Lab 12-1 Convolution Neural Network & Data Pipelines

Department of Computer Science,
National Tsing Hua University, Taiwan
2020.11.02

Outline

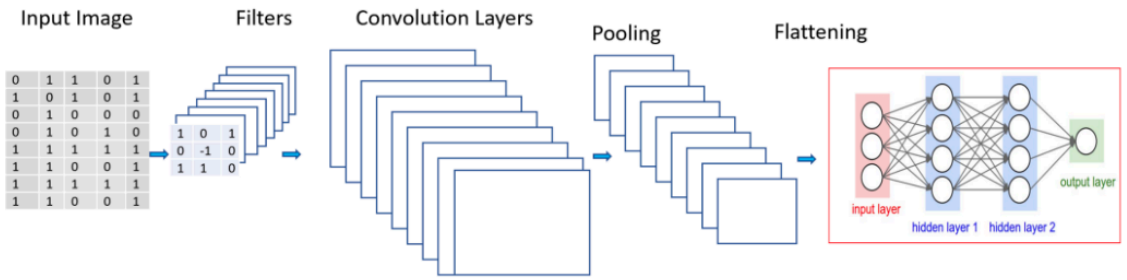
- Convolution neural network
- Input pipeline
- TFRecord

Outline

- Convolution neural network
- Input pipeline
- TFRecord

Convolution Neural Network

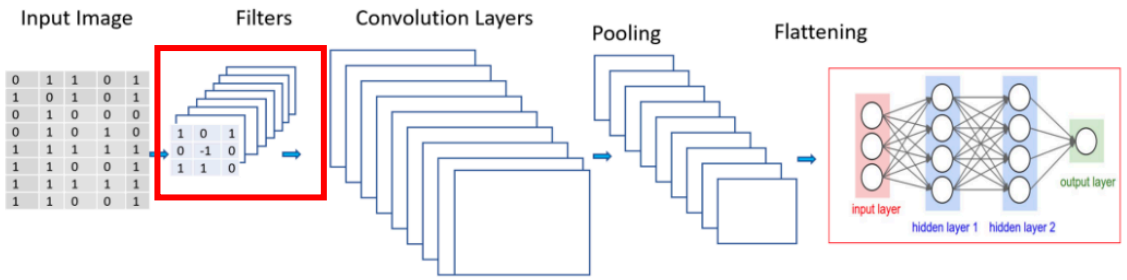
- Build a CNN model via Sequential API
  - A stack of Conv2D and MaxPooling2D layers




```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Convolution Neural Network

- Build a CNN model via Sequential API
  - A stack of Conv2D and MaxPooling2D layers

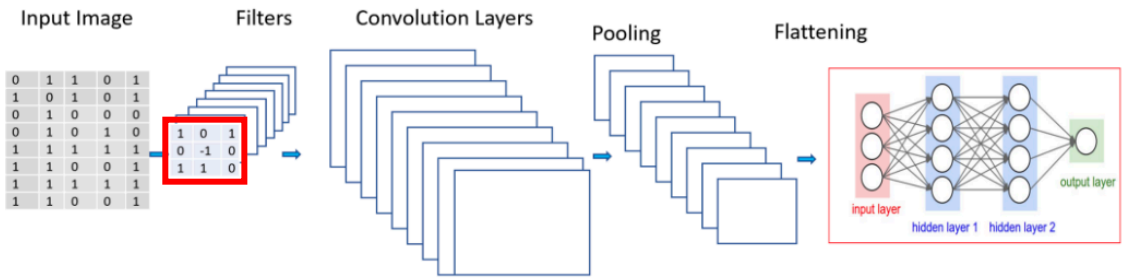


Filters: the number of output filters in the convolution

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Convolution Neural Network

- Build a CNN model via Sequential API
  - A stack of Conv2D and MaxPooling2D layers



Kernel size: specifying the height and width of the 2D convolution window

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3) activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Outline

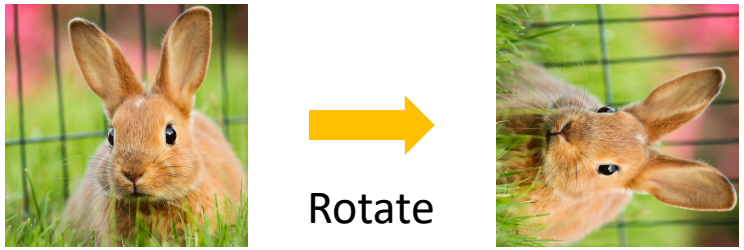
- Convolution neural network
- Input pipeline
- TFRecord

Input Pipeline

- A series of input data processing before training

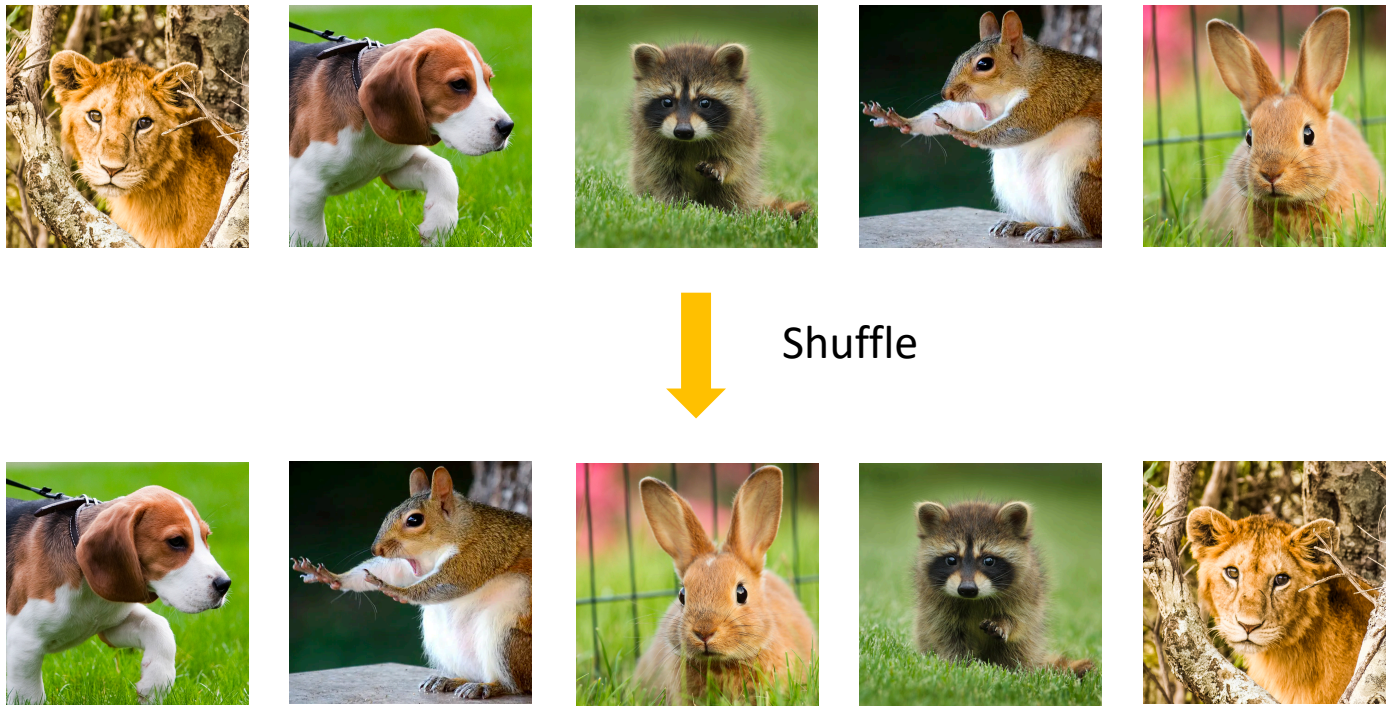
Input Pipeline

- A series of input data processing before training



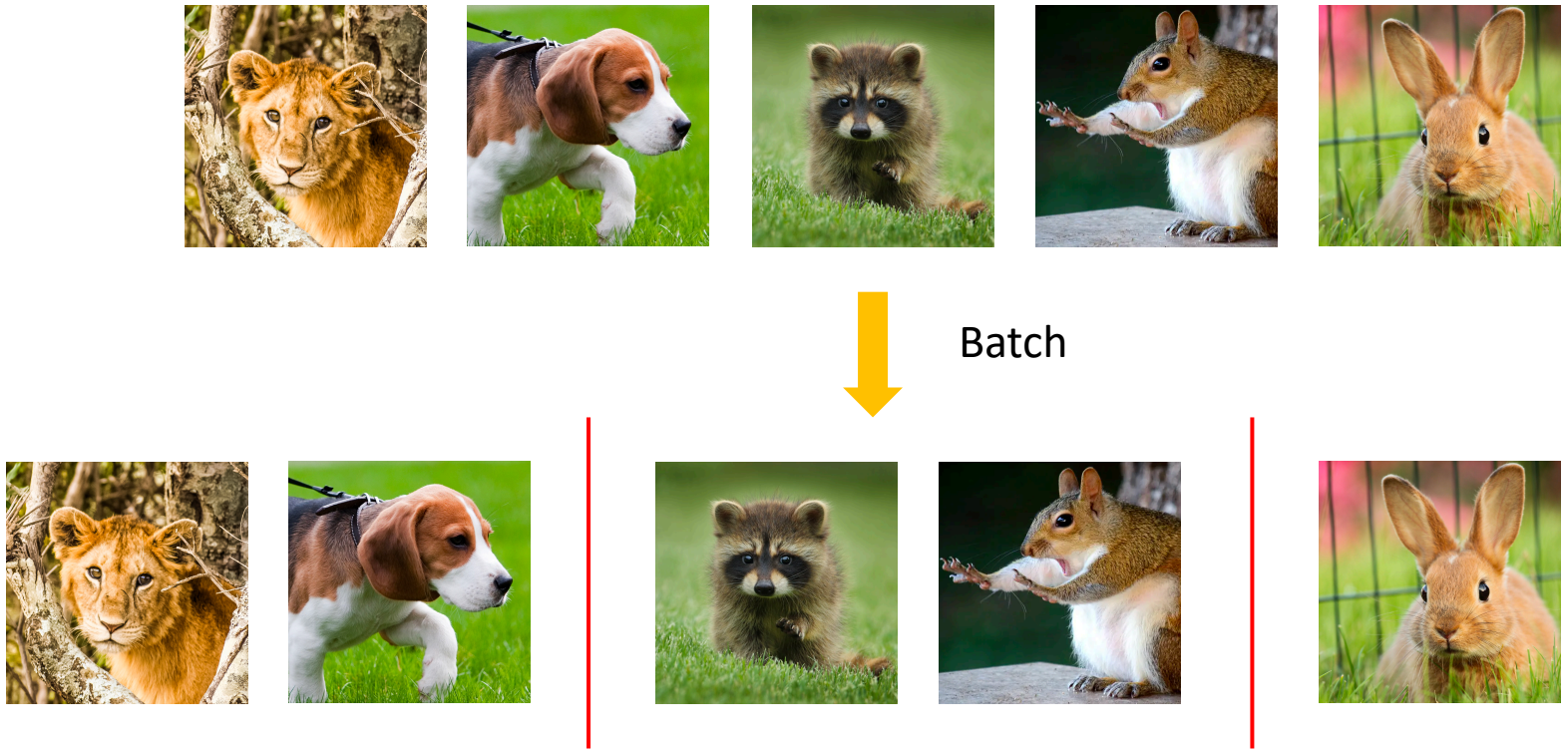
Input Pipeline

- A series of input data processing before training



Input Pipeline

- A series of input data processing before training



Input Pipeline

- A series of input data processing before training
- Building the input pipeline is long and painful, and it’s hard to reuse due to different type of data
- TensorFlow provides an API `tf.data` enables you to build complex input pipelines from simple, reusable pieces

Input Pipeline - TensorFlow

- To apply transformations on your input data, we will need to construct a `tf.data.Dataset` object

Input Pipeline - TensorFlow

- Construct Dataset
  - For small data (in-memory)

```
# number of samples
n_samples = 200

# an array with shape (n_samples, 5)
raw_data_a = np.random.rand(n_samples, 5)
# a list with length of n_samples from 0 to n_samples-1
raw_data_b = np.arange(n_samples)

# this tells the dataset that each row of raw_data_a is corresponding to each element of raw_data_b
raw_dataset = tf.data.Dataset.from_tensor_slices((raw_data_a, raw_data_b))
print(raw_dataset)
```

All input tensors must have the same size in their first dimensions

```
<TensorSliceDataset shapes: ((5,), ()), types: (tf.float64, tf.int64)>
```

The given tensors are sliced along their first dimension.

Input Pipeline - TensorFlow

- Construct Dataset
  - For small data (in-memory)

```
# number of samples
n_samples = 200

# an array with shape (n_samples, 5)
raw_data_a = np.random.rand(n_samples, 5)
# a list with length of n_samples from 0 to n_samples-1
raw_data_b = np.arange(n_samples)

# this tells the dataset that each row of raw_data_a is corresponding to each element of raw_data_b
raw_dataset = tf.data.Dataset.from_tensor_slices((raw_data_a, raw_data_b))
print(raw_dataset)
```

```
<TensorSliceDataset shapes: ((5,), ()), types: (tf.float64, tf.int64)>
```

data label

```
train_ds = tf.data.Dataset.from_tensor_slices((x_train, y_train))
test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test))
```

Input Pipeline - TensorFlow

- Transformations
  - Map: apply the function to each the elements of this dataset

```
def preprocess_function(one_row_a, one_b):
    """
        Input: one slice of the dataset
        Output: modified slice
    """
    # Do some data preprocessing, you can also input filenames and load data in here
    # Here, we transform each row of raw_data_a to its sum and mean
    one_row_a = [tf.reduce_sum(one_row_a), tf.reduce_mean(one_row_a)]

    return one_row_a, one_b

raw_dataset = raw_dataset.map(preprocess_function)
print(raw_dataset)
```

Map function

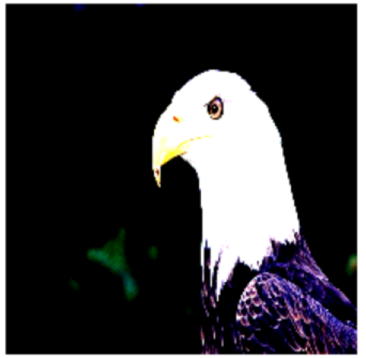
```
<MapDataset shapes: ((2,), ()), types: (tf.float64, tf.int64)>
```

reduce the dimension

Input Pipeline - TensorFlow

- Transformations
  - Map: apply the function to each the elements of this dataset
    - Data augmentation: a technique to increase the diversity of your training set by applying random transformations such as image rotation

```
def pre_train_data(img, label):
    distorted_img = tf.image.random_crop(img, [IMAGE_SIZE_CROPPED,IMAGE_SIZE_CROPPED,IMAGE_DEPTH])
    distorted_img = tf.image.random_flip_left_right(distorted_img)
    distorted_img = tf.image.random_brightness(distorted_img, max_delta=63)
    distorted_img = tf.image.random_contrast(distorted_img, lower=0.2, upper=1.8)
    distorted_img = tf.image.per_image_standardization(distorted_img)

    return distorted_img, label
```

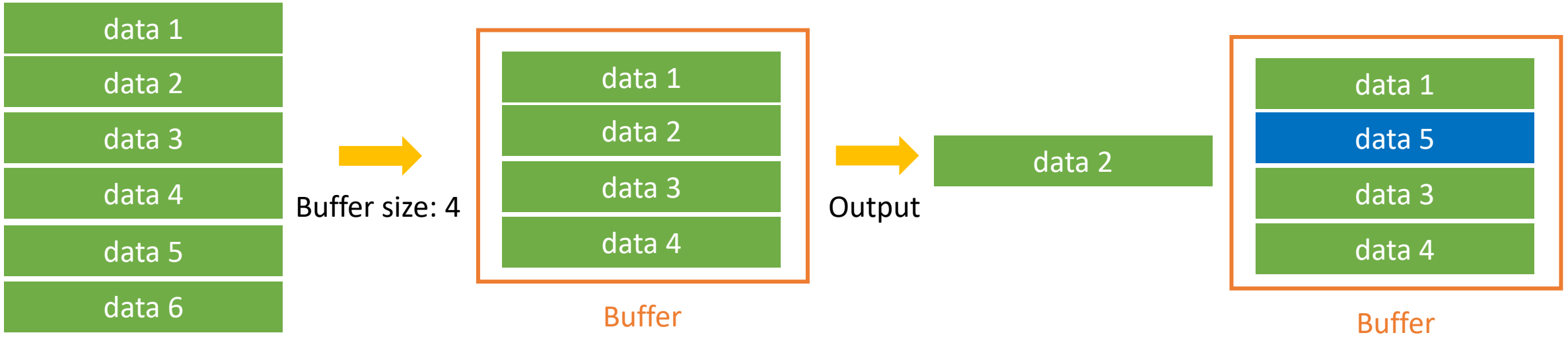


Input Pipeline - TensorFlow

- Transformations
  - Shuffle: maintains a fixed-size buffer and chooses the next element uniformly at random from that buffer

Buffer size

```
dataset = raw_dataset.shuffle(16)
```



Input Pipeline - TensorFlow

- Transformations
  - Batch: combines consecutive elements of this dataset into batches

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

  - Be careful that if you apply shuffle after batch, you'll get shuffled batch but data in a batch remains the same

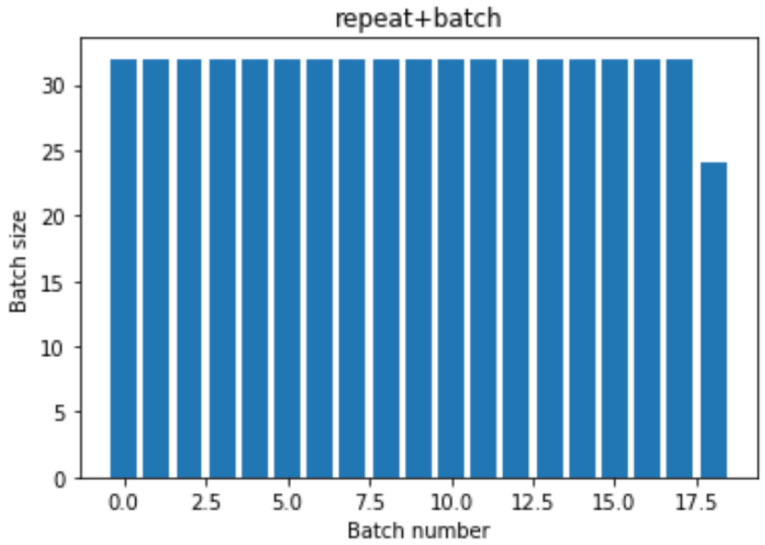
shuffle -> batch

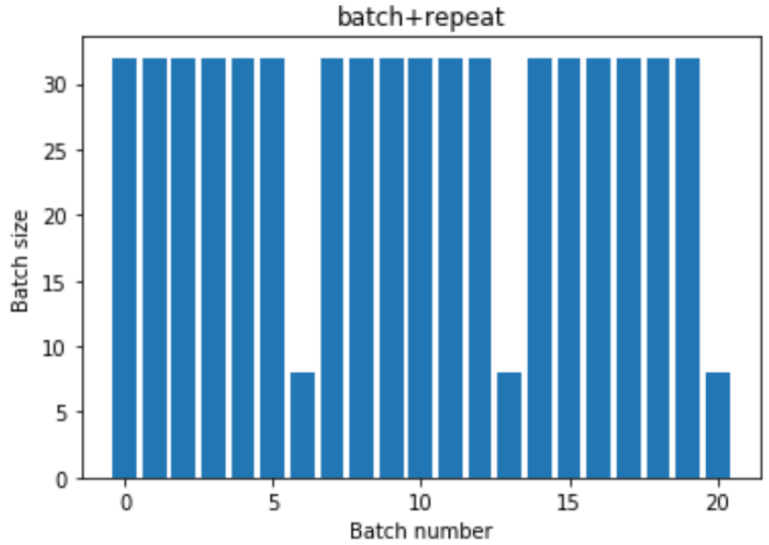
batch -> shuffle

Input Pipeline - TensorFlow

- Transformations
  - Repeat: repeat this dataset so each original value is seen multiple times

```
dataset = dataset.repeat(2)
```





Input Pipeline - TensorFlow

- Transformations
  - Prefetch: allows later elements to be prepared while the current element is being processed
  - This often improves latency and throughput, at the cost of using additional memory to store prefetched elements

```
dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

it will prompt the `tf.data` runtime to tune the value dynamically at runtime

Input Pipeline - TensorFlow

- Transformations
  - Prefetch: allows later elements to be prepared while the current element is being processed
  - This often improves latency and throughput, at the cost of using additional memory to store prefetched elements

```
dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

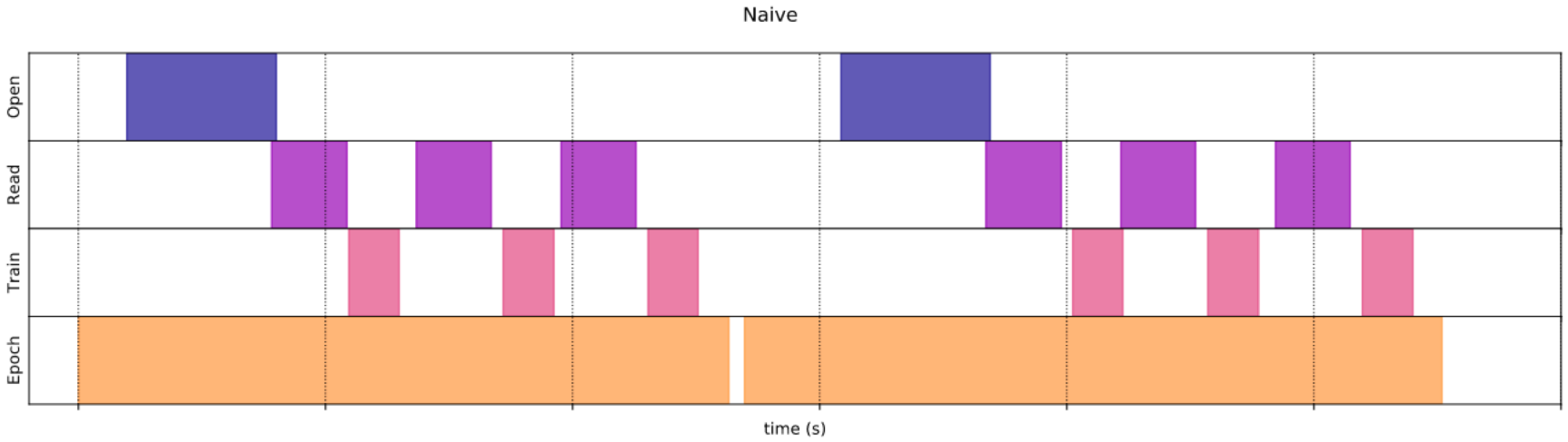
```
def preprocess_function(one_row_a, one_b):
    """
        Input: one slice of the dataset
        Output: modified slice
    """
    # Do some data preprocessing, you can also input filenames and load data in here
    # Here, we transform each row of raw_data_a to its sum and mean
    one_row_a = [tf.reduce_sum(one_row_a), tf.reduce_mean(one_row_a)]

    return one_row_a, one_b
raw_dataset = raw_dataset.map(preprocess_function, num_parallel_calls=tf.data.experimental.AUTOTUNE)
```

Input Pipeline - TensorFlow

- Transformations
  - Prefetch: allows later elements to be prepared while the current element is being processed

```
dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

it will prompt the `tf.data` runtime to tune the value dynamically at runtime

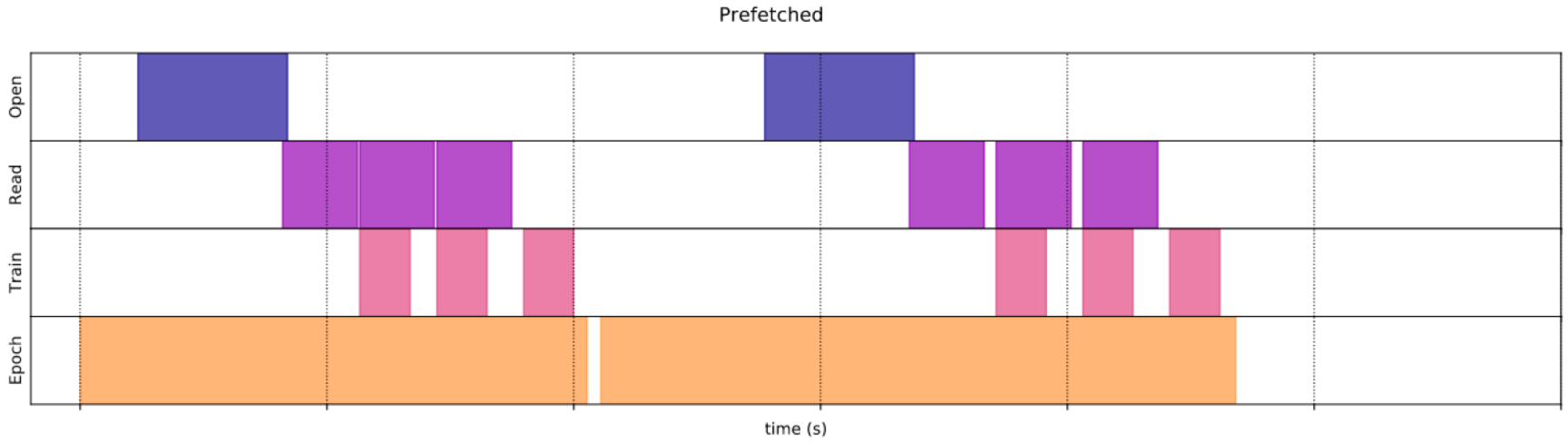


Input Pipeline - TensorFlow

- Transformations
  - Prefetch: allows later elements to be prepared while the current element is being processed

```
dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

it will prompt the `tf.data` runtime to tune the value dynamically at runtime



Input Pipeline - TensorFlow

- Now you can iterate through the data and train
- Aware that if you do batch, the first dimension will be the batch size

```
for img, label in dataset_train.take(1):
    print(img.shape)
    print(label.shape)
```

```
(32, 224, 224, 3)
(32,)
```

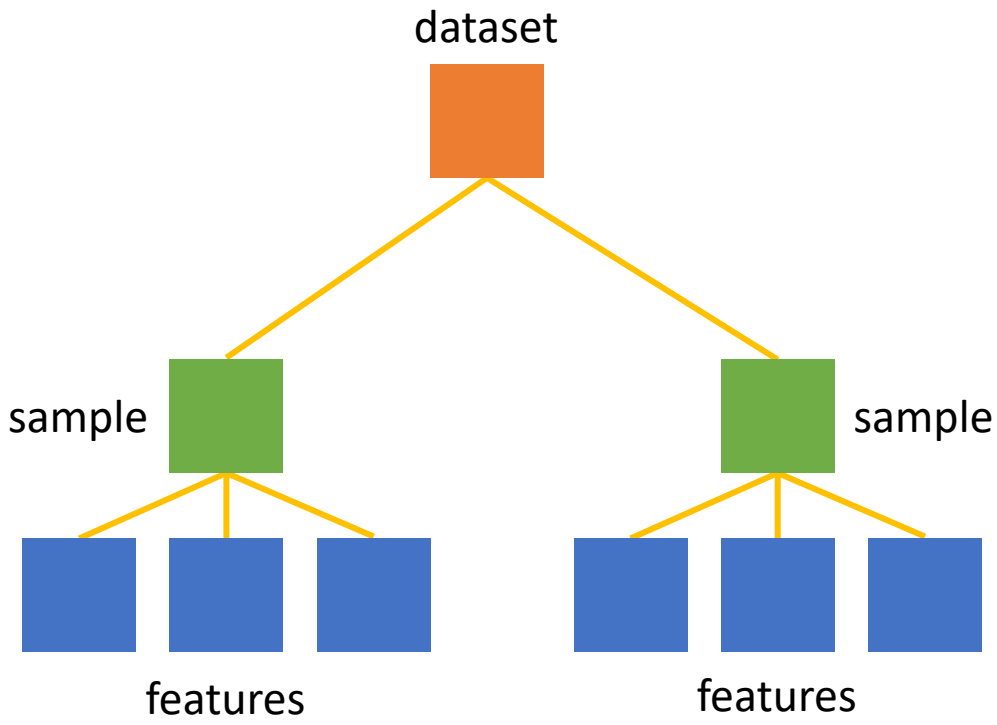
Outline

- Convolution neural network
- Input pipeline
- TFRecord

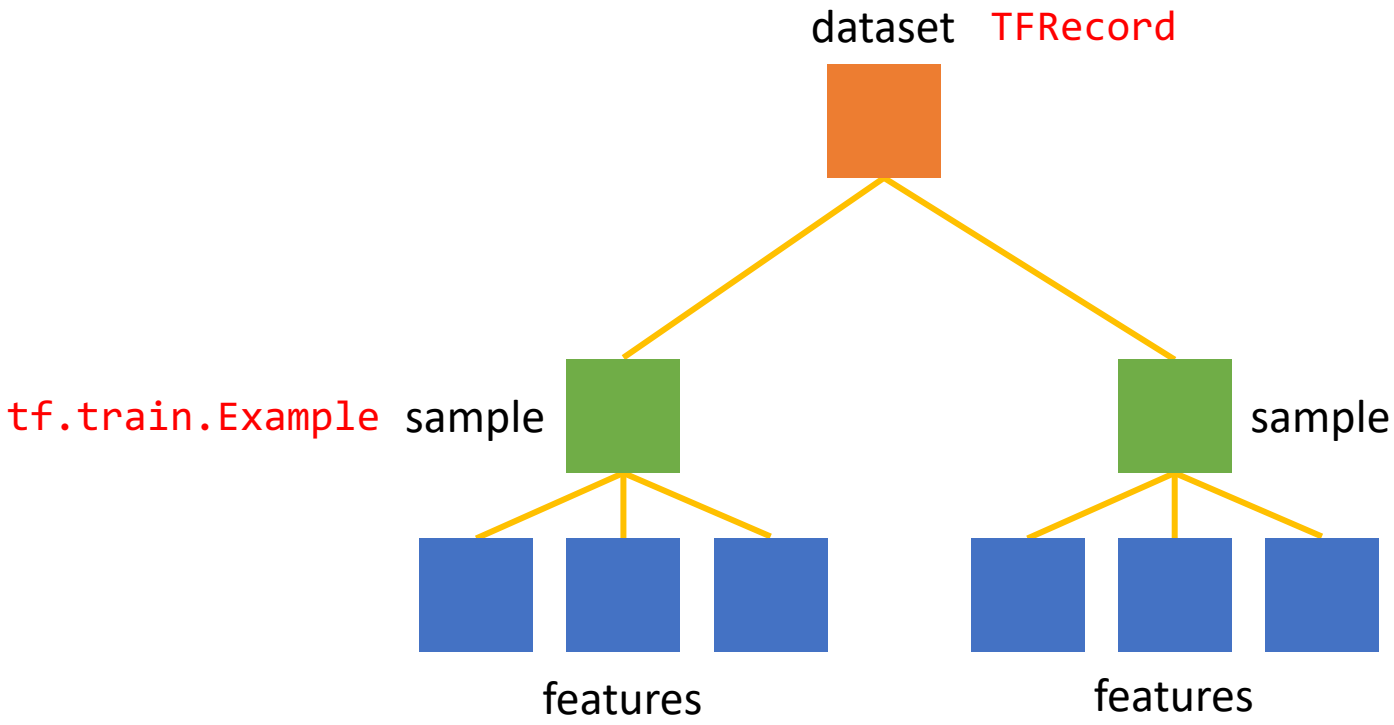
TFRecord

- TFRecord format is a simple format for storing a sequence of binary records
- Advantages
  - Binary data takes up less space on disk, takes less time to copy and can be read much more efficiently from disk
  - Only the data that is required at the time (e.g. a batch) is loaded from disk and then processed
- In this lab, we are going to store `tf.train.Example` messages in TFRecord

TFRecord



TFRecord


dataset
TFRecord
tf.train.Example
sample
sample
features
features
tf.train.Feature -> tf.train.Features

TFRecord

```
# dataset.tfrecords
[
    {   # example 1 (tf.train.Example)
        'feature_1': tf.train.Feature,
        ...
        'feature_k': tf.train.Feature
    },
    ...
    {   # example N (tf.train.Example)
        'feature_1': tf.train.Feature,
        ...
        'feature_k': tf.train.Feature
    }
]
```

TFRecord

- Feature - `tf.train.Feature` can accept one of the following three type
  - `tf.train.ByteList`: string and byte
  - `tf.train.FloatList`: float and double
  - `tf.train.Int64List`: bool, enum, int..etc.

```
def _bytes_feature(value):
  """Returns a bytes_list from a string / byte."""
  if isinstance(value, type(tf.constant(0))):
    value = value.numpy() # BytesList won't unpack a string from an EagerTensor.
  return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))

def _float_feature(value):
  """Returns a float_list from a float / double."""
  return tf.train.Feature(float_list=tf.train.FloatList(value=[value]))

def _int64_feature(value):
  """Returns an int64_list from a bool / enum / int / uint."""
  return tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))
```

TFRecord

- Feature - `tf.train.Feature`

```
bytes_list {
  value: "test_bytes"
}

float_list {
  value: 2.7182817459106445
}

int64_list {
  value: 1
}
```

TFRecord

- Sample - `tf.train.Example`

```
def serialize_example(feature0, feature1, feature2, feature3):
    """
    Creates a tf.Example message ready to be written to a file.
    """
    # Create a dictionary mapping the feature name to the tf.Example-compatible data type.
    feature = {
        'feature0': _int64_feature(feature0),
        'feature1': _int64_feature(feature1),
        'feature2': _bytes_feature(feature2),
        'feature3': _float_feature(feature3),
    }

    # Create a Features message using tf.train.Example.
    example_proto = tf.train.Example(features=tf.train.Features(feature=feature))

    return example_proto.SerializeToString()
```

Create a dictionary

Group the features

serialized to a binary-string

```
b'\x12\x06\n\x04T\xf8-@'
```

TFRecord

- Sample - `tf.train.Example`

```
features {
  feature {
    key: "feature0"
    value {
      int64_list {
        value: 0
      }
    }
  }
  feature {
    key: "feature1"
    value {
      int64_list {
        value: 4
      }
    }
  }
}
```

feature

features

TFRecord

- Write TFRecord - `tf.data`

```
features_dataset = tf.data.Dataset.from_tensor_slices((feature0, feature1, feature2, feature3))
features_dataset
```

```
<TensorSliceDataset shapes: ((), (), (), ()), types: (tf.bool, tf.int64, tf.string, tf.float64)>
```

TFRecord

- Write TFRecord - `tf.data`

```
features_dataset = tf.data.Dataset.from_tensor_slices((feature0, feature1, feature2, feature3))
features_dataset
```

```
<TensorSliceDataset shapes: ((), (), (), ()), types: (tf.bool, tf.int64, tf.string, tf.float64)>
```

```
def tf_serialize_example(f0,f1,f2,f3):
    tf_string = tf.py_function(
        serialize_example,
        (f0,f1,f2,f3),  # pass these args to the above function.
        tf.string)      # the return type is `tf.string`.
    return tf.reshape(tf_string, ()) # The result is a scalar
```

wraps a python function into a TensorFlow operation

```
filename = 'test.tfrecord'
writer = tf.data.experimental.TFRecordWriter(filename)
writer.write(serialized_features_dataset)
```

TFRecord

- Write TFRecord - python

```
# Write the `tf.Example` observations to the file.
with tf.io.TFRecordWriter(filename) as writer:
    for i in range(n_samples_):
        example = serialize_example(feature0[i], feature1[i], feature2[i], feature3[i])
        writer.write(example)
```

TFRecord

- Read TFRecord - `tf.data`

```
filenames = [filename]
raw_dataset = tf.data.TFRecordDataset(filenames)
raw_dataset
```

```
for raw_record in raw_dataset.take(3):
    print(repr(raw_record))
```

```
<tf.Tensor: shape=(), dtype=string, numpy=b'\nU\n\x11\n\x08feature0\x12\x05\x1a\x03\n\x01\x00\n\x11\n\x08feature1\x12
\x05\x1a\x03\n\x01\x02\n\x17\n\x08feature2\x12\x0b\n\t\n\x07chicken\n\x14\n\x08feature3\x12\x08\x12\x06\n\x04\x0eX\xe
f\xbe'>
<tf.Tensor: shape=(), dtype=string, numpy=b'\nQ\n\x14\n\x08feature3\x12\x08\x12\x06\n\x04\\\x10\x86?\n\x11\n\x08featu
re0\x12\x05\x1a\x03\n\x01\x01\n\x11\n\x08feature1\x12\x05\x1a\x03\n\x01\x01\n\x13\n\x08feature2\x12\x07\n\x05\n\x03do
g'>
<tf.Tensor: shape=(), dtype=string, numpy=b'\nU\n\x11\n\x08feature1\x12\x05\x1a\x03\n\x01\x02\n\x17\n\x08feature2\x12
\x0b\n\t\n\x07chicken\n\x14\n\x08feature3\x12\x08\x12\x06\n\x04\x00a6\xbe\n\x11\n\x08feature0\x12\x05\x1a\x03\n\x01\x
00'>
```

TFRecord

- Read TFRecord - `tf.data`

```
filenames = [filename]
raw_dataset = tf.data.TFRecordDataset(filenames)
raw_dataset
```

This description can help building shape and type

```
# Create a description of the features.
feature_description = {
    'feature0': tf.io.FixedLenFeature([], tf.int64, default_value=0),
    'feature1': tf.io.FixedLenFeature([], tf.int64, default_value=0),
    'feature2': tf.io.FixedLenFeature([], tf.string, default_value=''),
    'feature3': tf.io.FixedLenFeature([], tf.float32, default_value=0.0),
}

def _parse_function(example_proto):
  # Parse the input `tf.Example` proto using the dictionary above.
  return tf.io.parse_single_example(example_proto, feature_description)
```

```
for parsed_record in parsed_dataset.take(1):
    print(repr(parsed_record))
```

```
{'feature0': <tf.Tensor: shape=(), dtype=int64, numpy=0>, 'feature1': <tf.Tensor: shape=(), dtype=int64, numpy=2>, 'f
eature2': <tf.Tensor: shape=(), dtype=string, numpy=b'chicken'>, 'feature3': <tf.Tensor: shape=(), dtype=float32, num
py=-0.46746868>}
```

TFRecord

- Read TFRecord - python

```
filenames = [filename]
raw_dataset = tf.data.TFRecordDataset(filenames)
raw_dataset
```

```
for raw_record in raw_dataset.take(1):
    example = tf.train.Example()
    example.ParseFromString(raw_record.numpy())
    print(example)
```

```
features {
  feature {
    key: "feature0"
    value {
      int64_list {
        value: 0
      }
    }
  }
  feature {
    key: "feature1"
    value {
      int64_list {
        value: 4
      }
    }
  }
}
```

Assignment

- Goal
  - Write the training/testing dataset to TFRecord files
  - Read data from TFRecord files
  - Build the input pipeline, and train the model for at least 5 epochs
- Deadline
  - 2020/11/12 (Thur) 23:59