

텐서플로에서 데이터 적재와 전처리 하기

정인호

Data API

여러가지 method

```
[ ] dataset = tf.data.Dataset.range(10)
```

```
▶ for item in dataset:  
    print(item)
```

```
tf.Tensor(0, shape=(), dtype=int64)  
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)  
tf.Tensor(7, shape=(), dtype=int64)  
tf.Tensor(8, shape=(), dtype=int64)  
tf.Tensor(9, shape=(), dtype=int64)
```

```
[ ] dataset = dataset.repeat(3).batch(7)  
for item in dataset:  
    print(item)
```

```
tf.Tensor([0 1 2 3 4 5 6], shape=(7,), dtype=int64)  
tf.Tensor([7 8 9 0 1 2 3], shape=(7,), dtype=int64)  
tf.Tensor([4 5 6 7 8 9 0], shape=(7,), dtype=int64)  
tf.Tensor([1 2 3 4 5 6 7], shape=(7,), dtype=int64)  
tf.Tensor([8 9], shape=(2,), dtype=int64)
```



Data API

여러가지 method

```
[ ] dataset = dataset.map(lambda x: x * 2)
```

```
▶ for item in dataset:  
    print(item)
```

```
Ⓐ tf.Tensor([ 0  2  4  6  8 10 12], shape=(7,), dtype=int64)  
tf.Tensor([14 16 18  0  2  4  6], shape=(7,), dtype=int64)  
tf.Tensor([ 8 10 12 14 16 18  0], shape=(7,), dtype=int64)  
tf.Tensor([ 2  4  6  8 10 12 14], shape=(7,), dtype=int64)  
tf.Tensor([16 18], shape=(2,), dtype=int64)
```

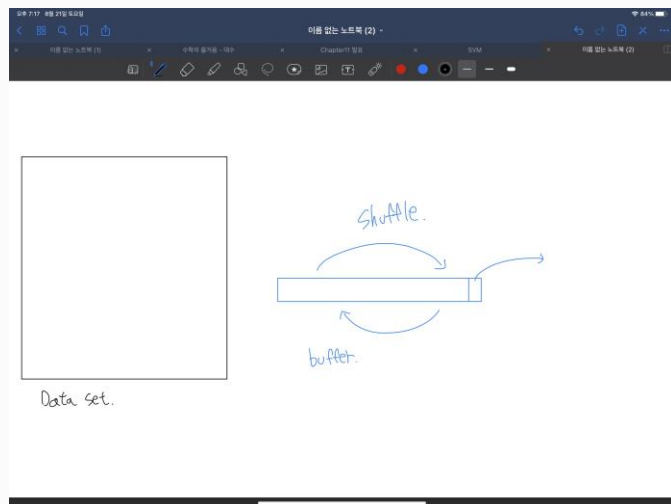
```
[ ] dataset = dataset.unbatch()
```

```
[ ] dataset = dataset.filter(lambda x: x < 10) # keep only items < 10
```

```
[ ] for item in dataset.take(3):  
    print(item)
```

```
tf.Tensor(0, shape=(), dtype=int64)  
tf.Tensor(2, shape=(), dtype=int64)  
tf.Tensor(4, shape=(), dtype=int64)
```

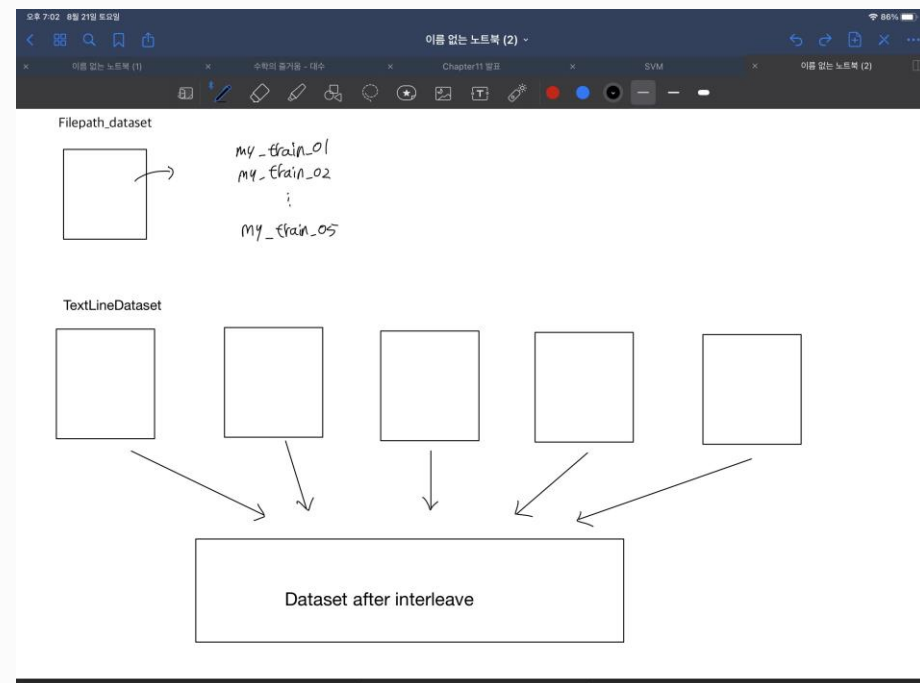
Data Shuffle



```
[ ] n_readers = 5
    dataset = filepath_dataset.interleave(
        lambda filepath: tf.data.TextLineDataset(filepath).skip(1),
        cycle_length=n_readers)
```

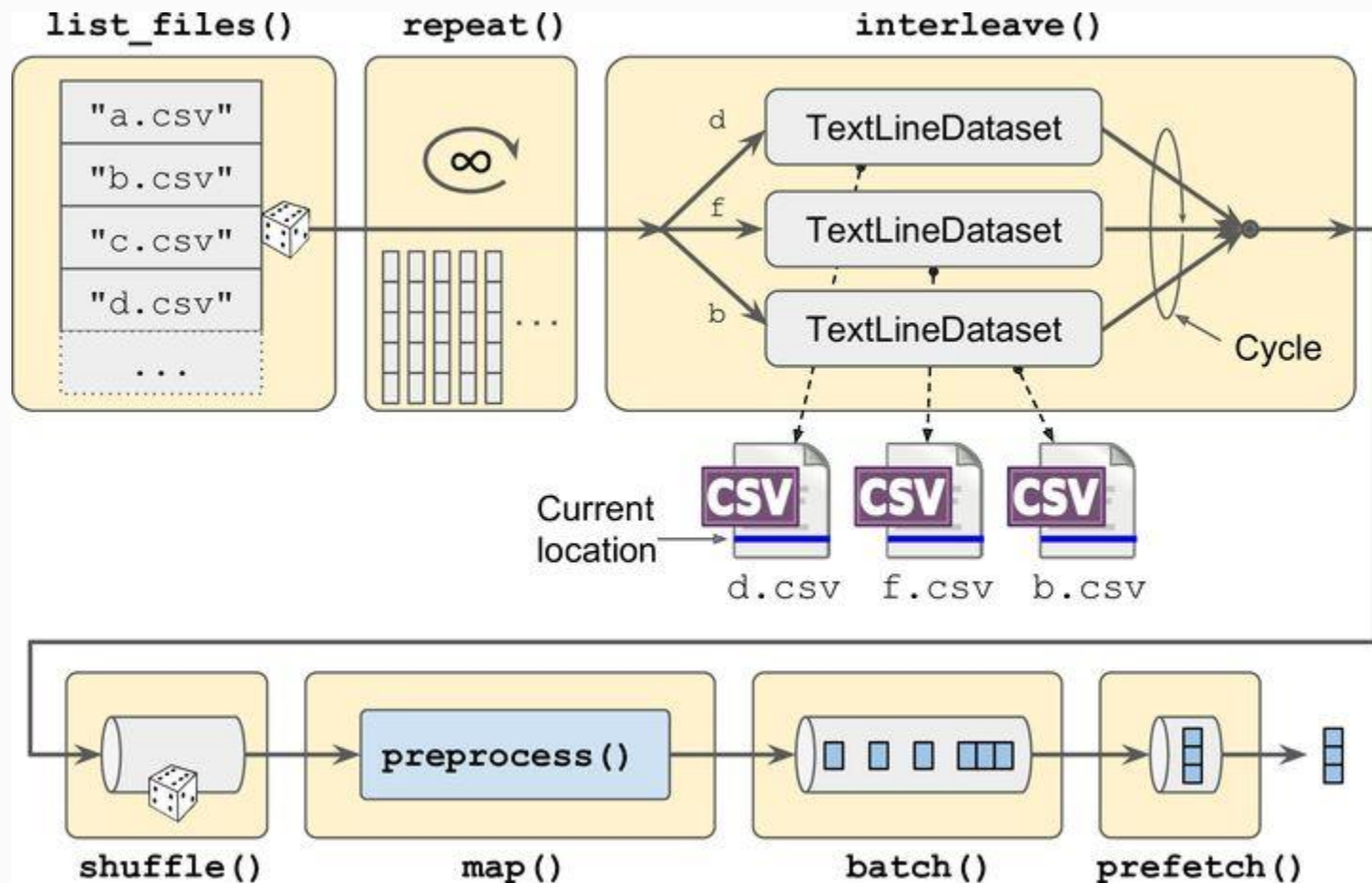
```
[ ] for line in dataset.take(5):
    print(line.numpy())
```

```
b'4.6477,38.0,5.03728813559322,0.911864406779661,745.0,2.5254237288135593,32.64,-117.07,1.504'
b'8.72,44.0,6.163179916317992,1.0460251046025104,668.0,2.794979079497908,34.2,-118.18,4.159'
b'3.8456,35.0,5.461346633416459,0.9576059850374065,1154.0,2.8778054862842892,37.96,-122.05,1.598'
b'3.3456,37.0,4.514084507042254,0.9084507042253521,458.0,3.2253521126760565,36.67,-121.7,2.526'
b'3.6875,44.0,4.524475524475524,0.993006993006993,457.0,3.195804195804196,34.04,-118.15,1.625'
```



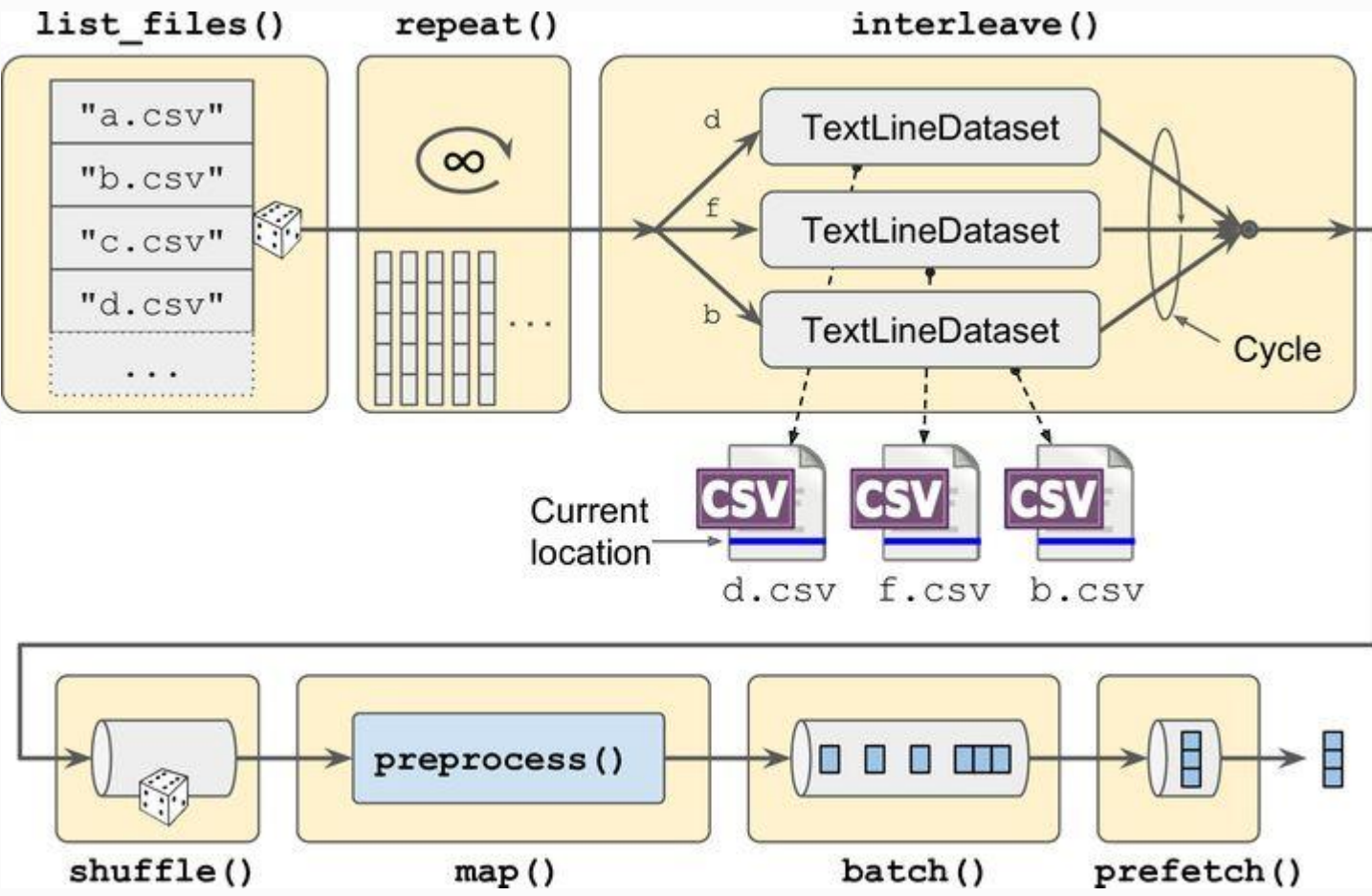
Data Preprocessing

```
[ ] def csv_reader_dataset(filepaths, repeat=1, n_readers=5,
                           n_read_threads=None, shuffle_buffer_size=10000,
                           n_parse_threads=5, batch_size=32):
    dataset = tf.data.Dataset.list_files(filepaths).repeat(repeat)
    dataset = dataset.interleave(
        lambda filepath: tf.data.TextLineDataset(filepath).skip(1),
        cycle_length=n_readers, num_parallel_calls=n_read_threads)
    dataset = dataset.shuffle(shuffle_buffer_size)
    dataset = dataset.map(preprocess, num_parallel_calls=n_parse_threads)
    dataset = dataset.batch(batch_size)
    return dataset.prefetch(1)
```

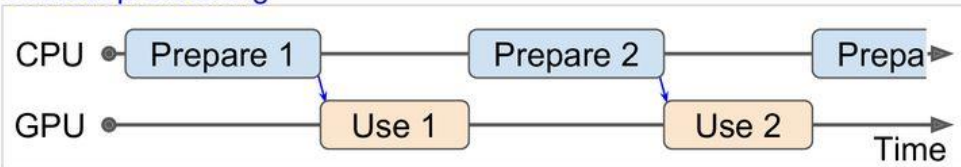


Data Preprocessing

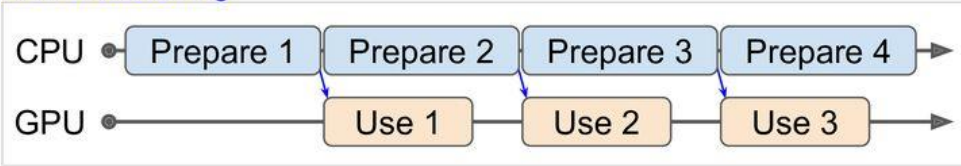
prefetch



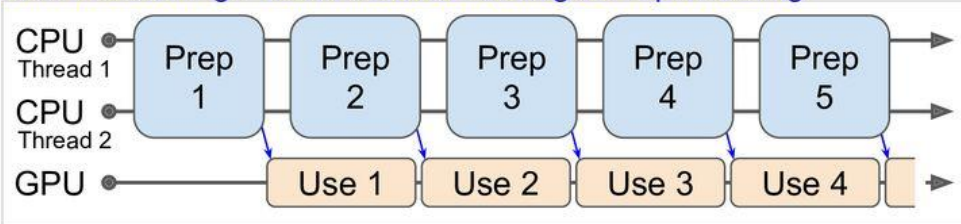
Without prefetching



With Prefetching



With Prefetching + Multithreaded Loading & Preprocessing




Standardization

데이터 적재할 때 동적으로 전처리

or

전처리 층을 모델에 직접 포함

```
 class Standardization(keras.layers.Layer):  
    def adapt(self, data_sample):  
        self.means_ = np.mean(data_sample, axis=0, keepdims=True)  
        self.stds_ = np.std(data_sample, axis=0, keepdims=True)  
    def call(self, inputs):  
        return (input - self.means_) / (self.stds_ + keras.backend.epsilon())
```

```
[ ] std_layer = Standardization()  
    std_layer.adapt(data_sample)
```

범주형 특성 인코딩

원-핫 벡터

```
[ ] vocab = ["<1H OCEAN", "INLAND", "NEAR OCEAN", "NEAR BAY", "ISLAND"]
indices = tf.range(len(vocab), dtype=tf.int64)
table_init = tf.lookup.KeyValueTensorInitializer(vocab, indices)
num_oov_buckets = 2
table = tf.lookup.StaticVocabularyTable(table_init, num_oov_buckets)
```

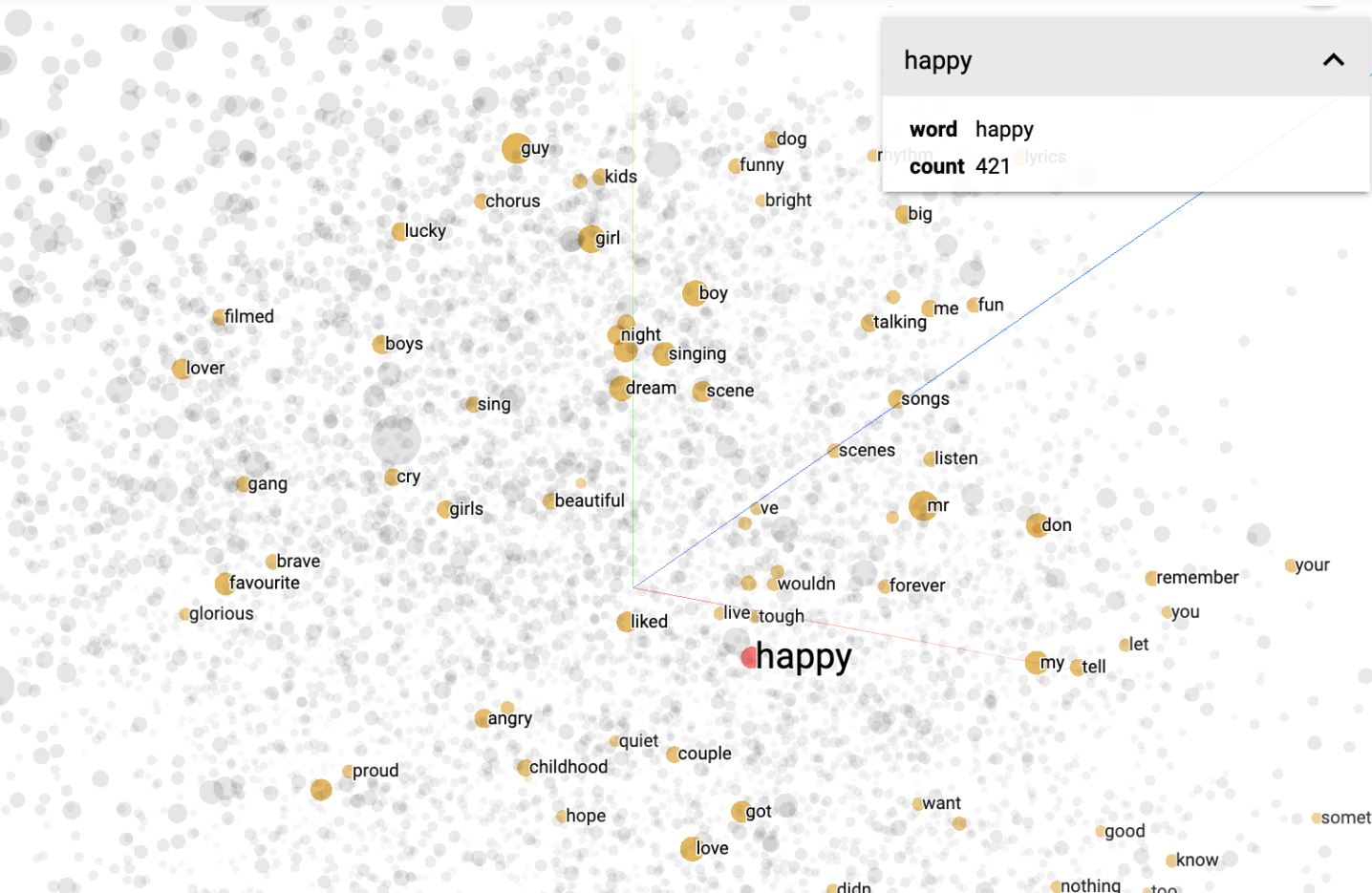
Q. Oov 버킷 사용하는 이유?

Keras.layers.TextVectorization

```
tf.keras.layers.TextVectorization(
    max_tokens=None, standardize='lower_and_strip_punctuation',
    split='whitespace', ngrams=None, output_mode='int',
    output_sequence_length=None, pad_to_max_tokens=False, vocabulary=None, **kwargs
)
```


범주형 특성 인코딩

단어 임베딩



Search by
happy .* word

neighbors 100

distance ☒ COSINE ☐ EUCLIDEAN

Nearest points in the original space:

quiet	0.590
funny	0.634
you	0.645
love	0.646
remember	0.651
sing	0.656
lucky	0.657
kids	0.663
baby	0.666
young	0.671
beautiful	0.671

Q. 훈련 비용 관점에서 원-핫 인코딩 vs 임베딩

TF 변환

```
try:
    import tensorflow_transform as tft

    def preprocess(inputs): # inputs is a batch of input features
        median_age = inputs["housing_median_age"]
        ocean_proximity = inputs["ocean_proximity"]
        standardized_age = tft.scale_to_z_score(median_age - tft.mean(median_age))
        ocean_proximity_id = tft.compute_and_apply_vocabulary(ocean_proximity)
        return {
            "standardized_median_age": standardized_age,
            "ocean_proximity_id": ocean_proximity_id
        }
except ImportError:
    print("TF Transform is not installed. Try running: pip3 install -U tensorflow-transform")
```

- 계산에 필요한 통계량 상수 존재
- 확장성, 이식성 매우 좋음

Q & A