Lab 11-1 Convolution Neural Network & Data Pipelines

DataLab NTHU, 2022

Outline

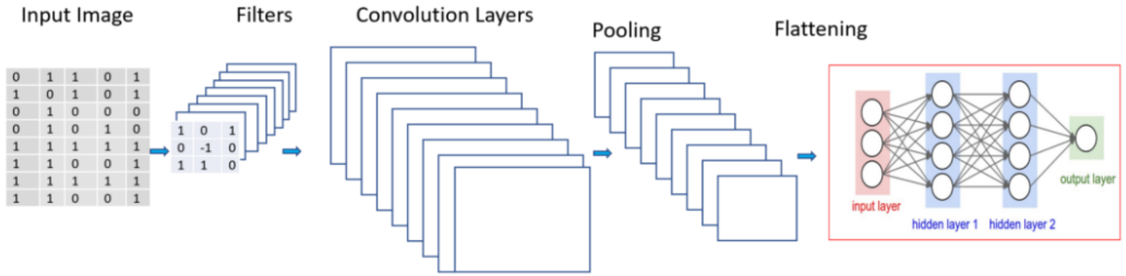
- Convolution neural network
- Input pipeline
- Optimization for Input pipeline

Outline

- Convolution neural network
- Input pipeline
- Optimization for Input pipeline

Convolution Neural Network

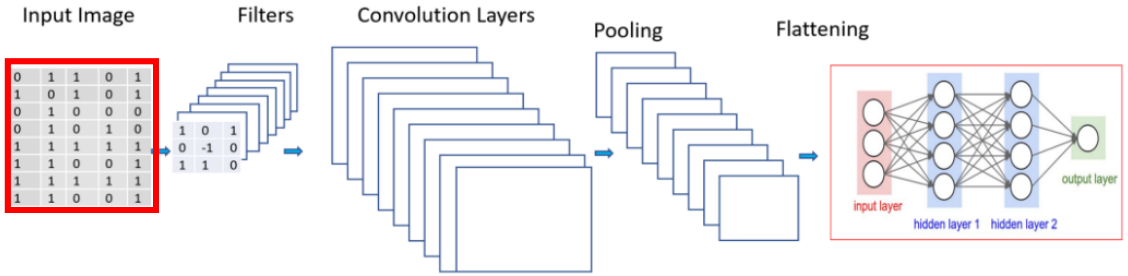
- Build a CNN model via Sequential API
  - A stack of Conv2D and MaxPooling2D layers




```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), strides=(1,1), padding='valid', activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(2,2), padding='valid', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu'))
```

Convolution Neural Network

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

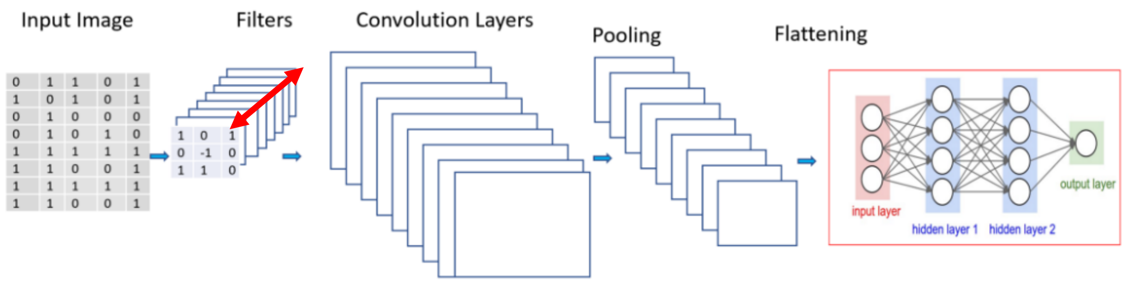


Width * Height * Channel

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), strides=(1,1), padding='valid', activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(2,2), padding='valid', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu'))
```

Convolution Neural Network

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

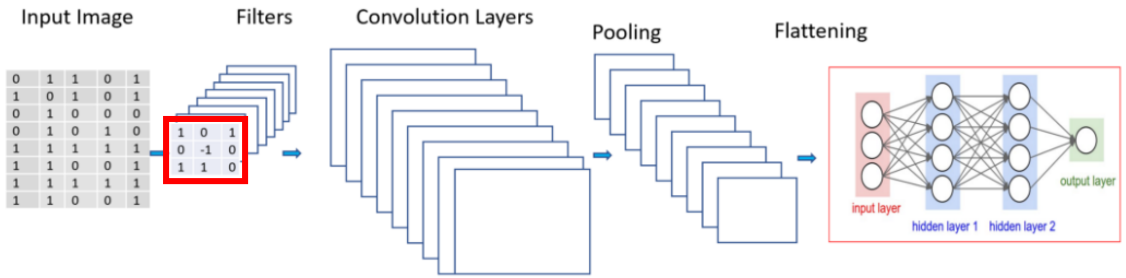


Filters: the number of output filters in the convolution

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), strides=(1,1), padding='valid', activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(2,2), padding='valid', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu'))
```

Convolution Neural Network

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



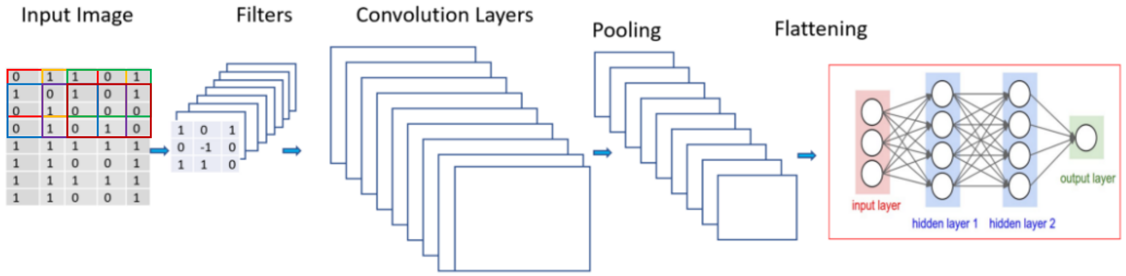
Kernel size: specifying the height and width of the 2D convolution window

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), strides=(1,1), padding='valid', activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(2,2), padding='valid', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu'))
```

Convolution Neural Network

```
Layer (type)                 Output Shape
=================================================
conv2d_28 (Conv2D)           (None, 26, 26, 32)

max_pooling2d_12 (MaxPoolin  (None, 13, 13, 32)
g2D)

conv2d_29 (Conv2D)           (None, 6, 6, 64)

max_pooling2d_13 (MaxPoolin  (None, 3, 3, 64)
g2D)

conv2d_30 (Conv2D)           (None, 3, 3, 64)
```

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



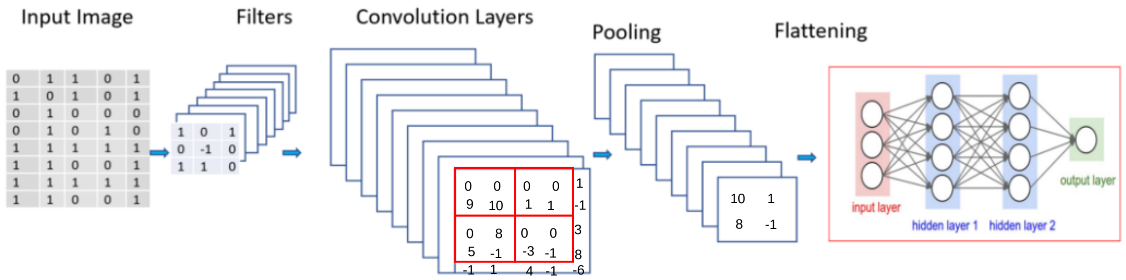
(Default value)

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), strides=(1,1), padding='valid', activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(2,2), padding='valid', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu'))
```

Convolution Neural Network

```
Layer (type)                 Output Shape
=================================================
conv2d_28 (Conv2D)           (None, 26, 26, 32)

max_pooling2d_12 (MaxPoolin  (None, 13, 13, 32)
g2D)

conv2d_29 (Conv2D)           (None, 6, 6, 64)

max_pooling2d_13 (MaxPoolin  (None, 3, 3, 64)
g2D)

conv2d_30 (Conv2D)           (None, 3, 3, 64)
```

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

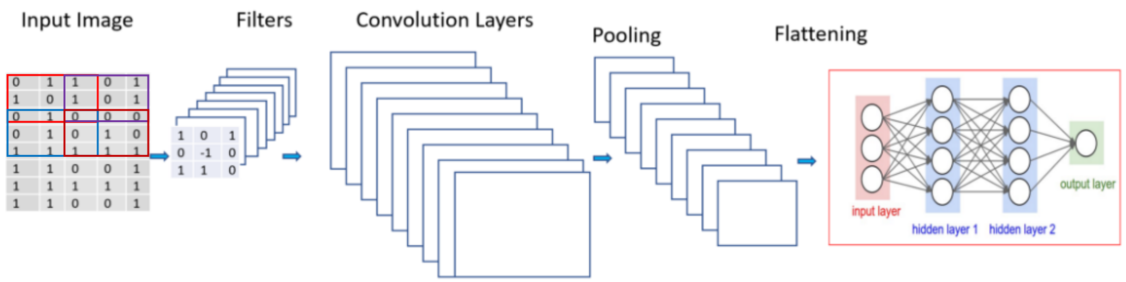



```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), strides=(1,1), padding='valid', activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(2,2), padding='valid', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu'))
```

Convolution Neural Network

```
Layer (type)                 Output Shape
=================================================
conv2d_28 (Conv2D)           (None, 26, 26, 32)

max_pooling2d_12 (MaxPoolin  (None, 13, 13, 32)
g2D)

conv2d_29 (Conv2D)           (None, 6, 6, 64)

max_pooling2d_13 (MaxPoolin  (None, 3, 3, 64)
g2D)

conv2d_30 (Conv2D)           (None, 3, 3, 64)
```

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

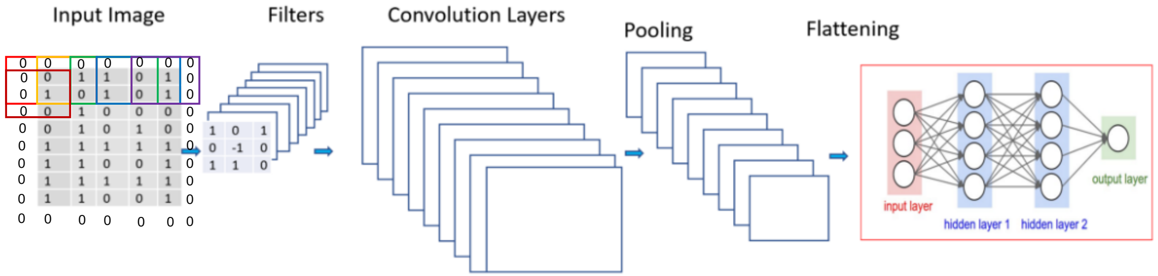



```python
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), strides=(1,1), padding='valid', activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(2,2), padding='valid', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu'))
```

Convolution Neural Network

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

Layer (type)	Output Shape
conv2d_28 (Conv2D)	(None, 26, 26, 32)
max_pooling2d_12 (MaxPooling2D)	(None, 13, 13, 32)
conv2d_29 (Conv2D)	(None, 6, 6, 64)
max_pooling2d_13 (MaxPooling2D)	(None, 3, 3, 64)
conv2d_30 (Conv2D)	(None, 3, 3, 64)




```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), strides=(1,1), padding='valid', activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(2,2), padding='valid', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), strides=(1,1), padding='same', activation='relu'))
```

When padding="same" and strides=1, the output has the same size as the input.

Outline

- Convolution neural network
- Input pipeline
- Optimization for Input pipeline

Input Pipeline

- A series of input data processing before training

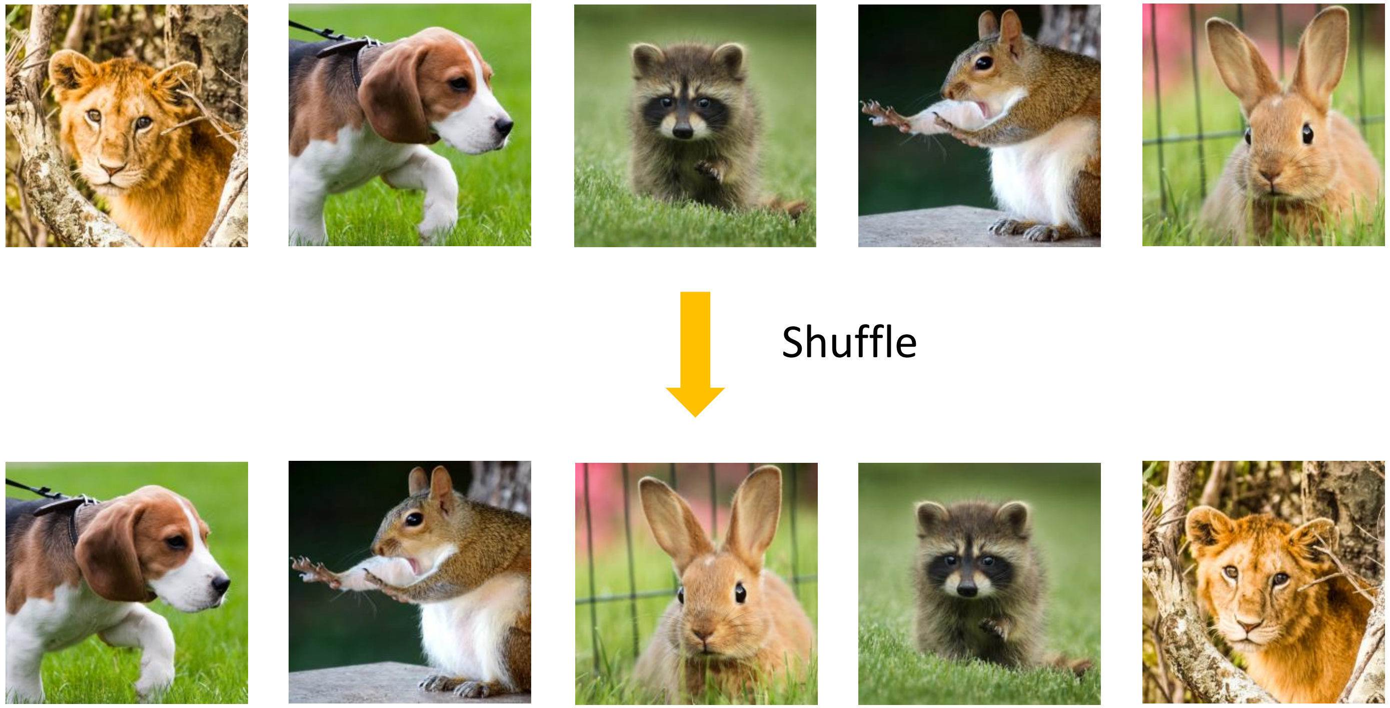
Input Pipeline

- A series of input data processing before training

Rotat
e

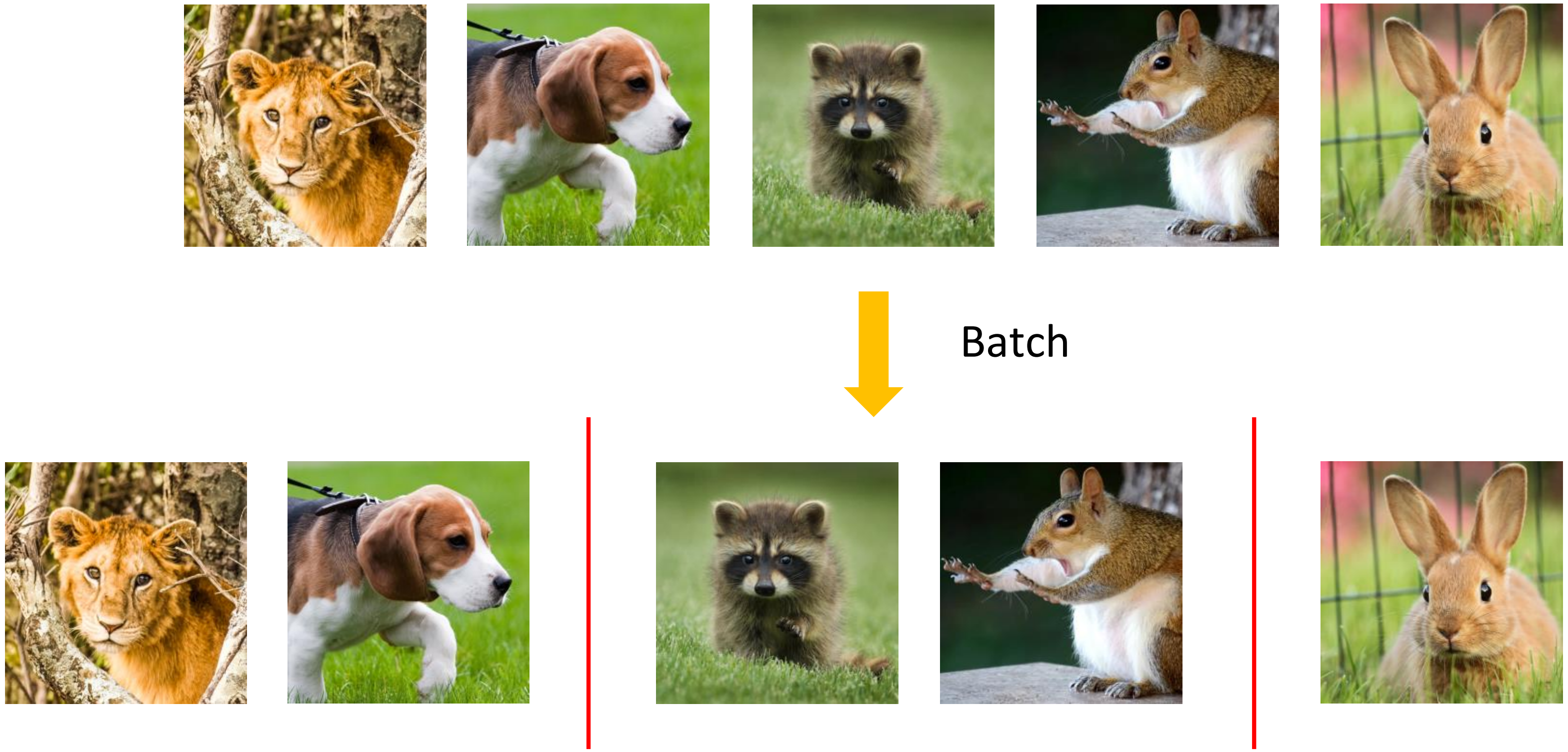
Input Pipeline

- 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**)

raw_data_a: [[0.59004802, 0.68869704, 0.67771658, 0.25277111, 0.44878355],
[0.2194635 , 0.23323033, 0.37668097, 0.0523581 , 0.84413446],
⋮
[0.2194635 , 0.23323033, 0.37668097, 0.0523581 , 0.84413446],
[0.94882014, 0.3818479 , 0.93550471, 0.23102154, 0.66095901]]

raw_data_b: [0, 1, 2, … , 199]

```
# 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**)

```python
# 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

```python
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

```python
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

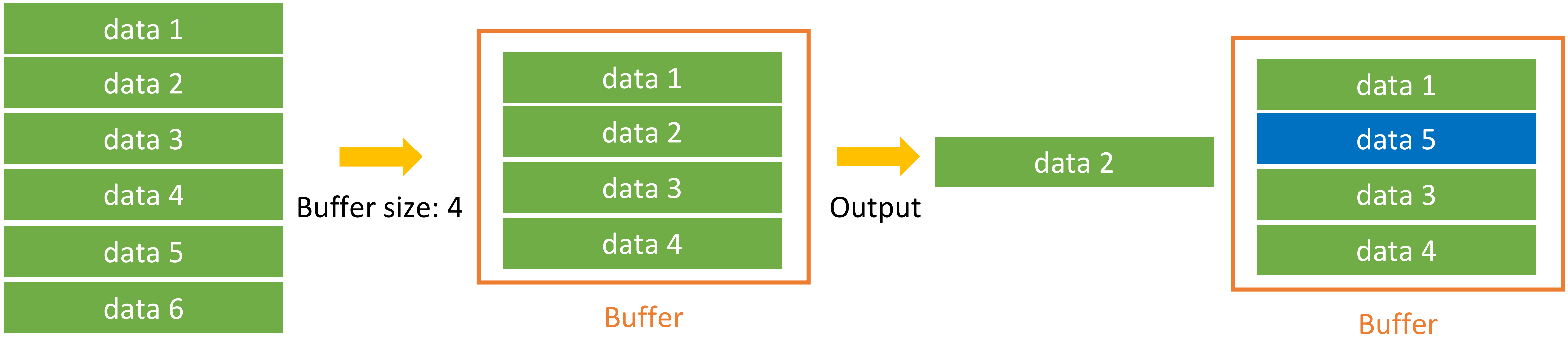
```
def pre_train_data(img, label):
    distorted_img = tf.image.random_crop(img, [IMAGE_SIZE_CROPPED,IMAGE_SIZE_CROPPED,IMAGE_DEPTH])
    distorted_img = tf.image.random_flip_left_right(distorted_img)
    distorted_img = tf.image.random_brightness(distorted_img, max_delta=63)
    distorted_img = tf.image.random_contrast(distorted_img,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

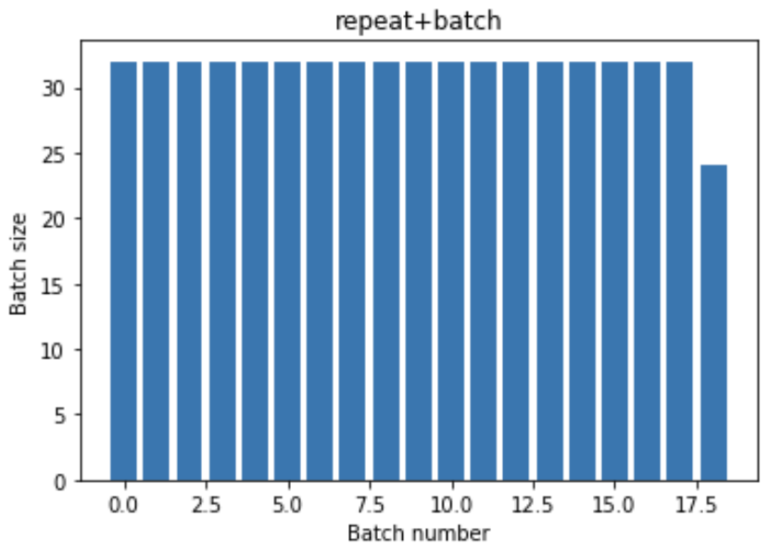
shuffle -> batch

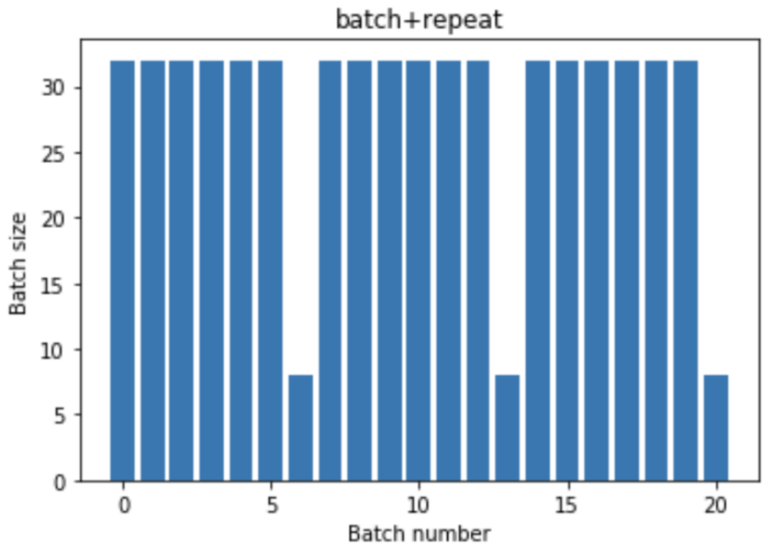
batch -> shuffle

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)
```

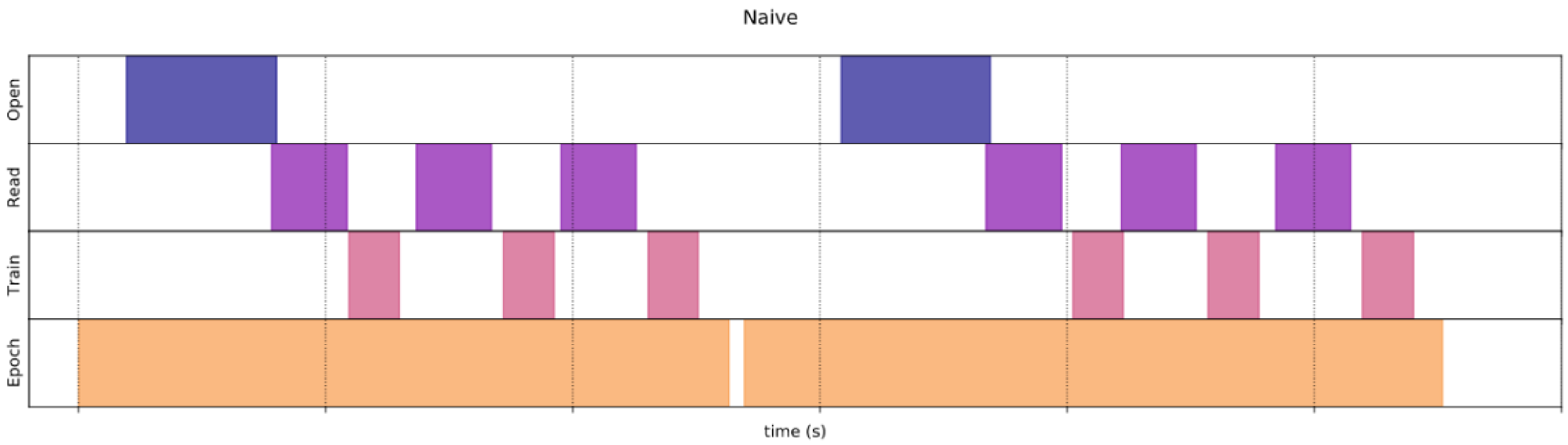
```
def preprocess_function(one_row_a, one_b):
    """
        Input: one slice of the dataset
        Output: modified slice
    """
    # Do some data preprocessing, you can also input filenames and load data in here
    # Here, we transform each row of raw_data_a to its sum and mean
    one_row_a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

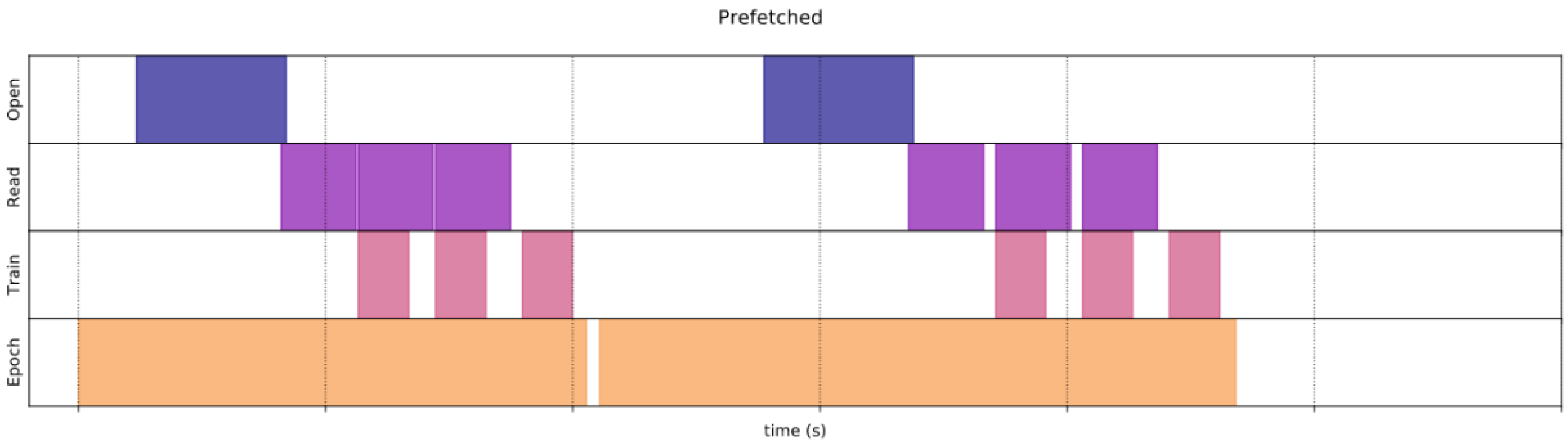


Input Pipeline - TensorFlow

- 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,)
```

Memory limited?

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

200 * 5 * 8 byte ~ 7.8KB

```
raw_data_a: [[0.59004802, 0.68869704, 0.67771658, 0.25277111, 0.44878355],
             [0.2194635 , 0.23323033, 0.37668097, 0.0523581 , 0.84413446],
                                        ⋮
             [0.2194635 , 0.23323033, 0.37668097, 0.0523581 , 0.84413446],
             [0.94882014, 0.3818479 , 0.93550471, 0.23102154, 0.66095901]]

raw_data_b: [ 0, 1, 2, … , 199 ]
```

```python
# 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

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

What if x_train are 4000 colored images with (height, width) =(1024,1024)?

Memory limited?

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

200 * 5 * 8 byte ~ 7.8KB

raw_data_a: [[0.59004802, 0.68869704, 0.67771658, 0.25277111, 0.44878355],
[0.2194635 , 0.23323033, 0.37668097, 0.0523581 , 0.84413446],
⋮
[0.2194635 , 0.23323033, 0.37668097, 0.0523581 , 0.84413446],
[0.94882014, 0.3818479 , 0.93550471, 0.23102154, 0.66095901]]

raw_data_b: [0, 1, 2, … , 199]

```
# number of samples
n_samples = 200

# an array with sha
raw_data_a = np.ran
# a list with lengt
raw_data_b = np.ara

# this tells the da
raw_dataset = tf.da
print(raw_dataset)
```

```
import numpy as np
a = np.zeros((4000, 1024, 1024, 3), dtype='float64')
a.nbytes
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
/tmp/ipykernel_27592/812912113.py in <module>
      1 import numpy as np
----> 2 a = np.zeros((4000, 1024, 1024, 3), dtype='float64')
      3 a.nbytes

MemoryError: Unable to allocate 93.8 GiB for an array with shape (4000, 1024, 1024, 3) and data type float64
```

(CPU Memory, irrelative to smaller batch size

```
<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))
```

What if x_train are 4000 colored images with (height, width) =(1024,1024)?

Memory limited?

- Use **image path** as x_train, rather than digits of pixels directly
  - Load digits during training (**prefetch**)

```
# loda images
def load_image(image_path, label):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_jpeg(img, channels=IMAGE_DEPTH)
    img = tf.image.resize(img, (IMAGE_HEIGHT, IMAGE_WIDTH))
    img = tf.cast(img, tf.float32)
    img = tf.divide(img,255.0)
    return img, label
```

```
# the dataset objects we prepared for you
dataset_train = tf.data.Dataset.from_tensor_slices((img_path_train,label_train))
dataset_train = dataset_train.map(load_image)

dataset_val = tf.data.Dataset.from_tensor_slices((img_path_val,label_val))
dataset_val = dataset_val.map(load_image)
```

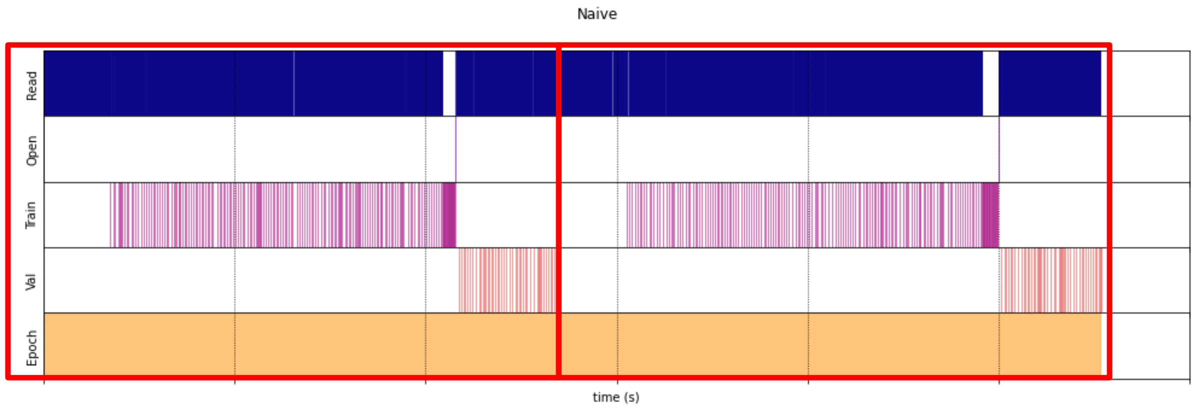
```
dataset_train = dataset_train.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

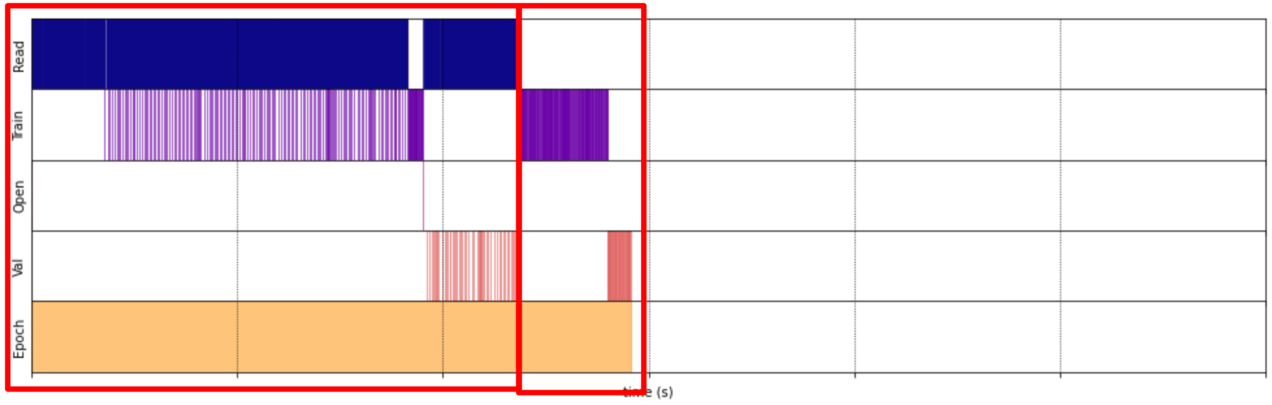
Outline

- Convolution neural network
- Input pipeline
- Optimization for Input pipeline

Optimization for Input pipeline

1. prefetching: overlaps the preprocessing and model execution of a training step.
2. Interleave (Parallelizing data extraction): parallelize the data loading step, interleaving the contents of other datasets (such as data file readers).
3. Parallel mapping: parallelized mapping across multiple CPU cores.
4. Caching: cache a dataset, save some operations (like file opening and data reading) from being executed during each epoch.
5. Vectorizing mapping: batch before map, so that mapping can be vectorized.





Assignment

- Goal
  - Try some the input transfromation mentioned above (e.g. shuffle, batch, repeat, map(random_crop, random_flip_left_right, ...)) but without optimization terms (e.g. prefetch, cache, num_parallel_calls), comparing the performance to the no input transfromation.
  - Retrain your model with optimized terms, compare the time consuming.
  - Training both models above for at least 3 epochs.
  - Briefly summarize what you did and explain the performance results (accuracy and time consuming).
- Deadline
  - 2022/11/6 (Sun) 23:59