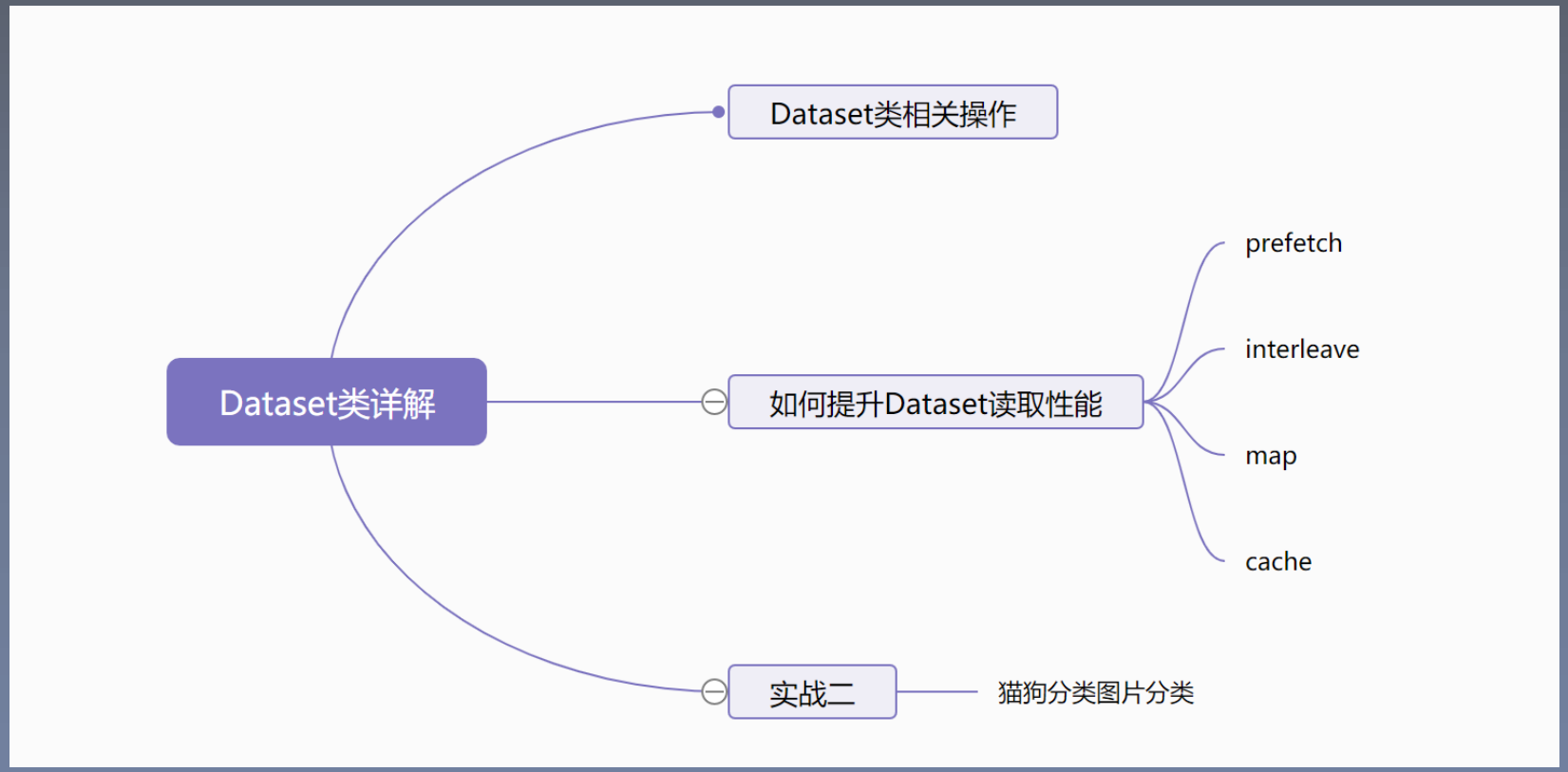
导师：GAUSS

深度之眼
deepshare.net

目录

知识树

Knowledge tree



深度之眼
deepshare.net

Dataset类相关操作

Dataset创建数据集

tf.data.Dataset 类创建数据集，对数据集实例化。最常用的如：

tf.data.Dataset.from_tensors() ： 创建Dataset对象， 合并输入并返回具有单个元素的数据集。

tf.data.Dataset.from_tensor_slices() ： 创建一个Dataset对象，输入可以是一个或者多个 tensor，若是多个 tensor，需要以元组或者字典等形式组装起来。

tf.data.Dataset. from_generator() ： 迭代生成所需的数据集，一般数据量较大时使用。

注：Dataset可以看作是相同类型“元素”的有序列表。在实际使用时，单个“元素”可以是向量，也可以是字符串、图片，甚至是tuple或者dict。

Dataset类操作

Dataset包含了非常丰富的数据转换功能：

- map(f): 对数据集中的每个元素应用函数 f ，得到一个新的数据集（这部分往往结合 tf.io 进行读写和解码文件， tf.image 进行图像处理）
- shuffle(buffer_size)：将数据集打乱（设定一个固定大小的缓冲区（Buffer），取出前 buffer_size 个元素放入，并从缓冲区中随机采样，采样后的数据用后续数据替换）
- repeat(count)：数据集重复次数。
- batch(batch_size)：将数据集分成批次，即对每 batch_size 个元素，使用 tf.stack() 在第 0 维合并，成为一个元素；

Dataset类操作


```
def _decode_and_resize(filename, label):
    image_string = tf.io.read_file(filename)            # 读取原始文件
    image_decoded = tf.image.decode_jpeg(image_string)  # 解码JPEG图片
    image_resized = tf.image.resize(image_decoded, [256, 256]) / 255.0
    return image_resized, label


batch_size = 32
train_dataset = tf.data.Dataset.from_tensor_slices((train_filenames, train_labels))
train_dataset = train_dataset.map(
    map_func=_decode_and_resize,
    num_parallel_calls=tf.data.experimental.AUTOTUNE)


# 取出前buffer_size个数据放入buffer，并从其中随机采样，采样后的数据用后续数据替换
train_dataset = train_dataset.shuffle(buffer_size=23000)

train_dataset = train_dataset.repeat(count=3)

train_dataset = train_dataset.batch(batch_size)
```

Dataset类操作

- flat_map(): 将map函数映射到数据集的每一个元素，并将嵌套的Dataset压平。
- interleave(): 效果类似flat_map,但可以将不同来源的数据夹在一起。
- take(): 截取数据集中的前若干个元素

flat_map使用


```
flat_map(
    map_func
)
```

将map函数映射到数据集的每一个元素，并将嵌套的Dataset压平。

```
dataset = tf.data.Dataset.from_tensor_slices([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
dataset_flat = dataset.flat_map(lambda x: tf.data.Dataset.from_tensor_slices(x))
```

```
for line in dataset:
    print(line)
```

```
tf.Tensor([1 2 3], shape=(3,), dtype=int32)
tf.Tensor([4 5 6], shape=(3,), dtype=int32)
tf.Tensor([7 8 9], shape=(3,), dtype=int32)
```

```
for line in dataset_flat:
    print(line)
```

```
tf.Tensor(1, shape=(), dtype=int32)
tf.Tensor(2, shape=(), dtype=int32)
tf.Tensor(3, shape=(), dtype=int32)
tf.Tensor(4, shape=(), dtype=int32)
tf.Tensor(5, shape=(), dtype=int32)
tf.Tensor(6, shape=(), dtype=int32)
tf.Tensor(7, shape=(), dtype=int32)
tf.Tensor(8, shape=(), dtype=int32)
tf.Tensor(9, shape=(), dtype=int32)
```

```
a = tf.data.Dataset.range(1, 6)  # ==> [ 1, 2, 3, 4, 5 ]
# NOTE: New lines indicate "block" boundaries.
b=a.flat_map(lambda x: tf.data.Dataset.from_tensors(x).repeat(6))
for item in b:
    print(item.numpy(),end=', ')
```

```
1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5,
```

interleave使用


```
interleave(
    map_func, cycle_length=AUTOTUNE, block_length=1, num_parallel_calls=None,
    deterministic=None
)
```

interleave()是Dataset的类方法，所以interleave是作用在一个Dataset上的。

首先该方法会从该Dataset中取出cycle_length个element，然后对这些element apply map_func，得到cycle_length个新的Dataset对象。然后从这些新生成的Dataset对象中取数据，每个Dataset对象一次取block_length个数据。当新生成的某个Dataset的对象取尽时，从原Dataset中再取一个element，然后apply map_func，以此类推。

interleave使用

interleave使用案例

```
filenames = ["./interleave_data/train.csv", "./interleave_data/eval.csv",
             "./interleave_data/train.csv", "./interleave_data/eval.csv",]
dataset = tf.data.Dataset.from_tensor_slices(filenames)

def data_func(line):
    line = tf.strings.split(line, sep = ",")
    return line

dataset_1 = dataset.interleave(lambda x:
    tf.data.TextLineDataset(x).skip(1).map(data_func),
    cycle_length=4, block_length=16)
```

```
for line in dataset_1.take(2):
    print(line)
```

```
tf.Tensor(
[b'0' b'male' b'22.0' b'1' b'0' b'7.25' b'Third' b'unknown' b'Southampton'
 b'n'], shape=(10,), dtype=string)
tf.Tensor(
[b'1' b'female' b'38.0' b'1' b'0' b'71.2833' b'First' b'C' b'Cherbourg'
 b'n'], shape=(10,), dtype=string)
```

Dataset类操作

filter: 过滤掉某些元素。

zip: 将两个长度相同的Dataset横向铰合。

concatenate: 将两个Dataset纵向连接。

reduce: 执行归并操作。

zip方法

zip: 将两个长度相同的Dataset横向铰合。

```
a = tf.data.Dataset.range(1, 4)  # ==> [ 1, 2, 3 ]
b = tf.data.Dataset.range(4, 7) # ==> [ 4, 5, 6 ]
ds = tf.data.Dataset.zip((a, b))
```

```
for line in ds:
    print(line)
```

```
(<tf.Tensor: id=182, shape=(), dtype=int64, numpy=1>, <tf.Tensor: id=183, shape=(), dtype=int64, numpy=4>)
(<tf.Tensor: id=184, shape=(), dtype=int64, numpy=2>, <tf.Tensor: id=185, shape=(), dtype=int64, numpy=5>)
(<tf.Tensor: id=186, shape=(), dtype=int64, numpy=3>, <tf.Tensor: id=187, shape=(), dtype=int64, numpy=6>)
```

```
ds = tf.data.Dataset.zip((b, a))
for line in ds:
    print(line)
```

```
(<tf.Tensor: id=194, shape=(), dtype=int64, numpy=4>, <tf.Tensor: id=195, shape=(), dtype=int64, numpy=1>)
(<tf.Tensor: id=196, shape=(), dtype=int64, numpy=5>, <tf.Tensor: id=197, shape=(), dtype=int64, numpy=2>)
(<tf.Tensor: id=198, shape=(), dtype=int64, numpy=6>, <tf.Tensor: id=199, shape=(), dtype=int64, numpy=3>)
```

concatenate方法

concatenate: 将两个Dataset纵向连接。

```
a = tf.data.Dataset.range(1, 4)  # ==> [ 1, 2, 3 ]
b = tf.data.Dataset.range(4, 7) # ==> [ 4, 5, 6 ]
ds = a.concatenate(b)
for line in ds:
    print(line)
```

```
tf.Tensor(1, shape=(), dtype=int64)
tf.Tensor(2, shape=(), dtype=int64)
tf.Tensor(3, shape=(), dtype=int64)
tf.Tensor(4, shape=(), dtype=int64)
tf.Tensor(5, shape=(), dtype=int64)
tf.Tensor(6, shape=(), dtype=int64)
```

深度之眼
deepshare.net

如何提升Dataset读取性能

如何提升Dataset读取性能

训练深度学习模型常常会非常耗时。模型训练的耗时主要来自于两个部分，一部分来自**数据准备**，另一部分来自**参数迭代**。

参数迭代过程的耗时通常依赖于GPU来提升， 而数据准备过程的耗时则可以通过构建高效的数据管道进行提升。

如何提升Dataset读取性能

以下是一些构建高效数据管道的建议。

1、使用 prefetch 方法让数据准备和参数迭代两个过程相互并行。

2、使用 interleave 方法可以让数据读取过程多进程执行,并将不同来源数据夹在一起。

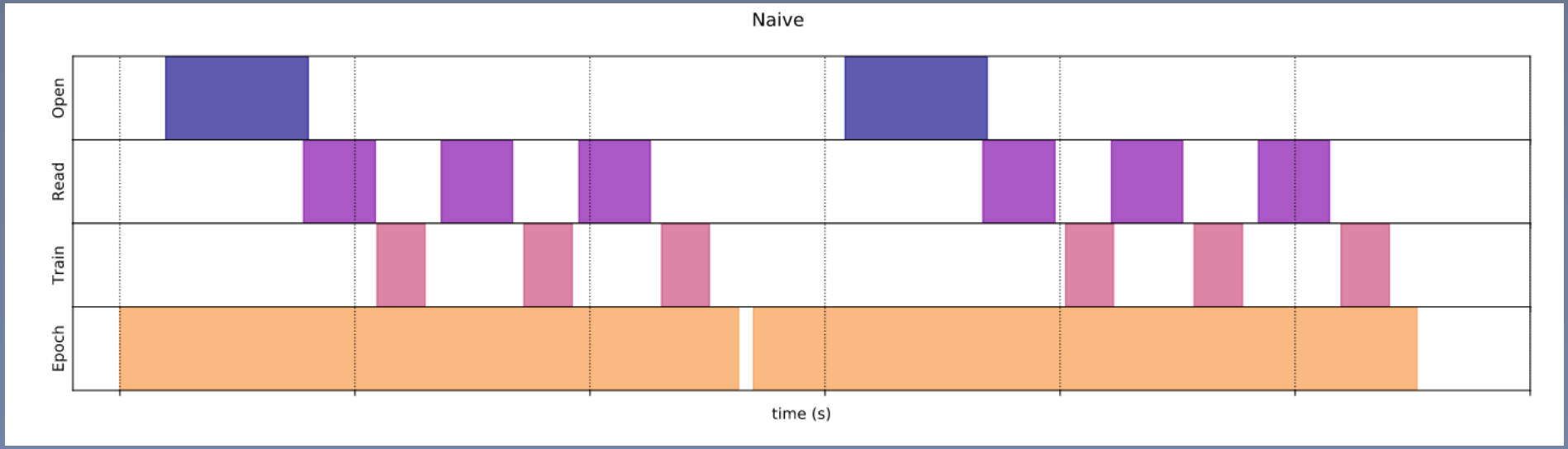
3、使用 map 时设置 num_parallel_calls 让数据转换过程多进程执行。

4、使用 cache 方法让数据在第一个epoch后缓存到内存中，仅限于数据集不大情形。

prefetch 方法

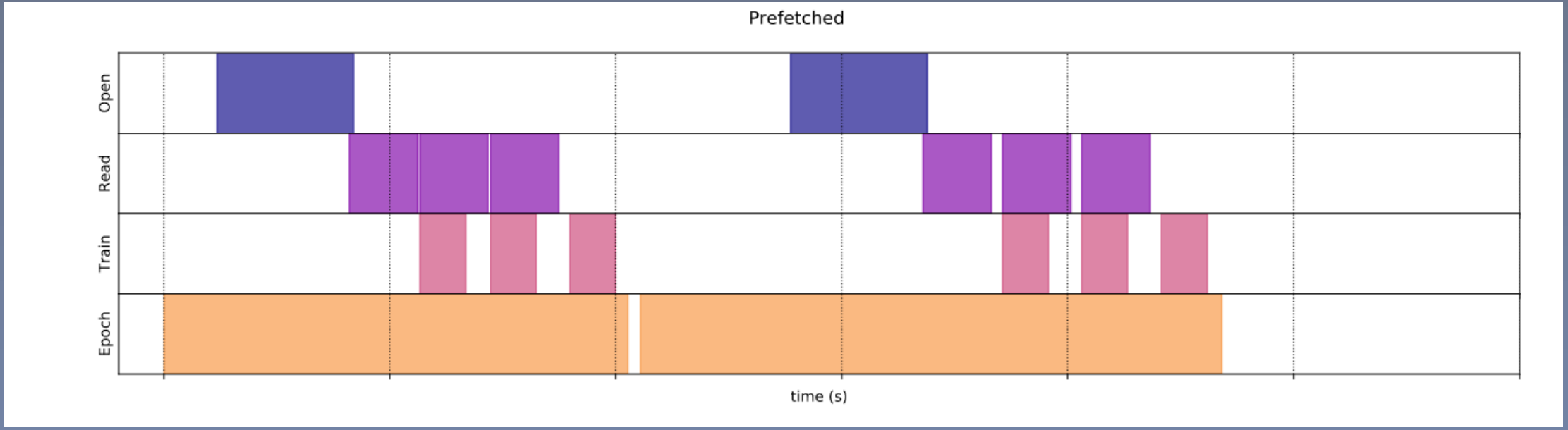
原始方法执行,可以看到执行训练步骤涉及：

- 打开文件（如果尚未打开）
- 从文件中获取数据条目，
- 使用数据进行训练。



prefetch 方法

prefetch与训练步骤的预处理和模型执行重叠。当模型执行训练步骤时s，输入管道将读取步骤s+1的数据。这样做可以将步长时间减少到训练的最大值（而不是总和），并减少提取数据所需的时间。



prefetch 方法

该tf.dataAPI提供了tf.data.Dataset.prefetch方法。它可用于将产生数据的时间与消耗数据的时间分开。特别是，map使用后台线程和内部缓冲区在请求输入之前，提前从输入数据集中预提取元素。

注意：要预取的元素数量应等于（或可能大于）单个训练步骤消耗的batch数量。可以手动调整此值，也可以将其设置为tf.data.experimental.AUTOTUNE，提示 tf.data运行时在运行时动态调整值的值。

prefetch 方法


```python
#构建训练集

def _decode_and_resize(filename, label):
    image_string = tf.io.read_file(filename)            # 读取原始文件
    image_decoded = tf.image.decode_jpeg(image_string)  # 解码JPEG图片
    image_resized = tf.image.resize(image_decoded, [256, 256]) / 255.0
    return image_resized, label


batch_size = 32
train_dataset = tf.data.Dataset.from_tensor_slices((train_filenames, train_labels))

def benchmark(dataset, num_epochs=1):
    start_time = time.perf_counter()
    for epoch_num in range(num_epochs):
        for sample in dataset:
            # Performing a training step
            time.sleep(0.01)
    tf.print("Execution time:", time.perf_counter() - start_time)
```

```python
benchmark(train_dataset.map(
    map_func=_decode_and_resize,
    num_parallel_calls=tf.data.experimental.AUTOTUNE),
    num_epochs=1)
```

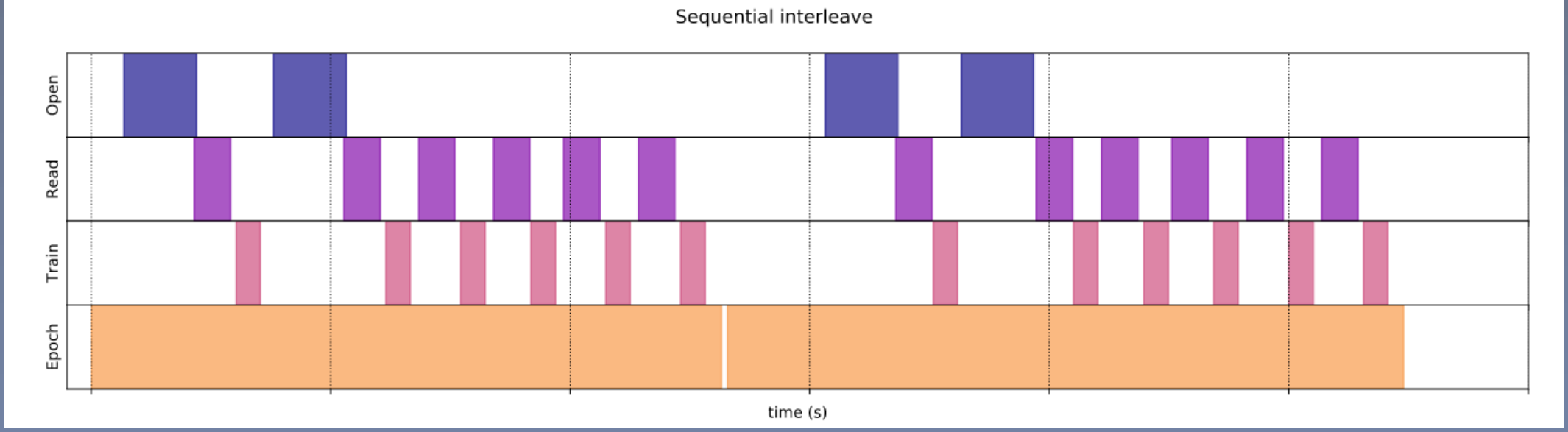
```
Execution time: 21.706319809699337
```

```python
benchmark(
    train_dataset.map(
    map_func=_decode_and_resize,
    num_parallel_calls=tf.data.experimental.AUTOTUNE)
    .prefetch(tf.data.experimental.AUTOTUNE),
    num_epochs=1
)
```

```
Execution time: 21.69249288979927
```

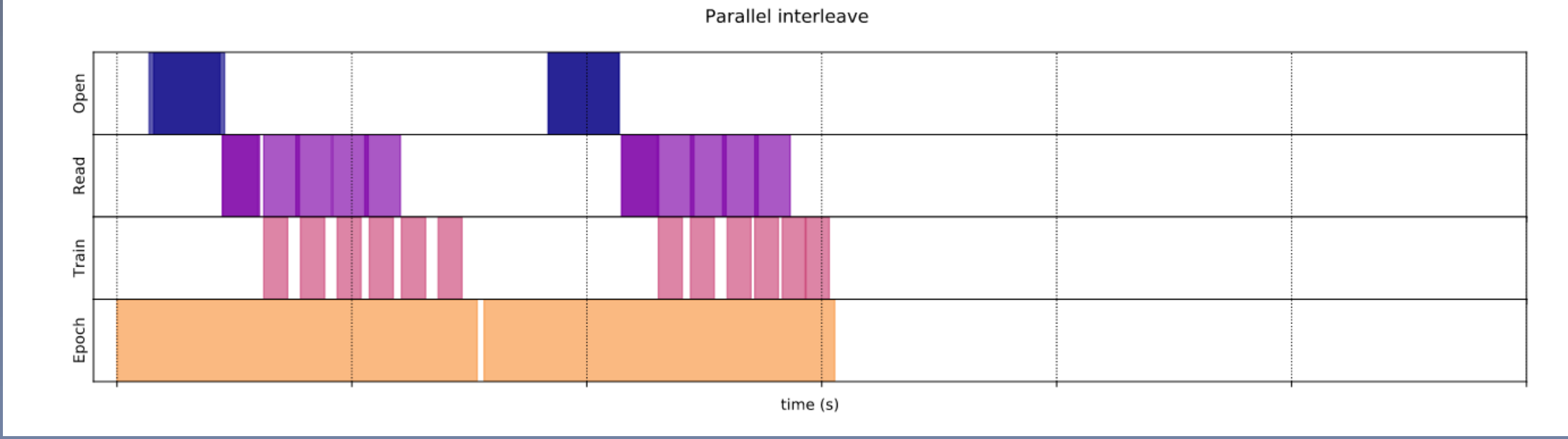
interleave 方法

tf.data.Dataset.interleave可以进行并行化数据加载，并交织其他数据集（例如数据文件读取器）的内容。可以通过**cycle_length**参数指定要重叠的数据集数量，而并行度则可以通过**num_parallel_calls**参数指定。



interleave 方法

现在使用interleave方法的**num_parallel_calls**。这样可以并行加载多个数据集，从而减少了等待文件打开的时间。



interleave 方法


```
filenames = ["./interleave_data/train.csv", "./interleave_data/eval.csv",
             "./interleave_data/train.csv", "./interleave_data/eval.csv",]
dataset = tf.data.Dataset.from_tensor_slices(filenames)

def data_func(line):
    line = tf.strings.split(line, sep = ",")
    return line

dataset_1 = dataset.interleave(lambda x:
    tf.data.TextLineDataset(x).skip(1).map(data_func),
    cycle_length=4, block_length=16)
```

```
benchmark(dataset_1,
    num_epochs=1)
```

```
Execution time: 19.46445847785003
```

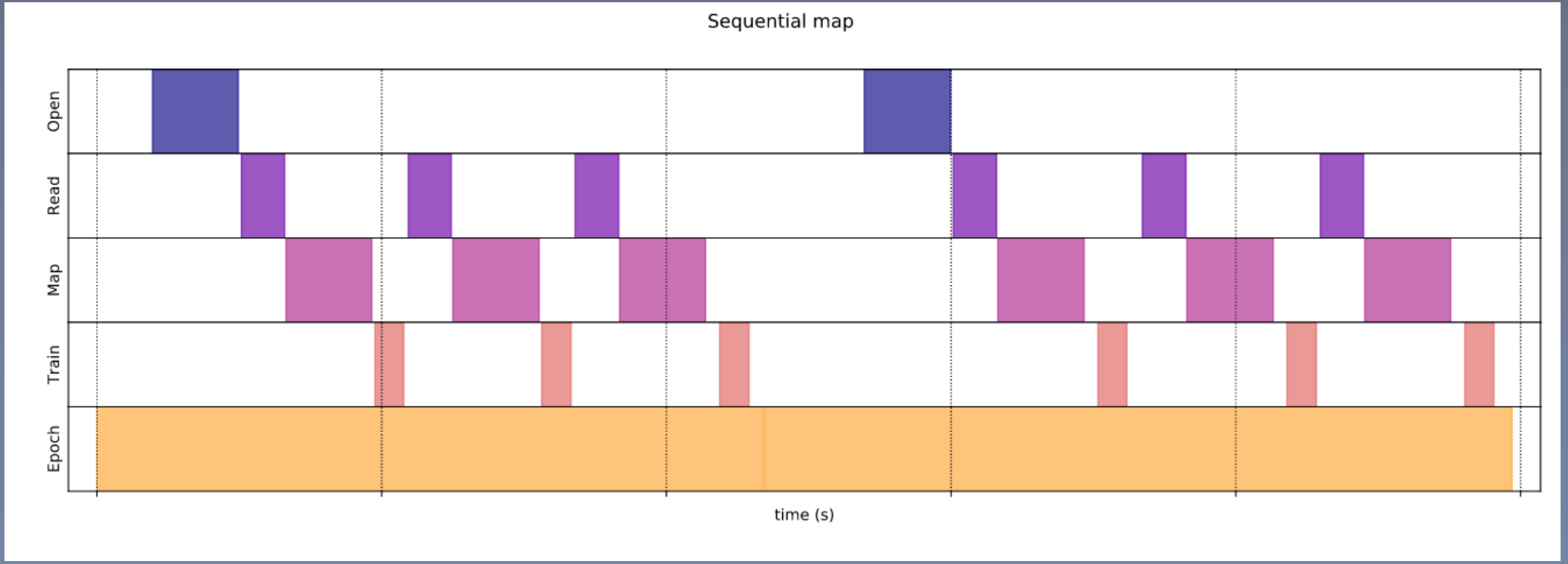
```
dataset_2 = dataset.interleave(lambda x:
    tf.data.TextLineDataset(x).skip(1).map(data_func),
    num_parallel_calls=tf.data.experimental.AUTOTUNE,
    cycle_length=4, block_length=16)
```

```
benchmark(dataset_2,
    num_epochs=1)
```

```
Execution time: 19.427184579315735
```

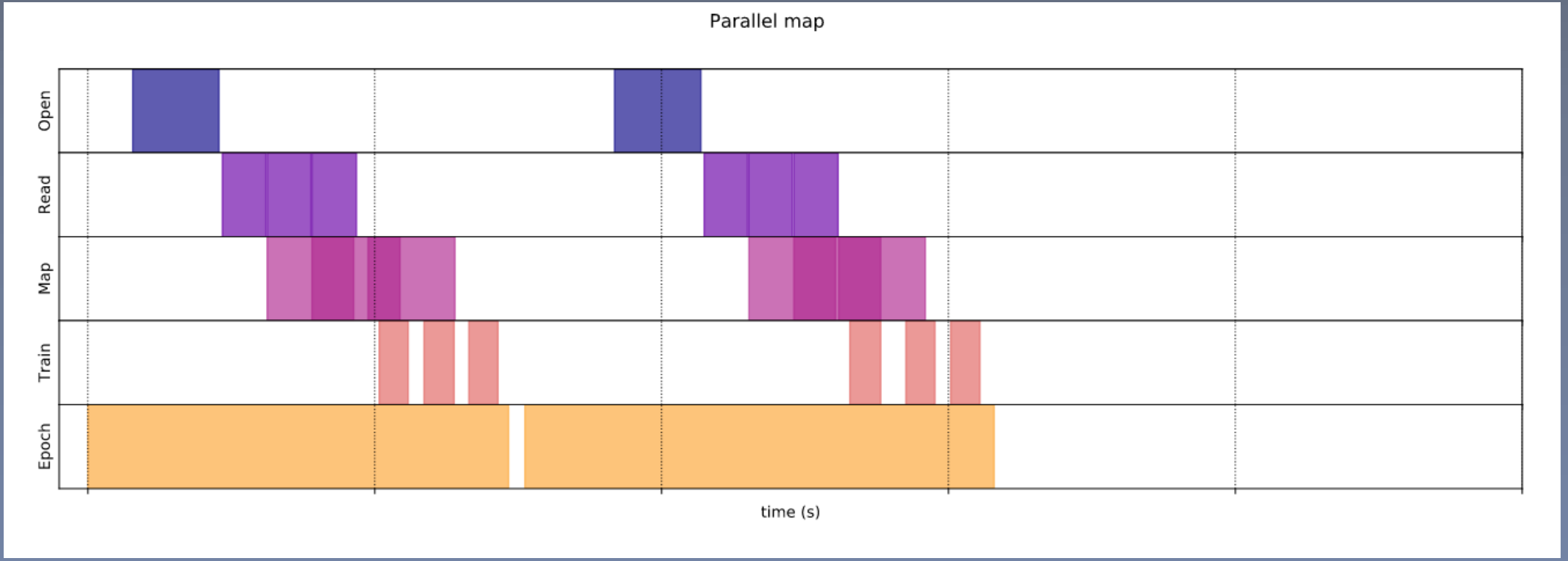
map多进程执行

最简单的方法这里花费在open，read，预处理（map）和训练步骤上的时间加在一起进行一次迭代。



map多进程执行

使用相同的预处理功能，但map多进程执行数据预处理。



map多进程执行

用cat vs dog 数据集测试结果如右图：

```python
def _decode_and_resize(filename, label):
    image_string = tf.io.read_file(filename)            # 读取原始文件
    image_decoded = tf.image.decode_jpeg(image_string)  # 解码JPEG图片
    image_resized = tf.image.resize(image_decoded, [256, 256]) / 255.0
    return image_resized, label


batch_size = 32
train_dataset = tf.data.Dataset.from_tensor_slices((train_filenames, train_labels))

def benchmark(dataset, num_epochs=1):
    start_time = time.perf_counter()
    for epoch_num in range(num_epochs):
        for sample in dataset:
            # Performing a training step
            time.sleep(0.00001)
    tf.print("Execution time:", time.perf_counter() - start_time)
```

```
benchmark(
    train_dataset.map(
    map_func=_decode_and_resize,
    num_parallel_calls=tf.data.experimental.AUTOTUNE)
)
```

```
Execution time: 52.91160157932336
```

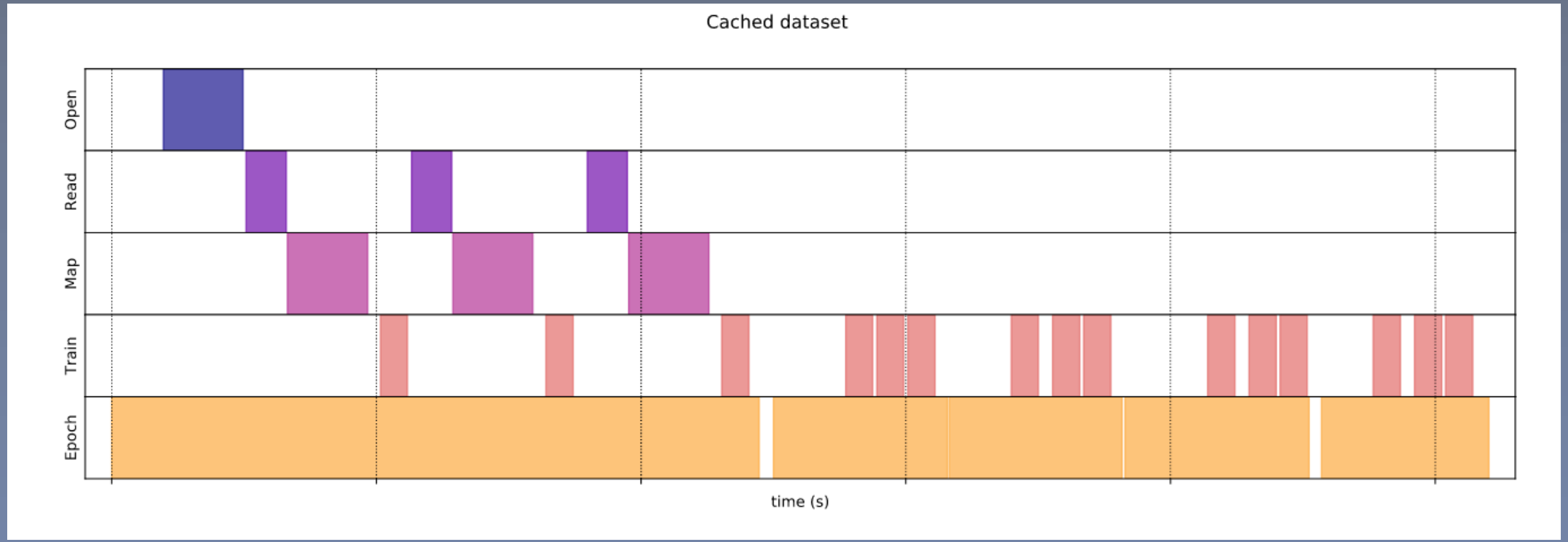
```
benchmark(
    train_dataset.map(
    map_func=_decode_and_resize
    )
)
```

```
Execution time: 106.86491876739251
```

cache 方法

tf.data.Dataset.cache方法可以将数据集缓存在内存或本地存储。这样可以避免在每个epoch执行某些操作（例如文件打开和数据读取）。

缺点：以内存换取时间的行为，适合小数据量，数据量较大请勿使用!



cache 方法

用cat vs dog 数据集测试结果如右图：

```python
def _decode_and_resize(filename, label):
    image_string = tf.io.read_file(filename)            # 读取原始文件
    image_decoded = tf.image.decode_jpeg(image_string)  # 解码JPEG图片
    image_resized = tf.image.resize(image_decoded, [256, 256]) / 255.0
    return image_resized, label


batch_size = 32
train_dataset = tf.data.Dataset.from_tensor_slices((train_filenames, train_labels))

def benchmark(dataset, num_epochs=1):
    start_time = time.perf_counter()
    for epoch_num in range(num_epochs):
        for sample in dataset:
            # Performing a training step
            time.sleep(0.00001)
    tf.print("Execution time:", time.perf_counter() - start_time)
```

```
benchmark(
    train_dataset.map(
    map_func=_decode_and_resize,
    num_parallel_calls=tf.data.experimental.AUTOTUNE
    ),
    num_epochs=2
)
```

```
Execution time: 91.68475344679246
```

```
benchmark(
    train_dataset.map(
    map_func=_decode_and_resize,
    num_parallel_calls=tf.data.experimental.AUTOTUNE
    ).cache(),
    num_epochs=2
)
```

```
benchmark(
    train_dataset.map(
    map_func=_decode_and_resize,
    num_parallel_calls=tf.data.experimental.AUTOTUNE
    ),
    num_epochs=2
)
```

```
Execution time: 8.020815502370919
```

```
benchmark(
    train_dataset.map(
    map_func=_decode_and_resize,
    num_parallel_calls=tf.data.experimental.AUTOTUNE
    ).cache(),
    num_epochs=2
)
```

```
Execution time: 7.906321487192521
```

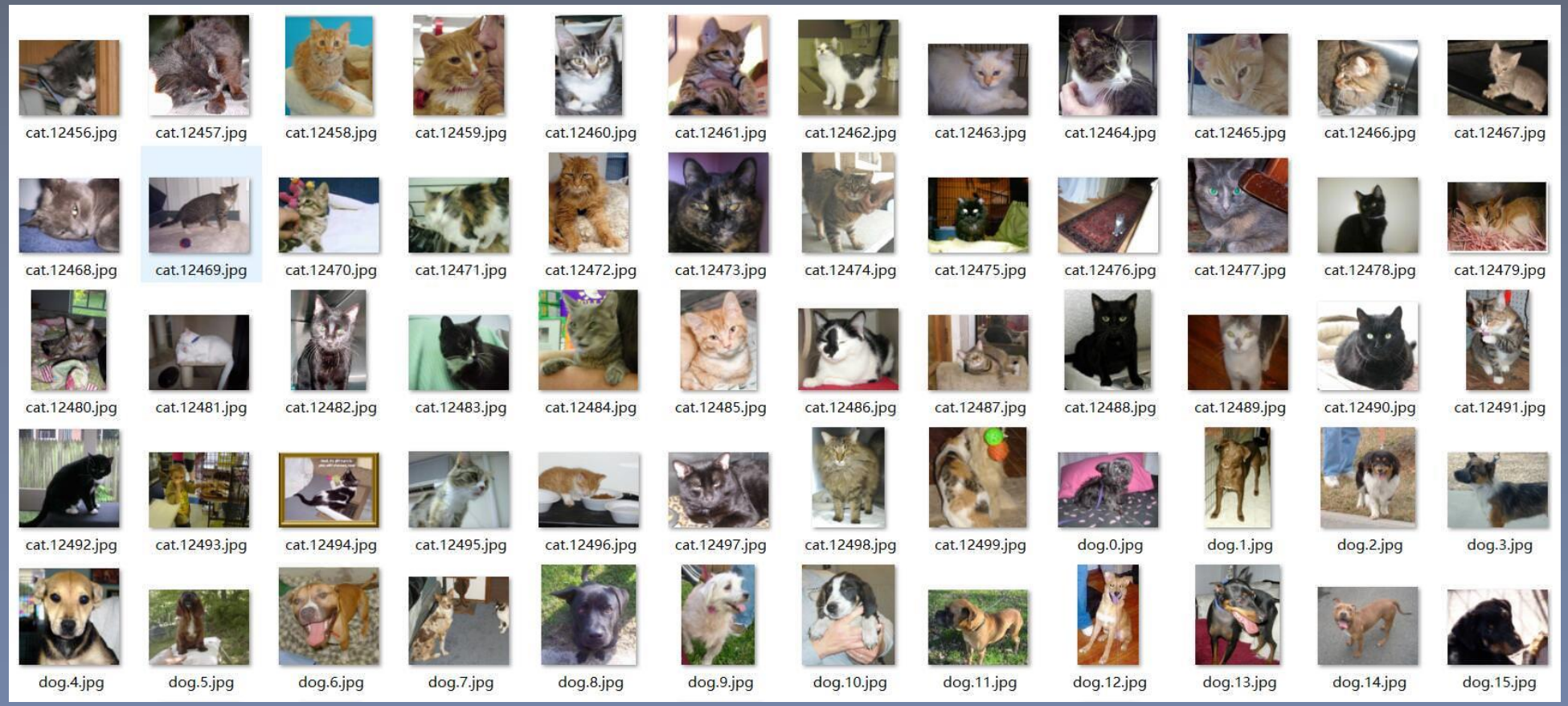
深度之眼
deepshare.net

案例讲解

实战二：Cats vs. Dogs比赛项目

深度之眼
deepshare.net

项目网址：https://www.kaggle.com/c/dogs-vs-cats

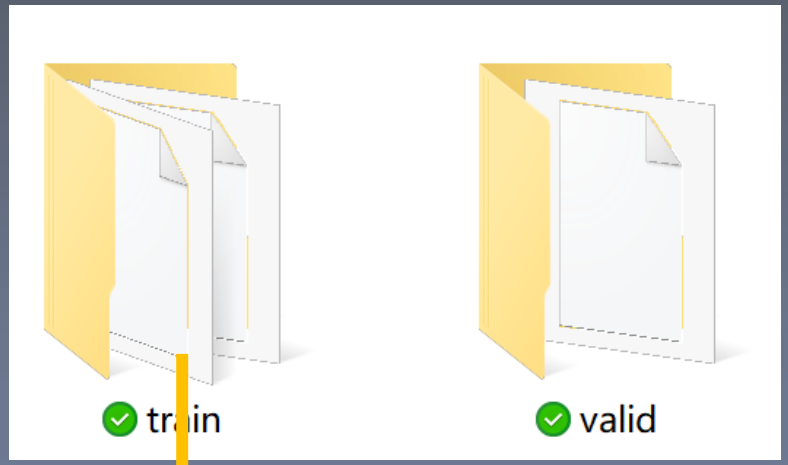


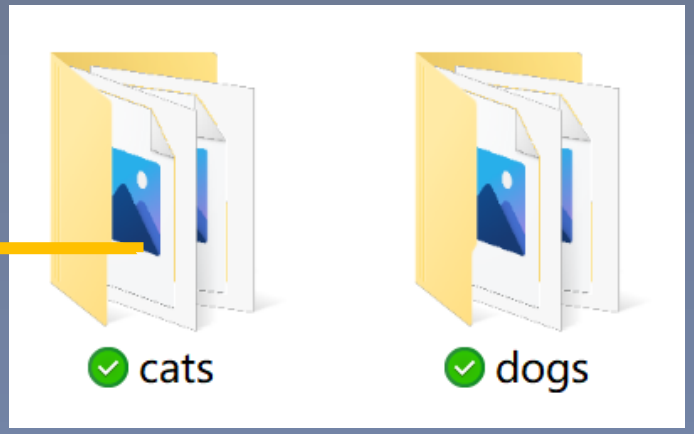
背景介绍

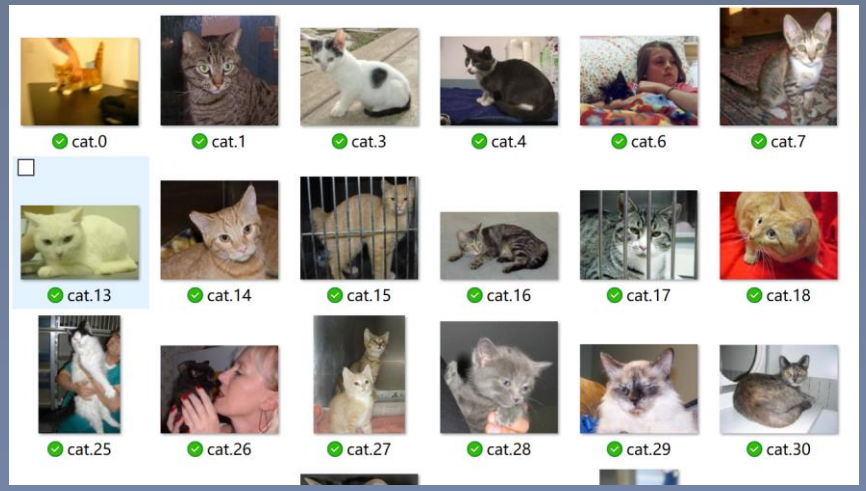
任务目标：Cats vs. Dogs（猫狗大战）是Kaggle大数据竞赛某一年的一道赛题，利用给定的数据集，用算法实现猫和狗的识别。

图像分类问题

数据集介绍







本节小结

Summary

深度之眼
deepshare.net

Dataset类详解	Dataset类相关操作	掌握map、shuffle、batch等相关操作
	如何提升Dataset读取性能	四种提升Dataset类性能的操作
	案例讲解	实战二：Cats vs. Dogs比赛项目

结语

我 说

GAUSS老师个人公众号，主要分享NLP、推荐、比赛实战相关知识！

深度之眼
deepshare.net

联系我们：

电话：18001992849

邮箱：service@deepshare.net

Q Q： 2677693114

公众号

客服微信