Lab-12
Convolution Neural Network
& Data Pipelines

Outline

- MNIST
- Cifar10
- Input Pipeline
- CNN Model for CIFAR 10
- TFRecords

MNIST

- Handwritten digits
- Size: 28x28 pixels

Training data	Testing data
60,000	10,000

MNIST

- Softmax Regression

```
model_1 = models.Sequential()

model_1.add(layers.Dense(10, activation='softmax',input_shape=(784,)))
```

- Convolutional Network

```
model_2 = models.Sequential()
model_2.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model_2.add(layers.MaxPooling2D((2, 2)))
model_2.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_2.add(layers.MaxPooling2D((2, 2)))
model_2.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_2.add(layers.Flatten())
model_2.add(layers.Dense(64, activation='relu'))
model_2.add(layers.Dropout(0.5))
model_2.add(layers.Dense(10, activation='softmax'))
```

MNIST

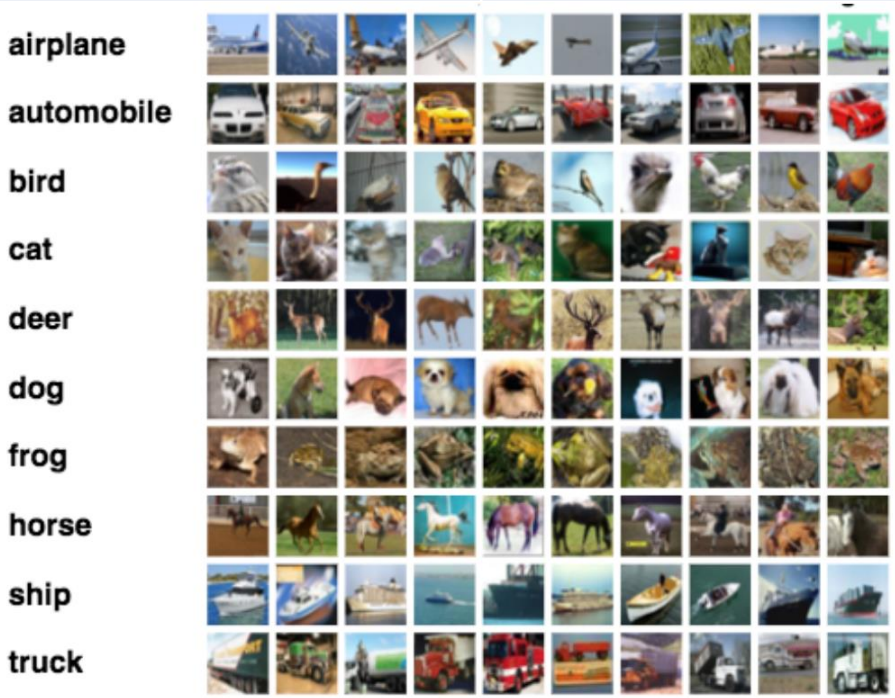
- Result

	Softmax Regression	Convolutional Network
Test accuracy	0.92	0.99

Cifar10

- Color images in 10 classes
- Size: 32x32 pixels

Training data	Testing data
50,000	10,000



Cifar10

- Result

	KNN	SVM	Convolutional Network
Test Accuracy	0.47	0.51	0.79

Input Pipeline

- A typical TensorFlow training input pipeline:
  - Extract: Read data from memory or persistent storage
  - Transform: Use CPU to parse and perform preprocessing operations on the data
  - Load: Load the transformed data onto the accelerator devices

Input Pipeline

- tf.data API
  - Define data source and initialize your Dataset object
  - Apply transformations on the dataset
  - Consume the elements of Dataset object

Input Pipeline

- Construct your Dataset
  - To construct a **Dataset** from data in:
    - Memory: tf.data.Dataset.from_tensor_slices()
    - CSV: tf.data.experimental.make_csv_dataset()
    - Tfrecord: tf.data.TFRecordDataset()

```
# number of samples
n_observations1 = 200
# an array with shape (n_observations1, 5)
raw_data_a = np.random.rand(n_observations1, 5)
# a list with length of n_observations1 from 0 to
n_observations1-1
raw_data_b = np.arange(n_observations1)
raw_dataset = tf.data.Dataset.from_tensor_slices((raw_data_a, raw_data_b))
```

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

Input Pipeline

- Apply transformations
  - map
    - provide element-wise customized data preprocessing

```
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)
```

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

Input Pipeline

- Apply transformations
  - shuffle
    - maintains a fixed-size buffer and chooses the next element uniformly at random from that buffer

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

Input Pipeline

- Apply transformations
  - batch
    - stack batch_size elements together
    - Be careful that you should apply **Dataset.shuffle** before **Dataset.batch**.

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

Input Pipeline

- Apply transformations
  - repeat
    - allow you iterate over a dataset in multiple epochs

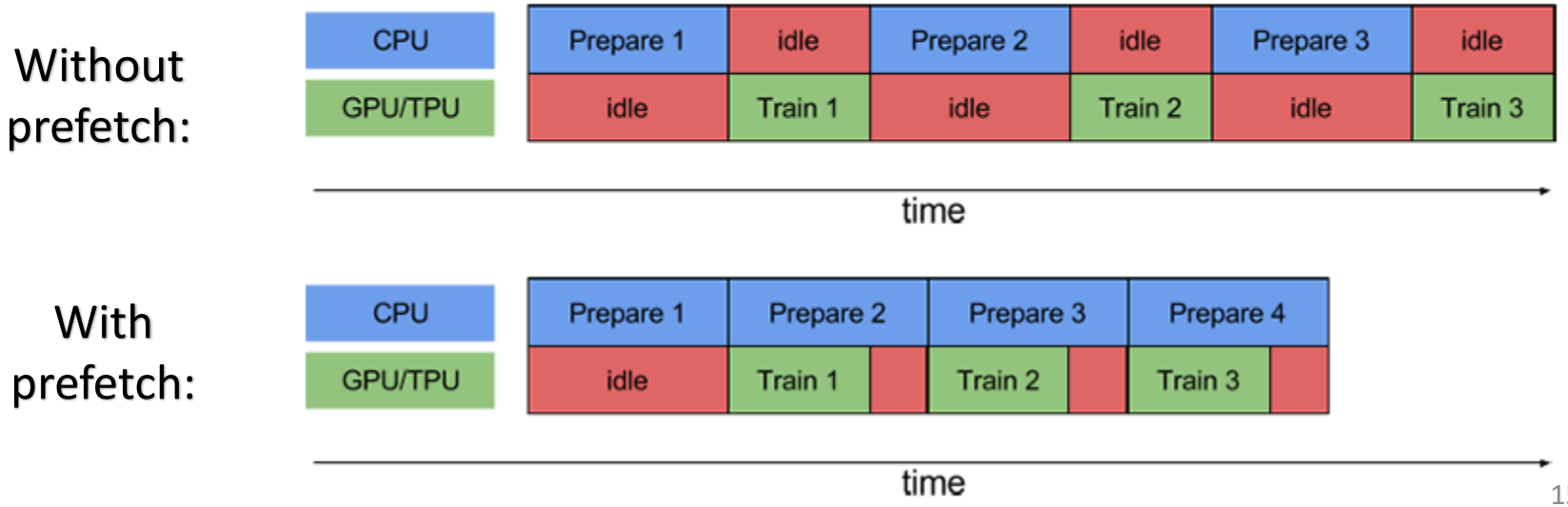
```
dataset = dataset.repeat(2)
```

```
epochs = 3

cus_dataset = raw_dataset.batch(32)
for epoch in range(epochs):
    size = 0
    for batch in cus_dataset:
        size += 1

    ("End of epoch: %d, Batch size of this epoch: %d"%(epoch, size))
```

Input Pipeline

- Apply transformations
  - prefetch
    - allow you decouple the time when data is produced from the time when data is consumed.

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



Input Pipeline

- Consume the elements
  - The **Dataset** object is a Python iterable.

```
# Here, we print the first 8 batches.
for i,elem in enumerate(dataset):
    print("Batch ", i, ", b are ", elem[1].numpy())
    if i==8:
        break
```

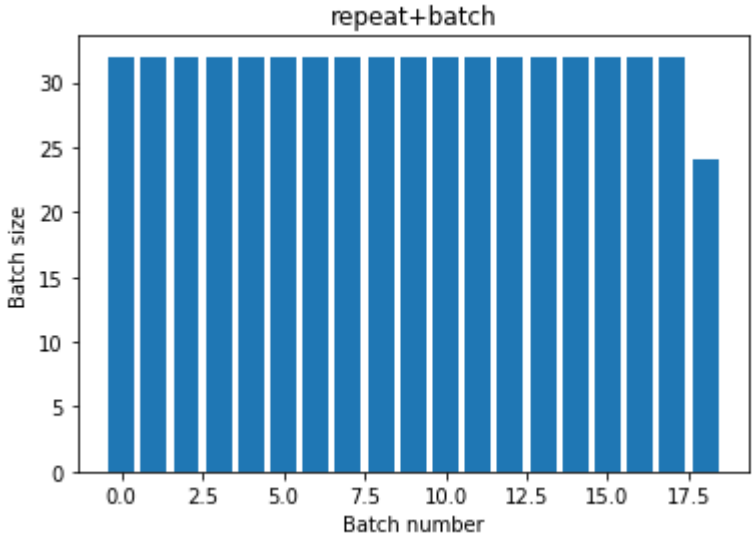
```
Batch  0 , b are  [2 0]
Batch  1 , b are  [5 3]
Batch  2 , b are  [ 8 16]
Batch  3 , b are  [ 4 12]
Batch  4 , b are  [11 14]
Batch  5 , b are  [13 18]
Batch  6 , b are  [25 10]
Batch  7 , b are  [27  7]
Batch  8 , b are  [22 15]
```

```
# Here, we print the first 8 batches.
it = iter(dataset)
for i in range(8):
    print("Batch ", i, ", b are ", next(it)[1].numpy())
```

```
Batch  0 , b are  [ 2 14]
Batch  1 , b are  [6 5]
Batch  2 , b are  [16  4]
Batch  3 , b are  [ 1 13]
Batch  4 , b are  [3 9]
Batch  5 , b are  [11 23]
Batch  6 , b are  [17 18]
Batch  7 , b are  [24 25]
```

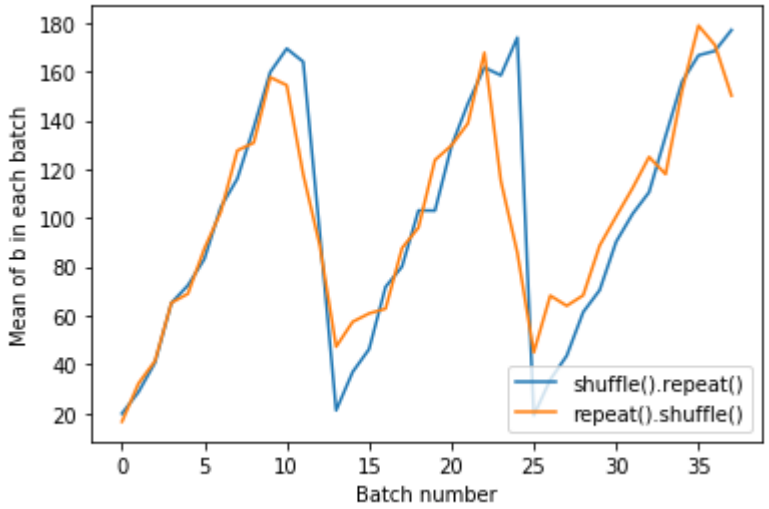
Input Pipeline

- repeat+batch / batch+repeat



Input Pipeline

- shufflt+repeat / repeat+shufflt



CNN Model for CIFAR 10

- Data augmentation

```
distorted_image = tf.image.random_crop(image, [IMAGE_SIZE_CROPPED,IMAGE_SIZE_CROPPED,3])
distorted_image = tf.image.random_flip_left_right(distorted_image)
distorted_image = tf.image.random_brightness(distorted_image, max_delta=63)
distorted_image = tf.image.random_contrast( distorted_image, lower=0.2, upper=1.8)
distorted_image = tf.image.per_image_standardization(distorted_image)
```

CNN Model for CIFAR 10

- Result

	KNN	SVM	Convolutional Network (without Data augmentation)	Convolutional Network (with Data augmentation)
Test Accuracy	0.47	0.51	0.79	0.84

Using TFRecords

- tf.Example
  - A tf.Example is a {"string": tf.train.Feature} mapping
  - The tf.train.Feature message type can accept one of the following three types:

1. tf.train.BytesList (the following types can be coerced)
   - string
   - byte
2. tf.train.FloatList (the following types can be coerced)
   - float (float32)
   - double (float64)
3. tf.train.Int64List (the following types can be coerced)
   - bool
   - enum
   - int32
   - uint32
   - int64
   - uint64

Using TFRecords

- tf.Example
  - The tf.train.Feature message type can accept one of the following three types:

```
# The following functions can be used to convert a value to a type compatible with tf.Example.

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]))
```

Using TFRecords

- Example
  - a sample consisting of 10,000 independently and identically distributed observations

```
# The number of observations in the dataset.
n_observations2 = int(1e4)

# Boolean feature, encoded as False or True.
feature0 = np.random.choice([False, True], n_observations2)

# Integer feature, random from 0 to 4.
feature1 = np.random.randint(0, 5, n_observations2)

# String feature
strings = np.array([b'cat', b'dog', b'chicken', b'horse', b'goat'])
feature2 = strings[feature1]

# Float feature, from a standard normal distribution
feature3 = np.random.randn(n_observations2)
```

Using TFRecords

- Example
  - Each of these features can be coerced into a tf.Example-compatible type
  - create a tf.Example message from these encoded features

```
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()
```

Using TFRecords

- Example
  - TFRecord files using tf.data
    - Writing a TFRecord file
      - tf.data.experimental.TFRecordWriter()
    - Reading a TFRecord file
      - tf.data.TFRecordDataset()
      - Parse by using the function below

```
# 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)
```

Using TFRecords

- Example
  - TFRecord files in Python
    - Writing a TFRecord file
      - tf.io.TFRecordWriter()
    - Reading a TFRecord file
      - tf.data.TFRecordDataset()
      - tf.train.Example()

```
example = tf.train.Example()
example.ParseFromString(raw_record.numpy())
```