Copyright Notice

Input to a Deep Neural Network

tf.data makes input pipelines in TensorFlow to be

- Fast
- Flexible
- Easy-to-use

Basic mechanics

Data sources

`tf.data.Dataset`

...

`tf.data.Dataset`

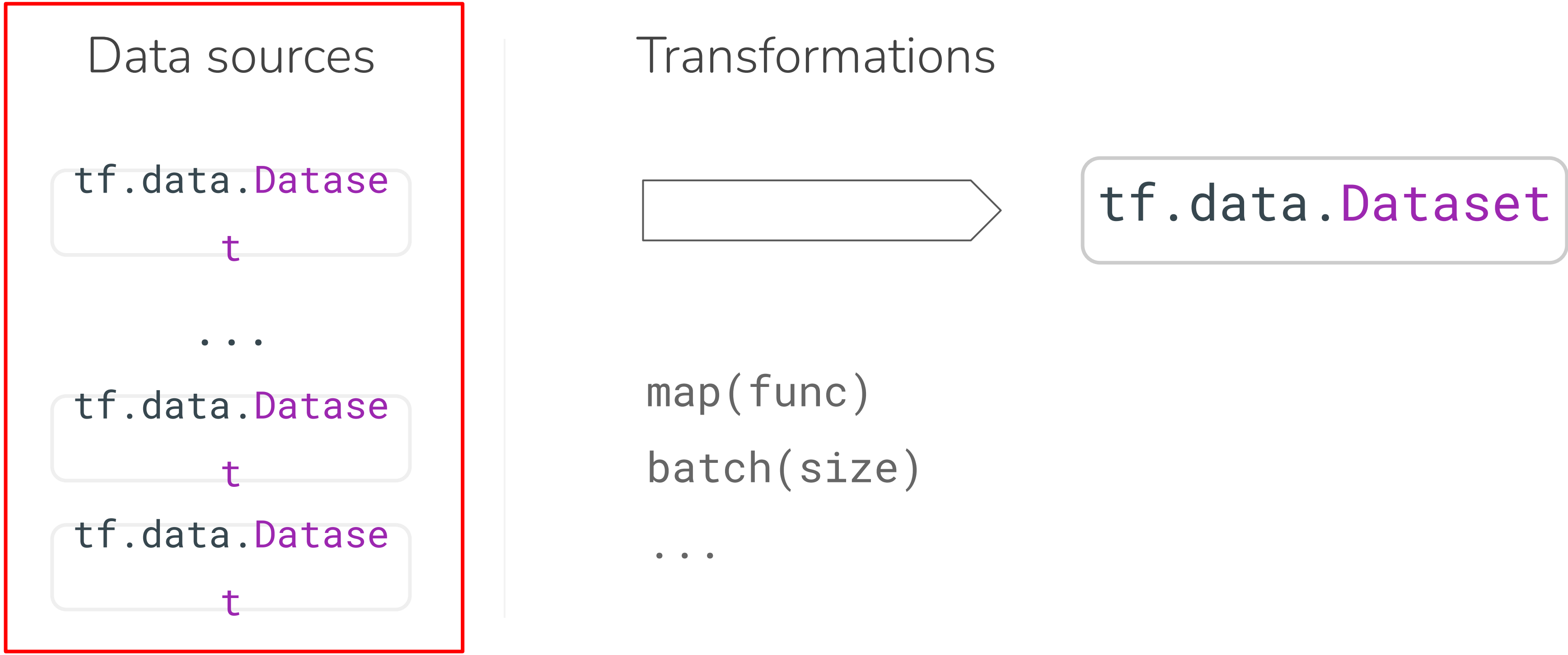
`tf.data.Dataset`

Transformations

`tf.data.Dataset`

```
map(func)
batch(size)
...
```

Basic mechanics



Data sources

`tf.data.Dataset`

...

`tf.data.Dataset`

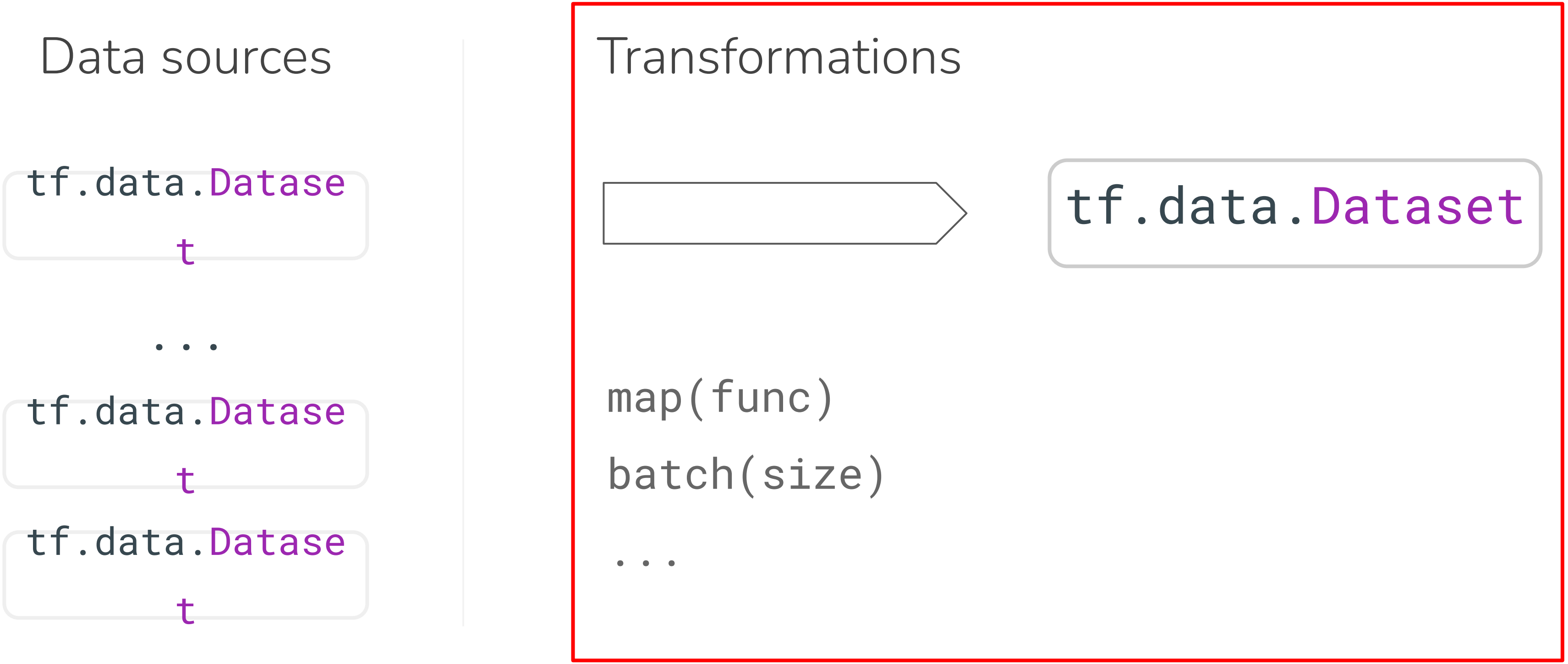
`tf.data.Dataset`

Transformations

`tf.data.Dataset`

```
map(func)
batch(size)
...
```

Basic mechanics



Data sources

tf.data.Dataset

...

tf.data.Dataset

tf.data.Dataset

Transformations

tf.data.Dataset

map(func)

batch(size)

...

Using an iterator to navigate

```
dataset = tf.data.Dataset.from_tensor_slices([1, 2, 3, 4])
it = iter(dataset)

>>> while True:
      try:
        print(next(it))
      except StopIteration as e:
        break
tf.Tensor(1, shape=(), dtype=int32)
tf.Tensor(2, shape=(), dtype=int32)
tf.Tensor(3, shape=(), dtype=int32)
tf.Tensor(4, shape=(), dtype=int32)
```

Loading numpy arrays `(from_tensor_slices)`

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))

>>> for image, label in tfds.as_numpy(dataset.take(2)):
      print(image.shape, label)
(32, 32, 3) [6]
(32, 32, 3) [9]
```

First	Last	Addr	Phone	Gender	Age	
Jane	Smith	123 Anywhere	555 555 5555	1	3	

First	Last	Addr	Phone	Gender	Age	
Jane	Smith	123 Anywhere	555 555 5555	1	3	

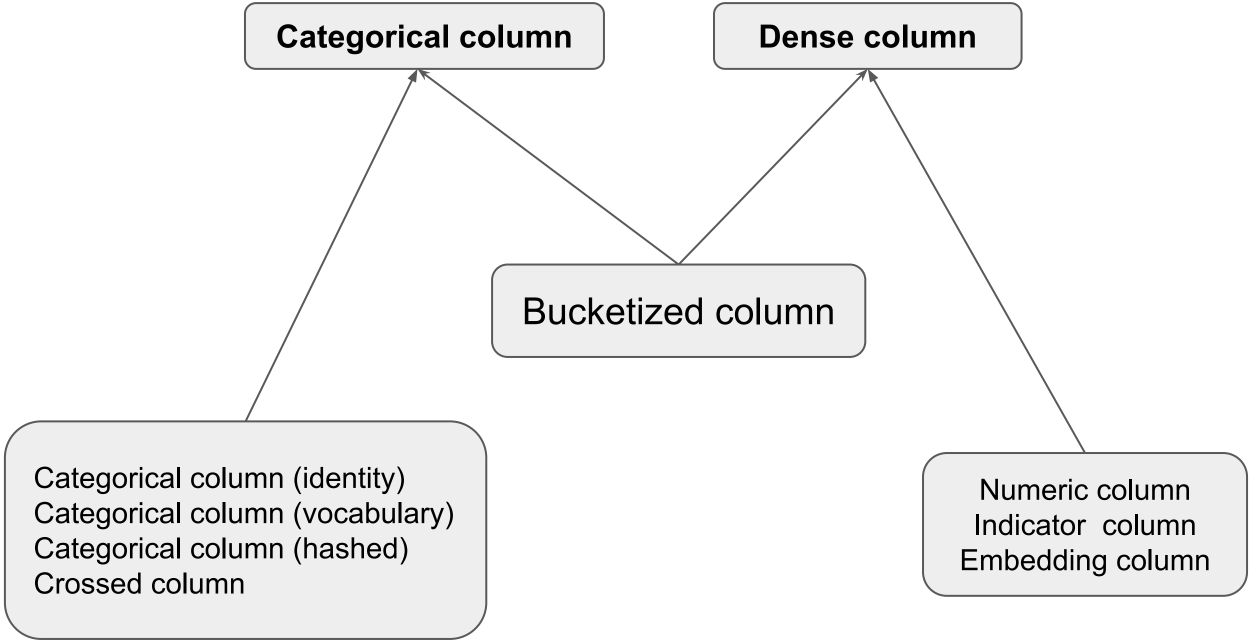
Index	Description
0	Male
1	Female
2	Nonbinary
3	Trans
4	Unassigned
...	...

First	Last	Addr	Phone	Gender	Age	
Jane	Smith	123 Anywhere	555 555 5555	1	3	

Index	Description
0	Male
1	Female
2	Nonbinary
3	Trans
4	Unassigned
...	...

Index	Description
0	Infant
1	Child
2	Teen
3	Young Adult
4	Adult
...	...

Primer on Feature Columns



Categorical column

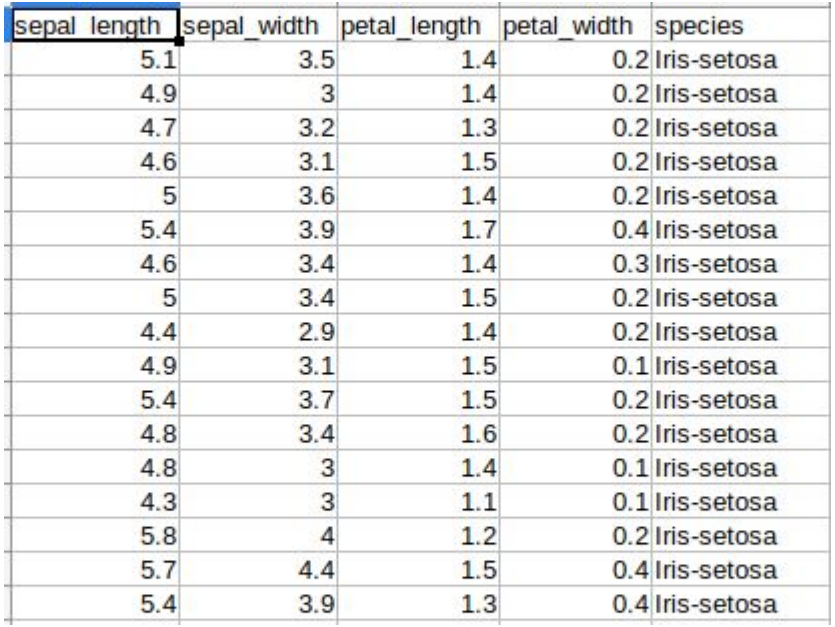
Dense column

Bucketized column

Categorical column (identity)
Categorical column (vocabulary)
Categorical column (hashed)
Crossed column

Numeric column
Indicator column
Embedding column

https://archive.ics.uci.edu/ml/datasets/iris



sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa
4.8	3.4	1.6	0.2	Iris-setosa
4.8	3	1.4	0.1	Iris-setosa
4.3	3	1.1	0.1	Iris-setosa
5.8	4	1.2	0.2	Iris-setosa
5.7	4.4	1.5	0.4	Iris-setosa
5.4	3.9	1.3	0.4	Iris-setosa

Numeric column

The Iris dataset has all numeric data as its input features:

- SepalLength
- SepalWidth
- PetalLength
- PetalWidth

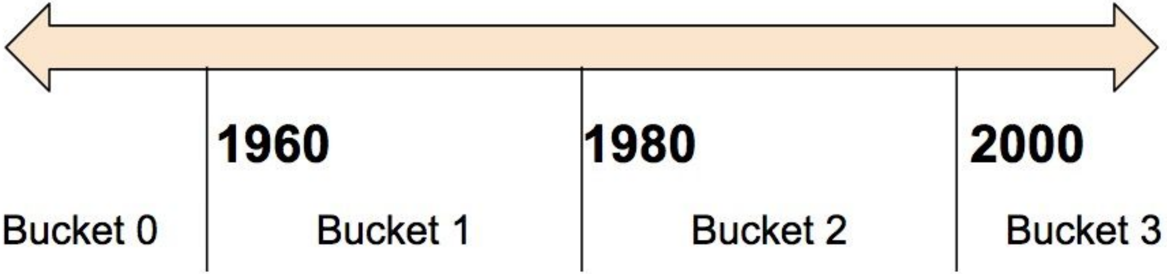
Specifying data types

```
# Defaults to a tf.float32 scalar.
numeric_feature_column = tf.feature_column.numeric_column(key="SepalLength")

# Represent a tf.float64 scalar.
numeric_feature_column = tf.feature_column.numeric_column(key="SepalLength",
                                                          dtype=tf.float64)
```

Shapes for different numeric data

```
# Represent a 10-element vector in which each cell contains a tf.float32.
vector_feature_column = tf.feature_column.numeric_column(key="Bowling",
                                                         shape=10)

# Represent a 10x5 matrix in which each cell contains a tf.float32.
matrix_feature_column = tf.feature_column.numeric_column(key="MyMatrix",
                                                         shape=[10,5])
```

Bucketized column

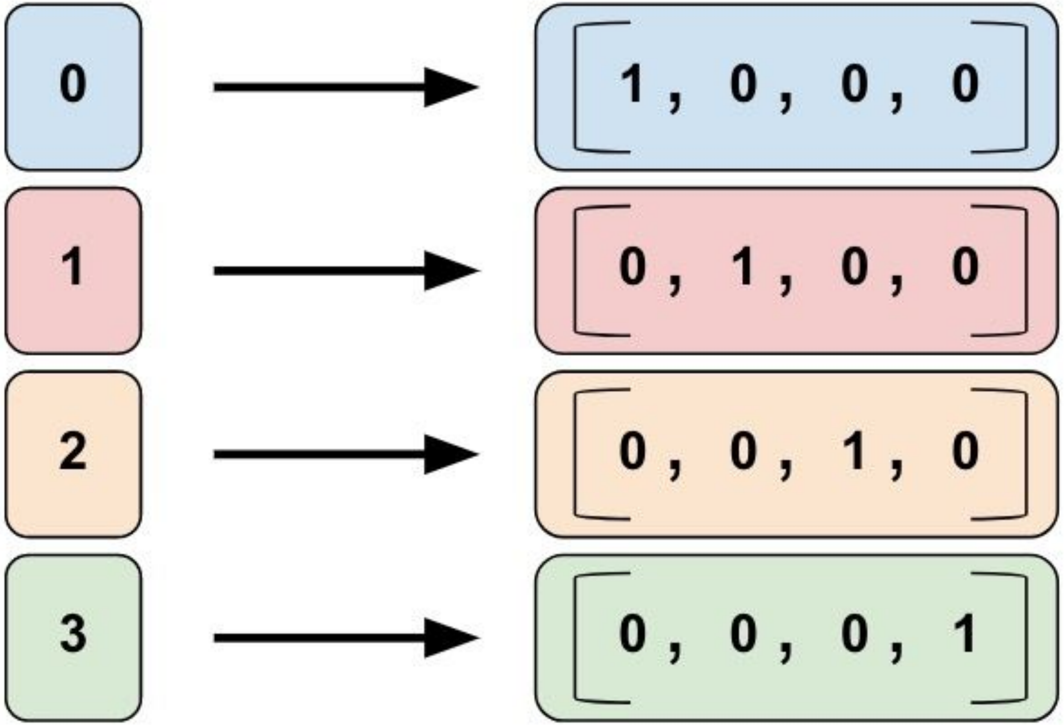


Date Range	Represented as...
< 1960	[1, 0, 0, 0]
>= 1960 but < 1980	[0, 1, 0, 0]
>= 1980 but < 2000	[0, 0, 1, 0]
>= 2000	[0, 0, 0, 1]

Bucketizing features

```
# First, convert the raw input to a numeric column.
numeric_feature_column = tf.feature_column.numeric_column("Year")

# Then, bucketize the numeric column on the years 1960, 1980, and 2000.
bucketized_feature_column = tf.feature_column.bucketized_column(
    source_column = numeric_feature_column,
    boundaries = [1960, 1980, 2000])
```

Categorical identity column



0 → [1, 0, 0, 0]

1 → [0, 1, 0, 0]

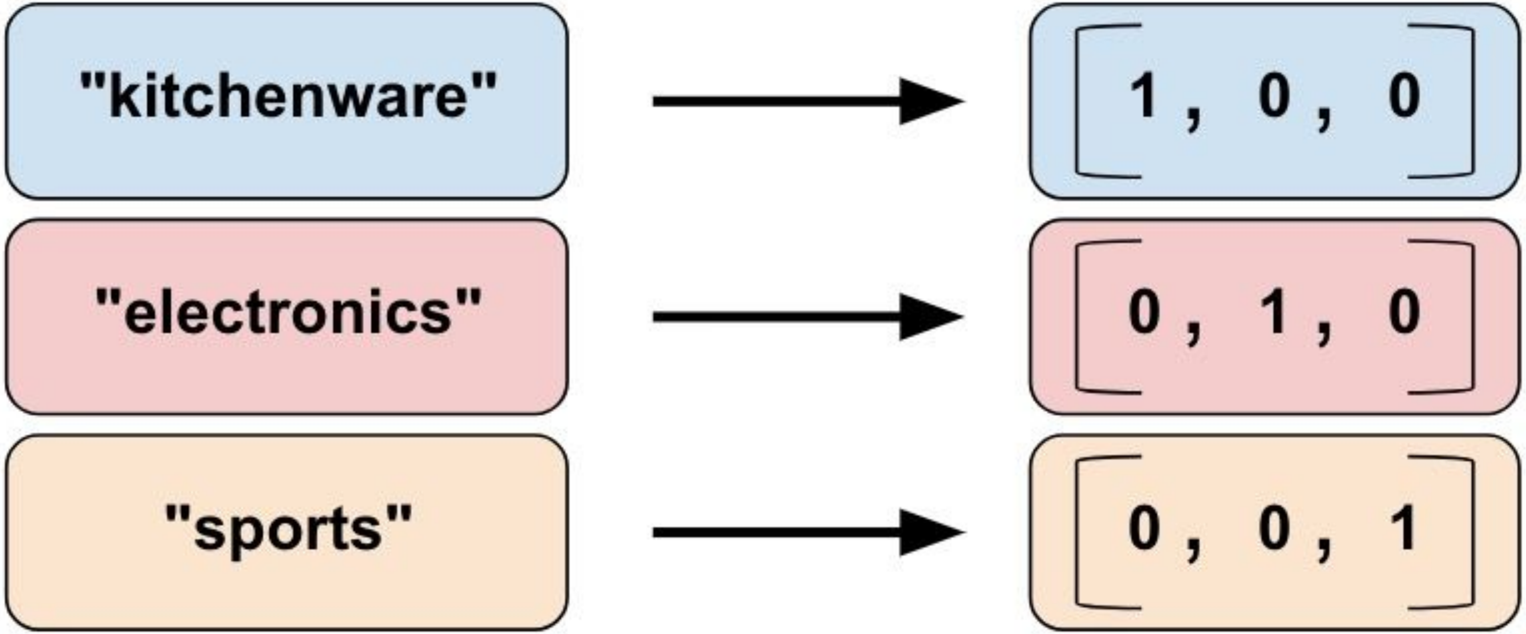
2 → [0, 0, 1, 0]

3 → [0, 0, 0, 1]

Categorizing identity features

```
identity_feature_column = tf.feature_column.categorical_column_with_identity(
    key='my_feature_b',
    num_buckets=4) # Values [0, 4]


def input_fn():
    ...
    return ({ 'my_feature_a':[7, 9, 5, 2], 'my_feature_b':[3, 1, 2, 2] },
            [Label_values])
```

Categorical vocabulary column



"kitchenware" → [1, 0, 0]

"electronics" → [0, 1, 0]

"sports" → [0, 0, 1]

Creating a categorical vocab column

From a vocabulary list

```
vocabulary_feature_column = tf.feature_column.categorical_column_with_vocabulary_list(
                            key=feature_name,
                            vocabulary_list=["kitchenware", "electronics", "sports"])
```

From a vocabulary file

```
vocabulary_feature_column = tf.feature_column.categorical_column_with_vocabulary_file(
                            key=feature_name,
                            vocabulary_file="product_class.txt",
                            vocabulary_size=3)
```

Creating a categorical vocab column

From a vocabulary list

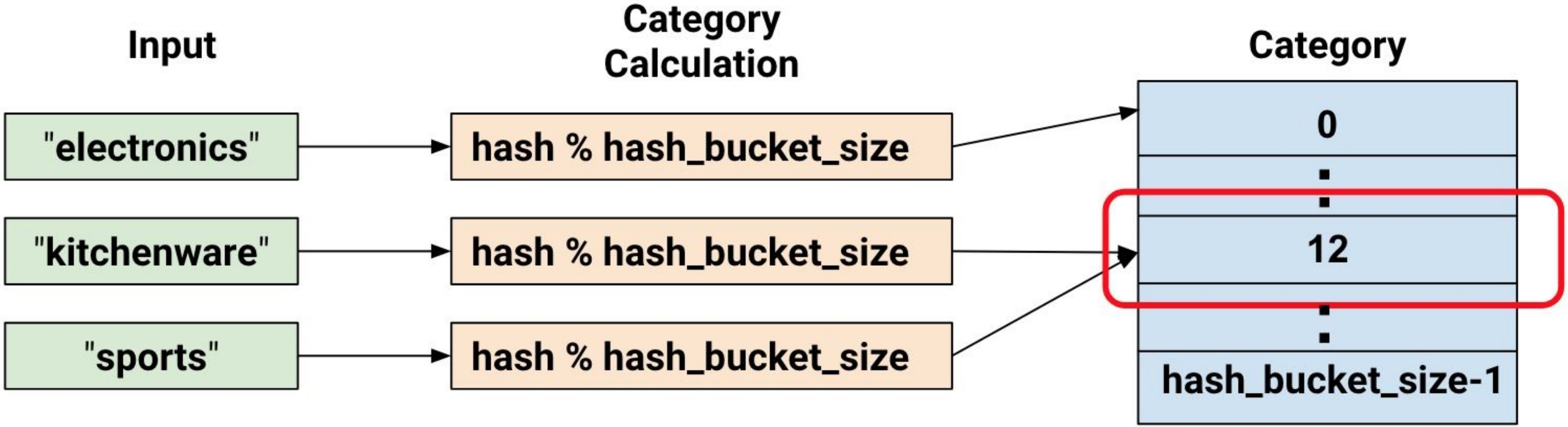
```
vocabulary_feature_column = tf.feature_column.categorical_column_with_vocabulary_list(
                            key=feature_name,
                            vocabulary_list=["kitchenware", "electronics", "sports"])
```

From a vocabulary file

```
vocabulary_feature_column = tf.feature_column.categorical_column_with_vocabulary_file(
                            key=feature_name,
                            vocabulary_file="product_class.txt",
                            vocabulary_size=3)
```

Hashed column

```
hash(raw_feature) % hash_bucket_size
```



Hashed column

```
hashed_feature_column = tf.feature_column.categorical_column_with_hash_bucket(
                        key="some_feature",
                        hash_bucket_size=100) # The number of categories
```

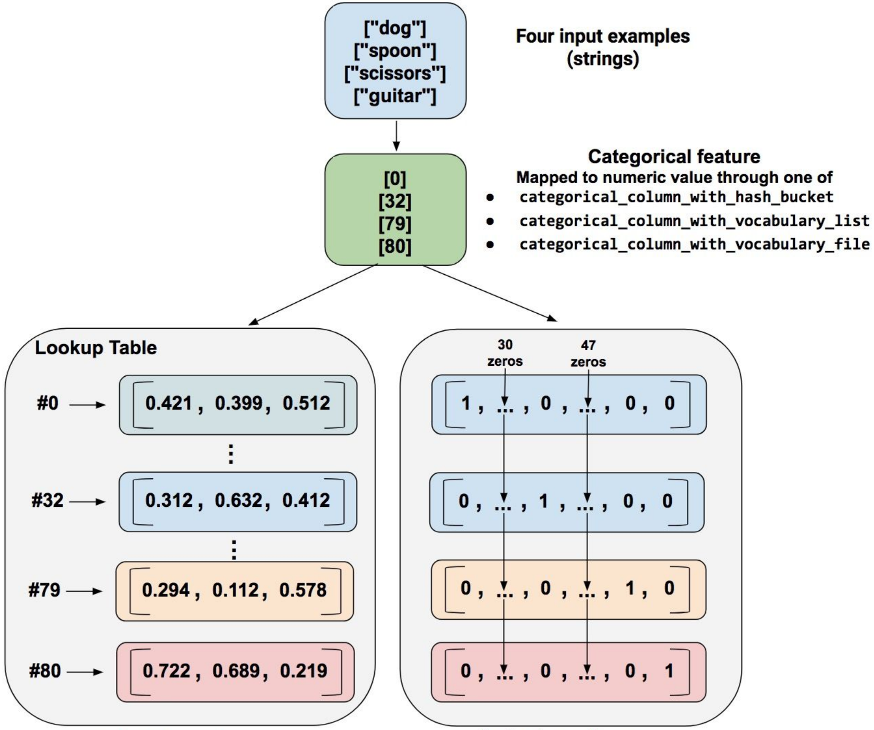
Crossed column

```
# Bucketize the latitude and longitude using the `edges`
latitude_bucket_fc = tf.feature_column.bucketized_column(
    tf.feature_column.numeric_column('latitude'),
    list(atlanta.latitude.edges))

longitude_bucket_fc = tf.feature_column.bucketized_column(
    tf.feature_column.numeric_column('longitude'),
    list(atlanta.longitude.edges))

# Cross the bucketized columns, using 5000 hash bins.
crossed_lat_lon_fc = tf.feature_column.crossed_column(
    [latitude_bucket_fc, longitude_bucket_fc], 5000)
```

Crossed column

```
# Bucketize the latitude and longitude using the `edges`
latitude_bucket_fc = tf.feature_column.bucketized_column(
    tf.feature_column.numeric_column('latitude'),
    list(atlanta.latitude.edges))

longitude_bucket_fc = tf.feature_column.bucketized_column(
    tf.feature_column.numeric_column('longitude'),
    list(atlanta.longitude.edges))

# Cross the bucketized columns, using 5000 hash bins.
crossed_lat_lon_fc = tf.feature_column.crossed_column(
    [latitude_bucket_fc, longitude_bucket_fc], 5000)
```

Embedding column



["dog"]
["spoon"]
["scissors"]
["guitar"]

Four input examples
(strings)

[0]
[32]
[79]
[80]

Categorical feature
Mapped to numeric value through one of

- `categorical_column_with_hash_bucket`
- `categorical_column_with_vocabulary_list`
- `categorical_column_with_vocabulary_file`

Lookup Table

#0 → [0.421 , 0.399 , 0.512]
⋮
#32 → [0.312 , 0.632 , 0.412]
⋮
#79 → [0.294 , 0.112 , 0.578]
#80 → [0.722 , 0.689 , 0.219]

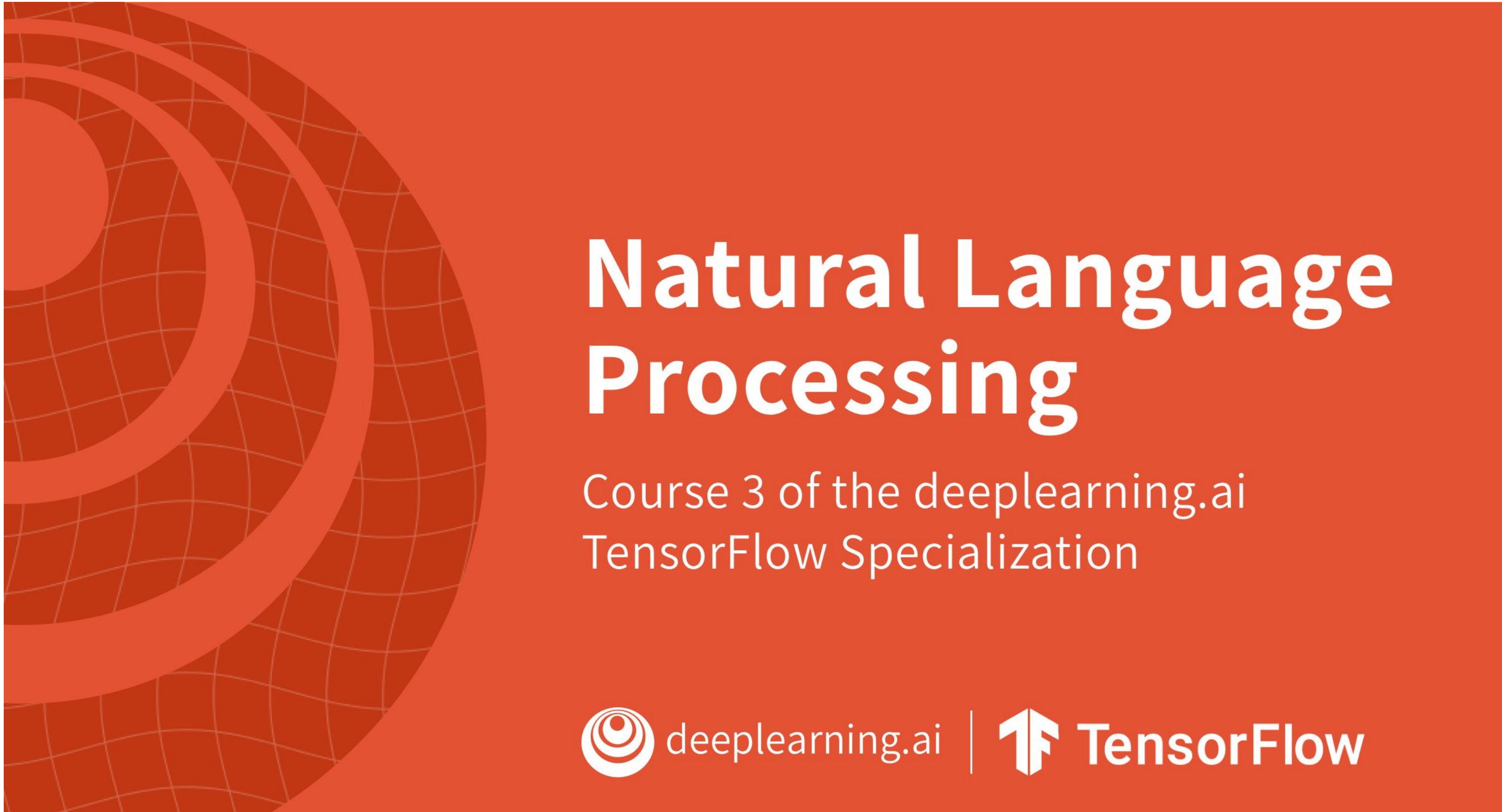
30 zeros
47 zeros

[1 , ... , 0 , ... , 0 , 0]
[0 , ... , 1 , ... , 0 , 0]
[0 , ... , 0 , ... , 1 , 0]
[0 , ... , 0 , ... , 0 , 1]

Embedding column
(vector of dimension 3)

Indicator column
(vector of dimension 81)

https://www.coursera.org/learn/natural-language-processing-tensorflow



Natural Language Processing

Course 3 of the deeplearning.ai TensorFlow Specialization

Embedding column

```
embedding_dimensions =  number_of_categories**0.25
categorical_column = ... # Create any categorical column

# Represent the categorical column as an embedding column.
# This means creating an embedding vector lookup table with one element for each
category.
embedding_column = tf.feature_column.embedding_column(
    categorical_column=categorical_column,
    dimension=embedding_dimensions)
```

Embedding column

```
embedding_dimensions =  number_of_categories**0.25

categorical_column = ... # Create any categorical column
```

```
# Represent the categorical column as an embedding column.

# This means creating an embedding vector lookup table with one element for each

category.

embedding_column = tf.feature_column.embedding_column(

    categorical_column=categorical_column,

    dimension=embedding_dimensions)
```

Embedding column

```
embedding_dimensions =  number_of_categories**0.25
categorical_column = ... # Create any categorical column

# Represent the categorical column as an embedding column.
# This means creating an embedding vector lookup table with one element for each
category.
embedding_column = tf.feature_column.embedding_column(
    categorical_column=categorical_column,
    dimension=embedding_dimensions)
```

Data sources

Numpy

DataFrames

Images

CSV and Text

TFRecords

Generators

Numpy

Loading a dataset from npz

```
# Download dataset
DATA_URL = 'https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz'
path = tf.keras.utils.get_file('mnist.npz', DATA_URL)

# Extract train and test examples
with np.load(path) as data:
  train_examples = data['x_train']
  train_labels = data['y_train']
  test_examples = data['x_test']

# Create train and test datasets out of the examples
train_dataset = tf.data.Dataset.from_tensor_slices((train_examples, train_labels))
test_dataset = tf.data.Dataset.from_tensor_slices(test_examples)

for feat, targ in train_dataset.take(2):
  print ('Features shape: {}, Target: {}'.format(feat.shape, targ))
Features shape: (28, 28), Target: 5
Features shape: (28, 28), Target: 0
```

Pandas

Create DataFrames out of CSVs

```
csv_file = tf.keras.utils.get_file('heart.csv', 'https://storage.googleapis.com/applied-dl/heart.csv')
df = pd.read_csv(csv_file)
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	1	145	233	1	2	150	0	2.3	3	0	fixed	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3	normal	1
2	67	1	4	120	229	0	2	129	1	2.6	2	2	reversible	0
3	37	1	3	130	250	0	0	187	0	3.5	3	0	normal	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	normal	0

Discretizing features

```
df['thal'] = pd.Categorical(df['thal'])
df['thal'] = df.thal.cat.codes
df.head()
```

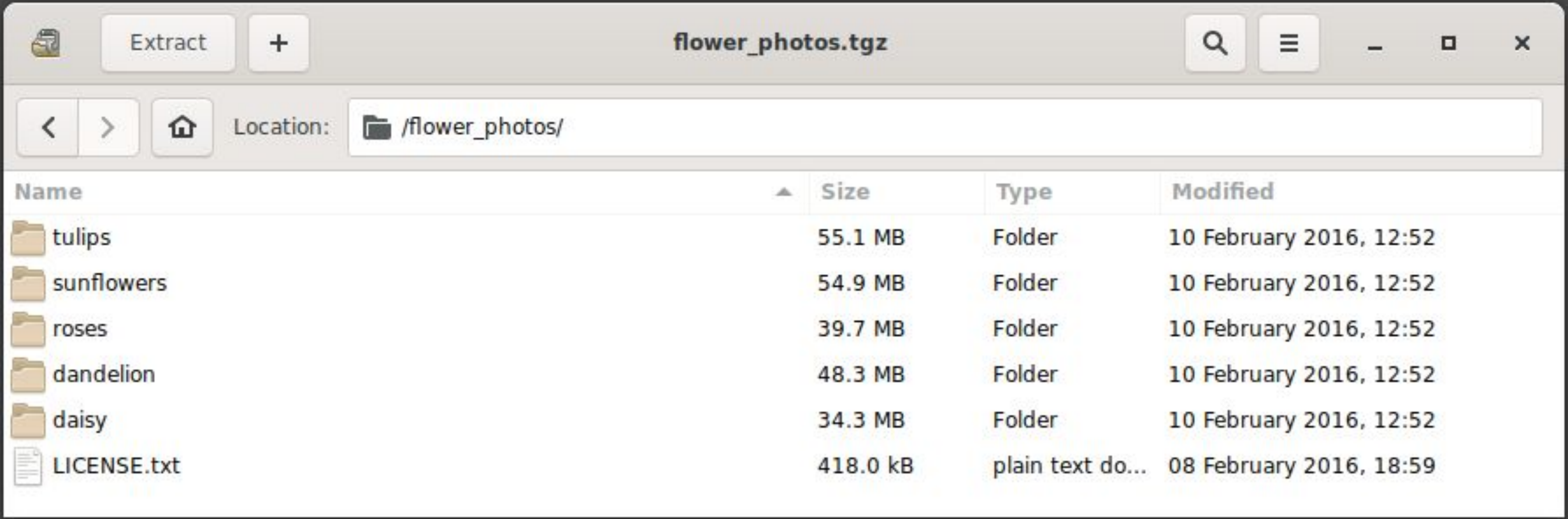
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	**thal**	target
0	63	1	1	145	233	1	2	150	0	2.3	3	0	**2**	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3	**3**	1
2	67	1	4	120	229	0	2	129	1	2.6	2	2	**4**	0
3	37	1	3	130	250	0	0	187	0	3.5	3	0	**3**	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	**3**	0

Dataset from features and targets

```
target = df.pop('target')
dataset = tf.data.Dataset.from_tensor_slices((df.values, target.values))

>>> for feat, targ in dataset.take(5):
      print ('Features: {}, Target: {}'.format(feat, targ))

Features: [ 63.    1.    1.  145.  233.    1.    2.  150.    0.    2.3   3.    0.    2. ], Target: 0
Features: [ 67.    1.    4.  160.  286.    0.    2.  108.    1.    1.5   2.    3.    3. ], Target: 1
Features: [ 67.    1.    4.  120.  229.    0.    2.  129.    1.    2.6   2.    2.    4. ], Target: 0
Features: [ 37.    1.    3.  130.  250.    0.    0.  187.    0.    3.5   3.    0.    3. ], Target: 0
Features: [ 41.    0.    2.  130.  204.    0.    2.  172.    0.    1.4   1.    0.    3. ], Target: 0
```

Download and extract images

```
import pathlib

DATA_URL =
'https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz'
data_root_orig = tf.keras.utils.get_file(origin=DATA_URL,
                                         fname='flower_photos', untar=True)
data_root = pathlib.Path(data_root_orig)

label_names = sorted(item.name for item in data_root.glob('*/') if item.is_dir())
>>> label_names
['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```



Location: /flower_photos/

Name	Size	Type	Modified
tulips	55.1 MB	Folder	10 February 2016, 12:52
sunflowers	54.9 MB	Folder	10 February 2016, 12:52
roses	39.7 MB	Folder	10 February 2016, 12:52
dandelion	48.3 MB	Folder	10 February 2016, 12:52
daisy	34.3 MB	Folder	10 February 2016, 12:52
LICENSE.txt	418.0 kB	plain text do...	08 February 2016, 18:59

Images

Download and extract images

```
import pathlib

DATA_URL =
'https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz'
data_root_orig = tf.keras.utils.get_file(origin=DATA_URL,
                                         fname='flower_photos', untar=True)
data_root = pathlib.Path(data_root_orig)

label_names = sorted(item.name for item in data_root.glob('*/') if item.is_dir())
>>> label_names
['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

Display a random sample from the loaded dataset

```
import random
import IPython.display as display

all_image_paths = list(data_root.glob('*/*'))
all_image_paths = [str(path) for path in all_image_paths]
random.shuffle(all_image_paths)

image_count = len(all_image_paths)
image_count

image_path = random.choice(all_image_paths)
display.display(display.Image(image_path))
```

CSV

Loading the structured dataset

```
TRAIN_DATA_URL = "https://storage.googleapis.com/tf-datasets/titanic/train.csv"
train_file_path = tf.keras.utils.get_file("train.csv", TRAIN_DATA_URL)

df = pd.read_csv(train_file_path, sep=',')
df.head()
```

	survived	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone
0	0	male	22.0	1	0	7.2500	Third	unknown	Southampton	n
1	1	female	38.0	1	0	71.2833	First	C	Cherbourg	n
2	1	female	26.0	0	0	7.9250	Third	unknown	Southampton	y
3	1	female	35.0	1	0	53.1000	First	C	Southampton	n
4	0	male	28.0	0	0	8.4583	Third	unknown	Queenstown	y

Loading the structured dataset

```
TRAIN_DATA_URL = "https://storage.googleapis.com/tf-datasets/titanic/train.csv"
train_file_path = tf.keras.utils.get_file("train.csv", TRAIN_DATA_URL)

df = pd.read_csv(train_file_path, sep=',')
df.head()
```

	survived	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone
0	0	male	22.0	1	0	7.2500	Third	unknown	Southampton	n
1	1	female	38.0	1	0	71.2833	First	C	Cherbourg	n
2	1	female	26.0	0	0	7.9250	Third	unknown	Southampton	y
3	1	female	35.0	1	0	53.1000	First	C	Southampton	n
4	0	male	28.0	0	0	8.4583	Third	unknown	Queenstown	y

Numeric data

```
NUMERIC_FEATURES = ['age','n_siblings_spouses','parch', 'fare']
dense_df = df[NUMERIC_FEATURES]
dense_df.head()
```

	age	n_siblings_spouses	parch	fare
0	22.0	1	0	7.2500
1	38.0	1	0	71.2833
2	26.0	0	0	7.9250
3	35.0	1	0	53.1000

Leveraging features columns

```
numeric_columns = []
for feature in NUMERIC_FEATURES:
  num_col = tf.feature_column.numeric_column(feature)
  numeric_columns.append(tf.feature_column.indicator_column(num_col))
```

```
>>> numeric_columns
[IndicatorColumn(categorical_column=NumericColumn(key='age', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None)),
 IndicatorColumn(categorical_column=NumericColumn(key='n_siblings_spouses', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None)),
 IndicatorColumn(categorical_column=NumericColumn(key='parch', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None)),
 IndicatorColumn(categorical_column=NumericColumn(key='fare', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None))]
```

CSV

Leveraging features columns

```
numeric_columns = []
for feature in NUMERIC_FEATURES:
  num_col = tf.feature_column.numeric_column(feature)
  numeric_columns.append(tf.feature_column.indicator_column(num_col))
```

```
>>> numeric_columns
[IndicatorColumn(categorical_column=NumericColumn(key='age', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None)),
 IndicatorColumn(categorical_column=NumericColumn(key='n_siblings_spouses', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None)),
 IndicatorColumn(categorical_column=NumericColumn(key='parch', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None)),
 IndicatorColumn(categorical_column=NumericColumn(key='fare', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None))]
```

CSV

Leveraging features columns

```
numeric_columns = []
for feature in NUMERIC_FEATURES:
  num_col = tf.feature_column.numeric_column(feature)
  numeric_columns.append(tf.feature_column.indicator_column(num_col))
```

```
>>> numeric_columns
[IndicatorColumn(categorical_column=NumericColumn(key='age', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None)),
 IndicatorColumn(categorical_column=NumericColumn(key='n_siblings_spouses', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None)),
 IndicatorColumn(categorical_column=NumericColumn(key='parch', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None)),
 IndicatorColumn(categorical_column=NumericColumn(key='fare', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None))]
```

Leveraging features columns

```
numeric_columns = []
for feature in NUMERIC_FEATURES:
  num_col = tf.feature_column.numeric_column(feature)
  numeric_columns.append(tf.feature_column.indicator_column(num_col))

>>> numeric_columns
[IndicatorColumn(categorical_column=NumericColumn(key='age', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None)),
 IndicatorColumn(categorical_column=NumericColumn(key='n_siblings_spouses', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None)),
 IndicatorColumn(categorical_column=NumericColumn(key='parch', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None)),
 IndicatorColumn(categorical_column=NumericColumn(key='fare', shape=(1,),
default_value=None, dtype=tf.float32, normalizer_fn=None))]
```

Leveraging features columns

```
>>> numeric_columns
[IndicatorColumn(categorical_column=NumericColumn(key='age',
shape=(1,), default_value=None, dtype=tf.float32,
normalizer_fn=None)),
...
```

Categorical data

```
CATEGORIES = {
    'sex': ['male', 'female'],
    'class' : ['First', 'Second', 'Third'],
    'deck' : ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'],
    'embark_town' : ['Cherbourg', 'Southampton', 'Queenstown'],
    'alone' : ['y', 'n']
}

cat_df = df[list(CATEGORIES.keys())]
cat_df.head()
```

Categorical data

```
cat_df = df[list(CATEGORIES.keys())]
cat_df.head()
```

```
CATEGORIES = {
    'sex': ['male', 'female'],
    'class' : ['First', 'Second', 'Third'],
    'deck' : ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'],
    'embark_town' : ['Cherbourg', 'Southhampton', 'Queenstown'],
    'alone' : ['y', 'n']
}
```

	sex	class	deck	embark_town	alone
0	male	Third	unknown	Southampton	n
1	female	First	C	Cherbourg	n
2	female	Third	unknown	Southampton	y
3	female	First	C	Southampton	n
4	male	Third	unknown	Queenstown	y

CSV

Categorical columns from raw data

```
categorical_columns = []
for feature, vocab in CATEGORIES.items():
  cat_col = tf.feature_column.categorical_column_with_vocabulary_list(
        key=feature, vocabulary_list=vocab)
  categorical_columns.append(tf.feature_column.indicator_column(cat_col))
```

CSV

Categorical columns from raw data

```
categorical_columns = []
for feature, vocab in CATEGORIES.items():
  cat_col = tf.feature_column.categorical_column_with_vocabulary_list(
        key=feature, vocabulary_list=vocab)
  categorical_columns.append(tf.feature_column.indicator_column(cat_col))
```

Categorical columns from raw data

```
categorical_columns = []
for feature, vocab in CATEGORIES.items():
  cat_col = tf.feature_column.categorical_column_with_vocabulary_list(
        key=feature, vocabulary_list=vocab)
  categorical_columns.append(tf.feature_column.indicator_column(cat_col))
```

CSV

Categorical columns from raw data

```
>>> categorical_columns
[IndicatorColumn(categorical_column=VocabularyListCategorica
lColumn(key='sex', vocabulary_list=('male', 'female'),
dtype=tf.string, default_value=-1, num_oov_buckets=0)),

IndicatorColumn(categorical_column=VocabularyListCategorical
Column(key='class', vocabulary_list=('First', 'Second',
'Third'), dtype=tf.string, default_value=-1,
num_oov_buckets=0)),
...
```

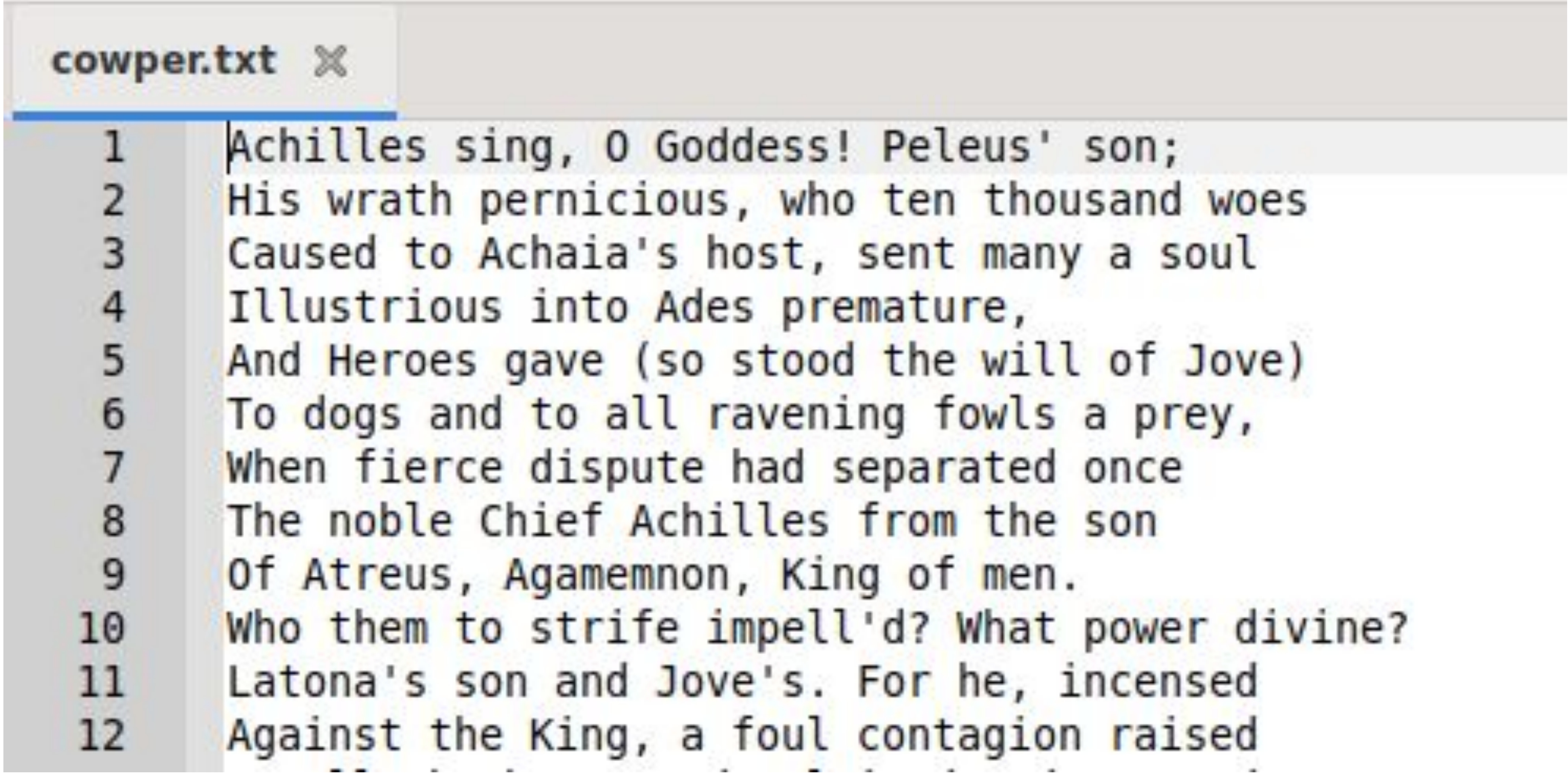
CSV

Categorical columns from raw data

```
>>> categorical_columns
[IndicatorColumn(categorical_column=VocabularyListCategorica
lColumn(key='sex', vocabulary_list=('male', 'female'),
dtype=tf.string, default_value=-1, num_oov_buckets=0)),

IndicatorColumn(categorical_column=VocabularyListCategorical
Column(key='class', vocabulary_list=('First', 'Second',
'Third'), dtype=tf.string, default_value=-1,
num_oov_buckets=0)),
...
```

Loading texts with TextLineDataset

```
DIRECTORY_URL = 'https://storage.googleapis.com/download.tensorflow.org/data/illiad/'
FILE_NAME = 'cowper.txt'
```



```
Achilles sing, O Goddess! Peleus' son;
His wrath pernicious, who ten thousand woes
Caused to Achaia's host, sent many a soul
Illustrious into Ades premature,
And Heroes gave (so stood the will of Jove)
To dogs and to all ravening fowls a prey,
When fierce dispute had separated once
The noble Chief Achilles from the son
Of Atreus, Agamemnon, King of men.
Who them to strife impell'd? What power divine?
Latona's son and Jove's. For he, incensed
Against the King, a foul contagion raised
```

Loading texts with TextLineDataset

```
file_path = tf.keras.utils.get_file(name,
               origin=DIRECTORY_URL + FILE_NAME)

lines_dataset = tf.data.TextLineDataset(file_path)
```

Inspecting texts

```
>>> for text_data in tfds.as_numpy(lines_dataset.take(3)):
      print(text_data.decode('utf-8'))
```

```
Achilles sing, O Goddess! Peleus' son;
His wrath pernicious, who ten thousand woes
Caused to Achaia's host, sent many a soul
```

Reading TFRecord files

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

feature_description = {
        'feature1': tf.io.FixedLenFeature((), tf.string),
        'feature2': tf.io.FixedLenFeature((), tf.int64)
}

for raw_record in raw_dataset.take(1):
  example = tf.io.parse_single_example(raw_record, feature_description)
  print(example)
```

Reading TFRecord files

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

feature_description = {
        'feature1': tf.io.FixedLenFeature((), tf.string),
        'feature2': tf.io.FixedLenFeature((), tf.int64)
}

for raw_record in raw_dataset.take(1):
  example = tf.io.parse_single_example(raw_record, feature_description)
  print(example)
```

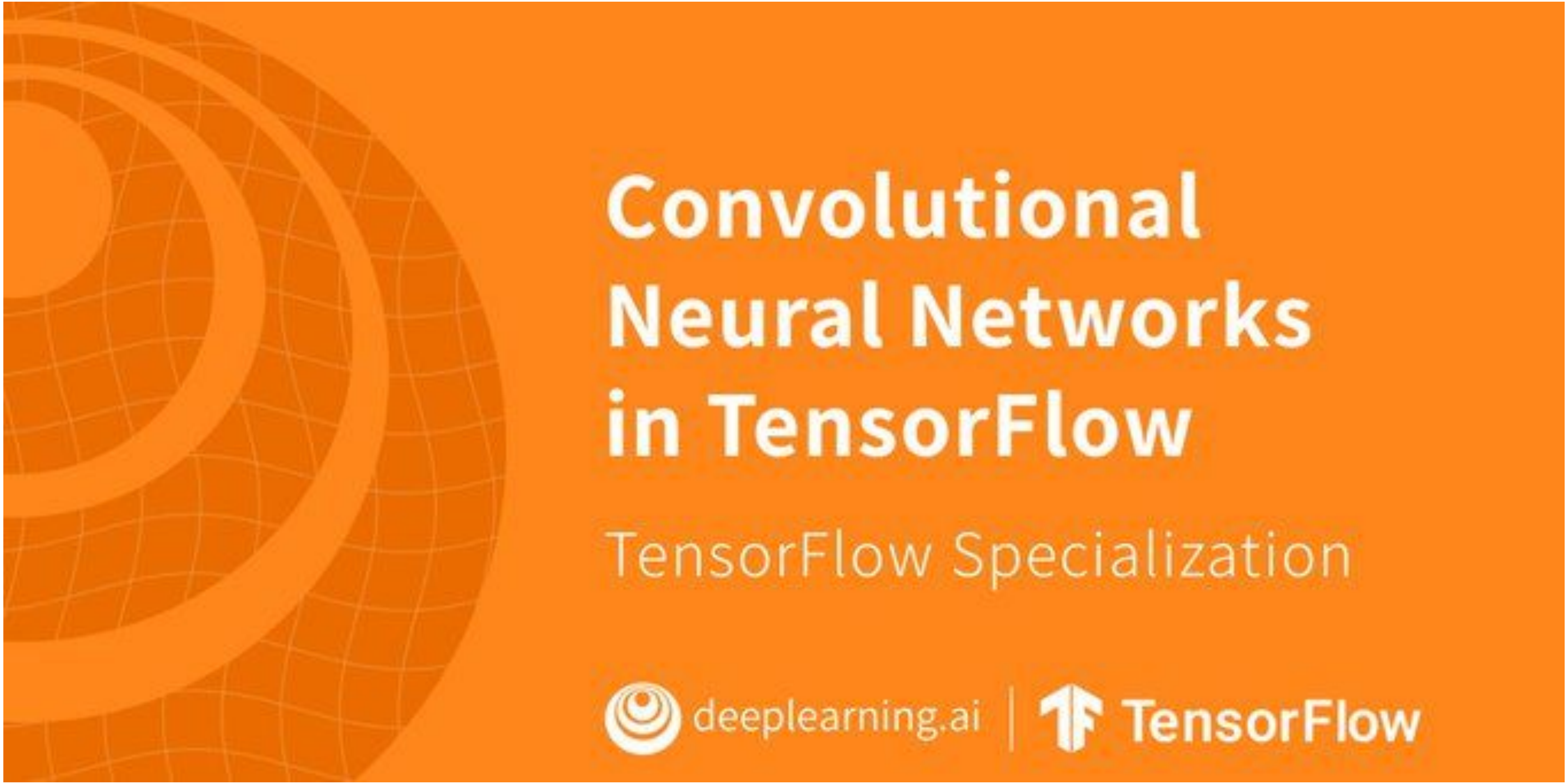
Reading TFRecord files

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

feature_description = {
        'feature1': tf.io.FixedLenFeature((), tf.string),
        'feature2': tf.io.FixedLenFeature((), tf.int64)
}

for raw_record in raw_dataset.take(1):
  example = tf.io.parse_single_example(raw_record, feature_description)
  print(example)
```

Reading TFRecord files

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

feature_description = {
        'feature1': tf.io.FixedLenFeature((), tf.string),
        'feature2': tf.io.FixedLenFeature((), tf.int64)
}

for raw_record in raw_dataset.take(1):
  example = tf.io.parse_single_example(raw_record, feature_description)
  print(example)
```

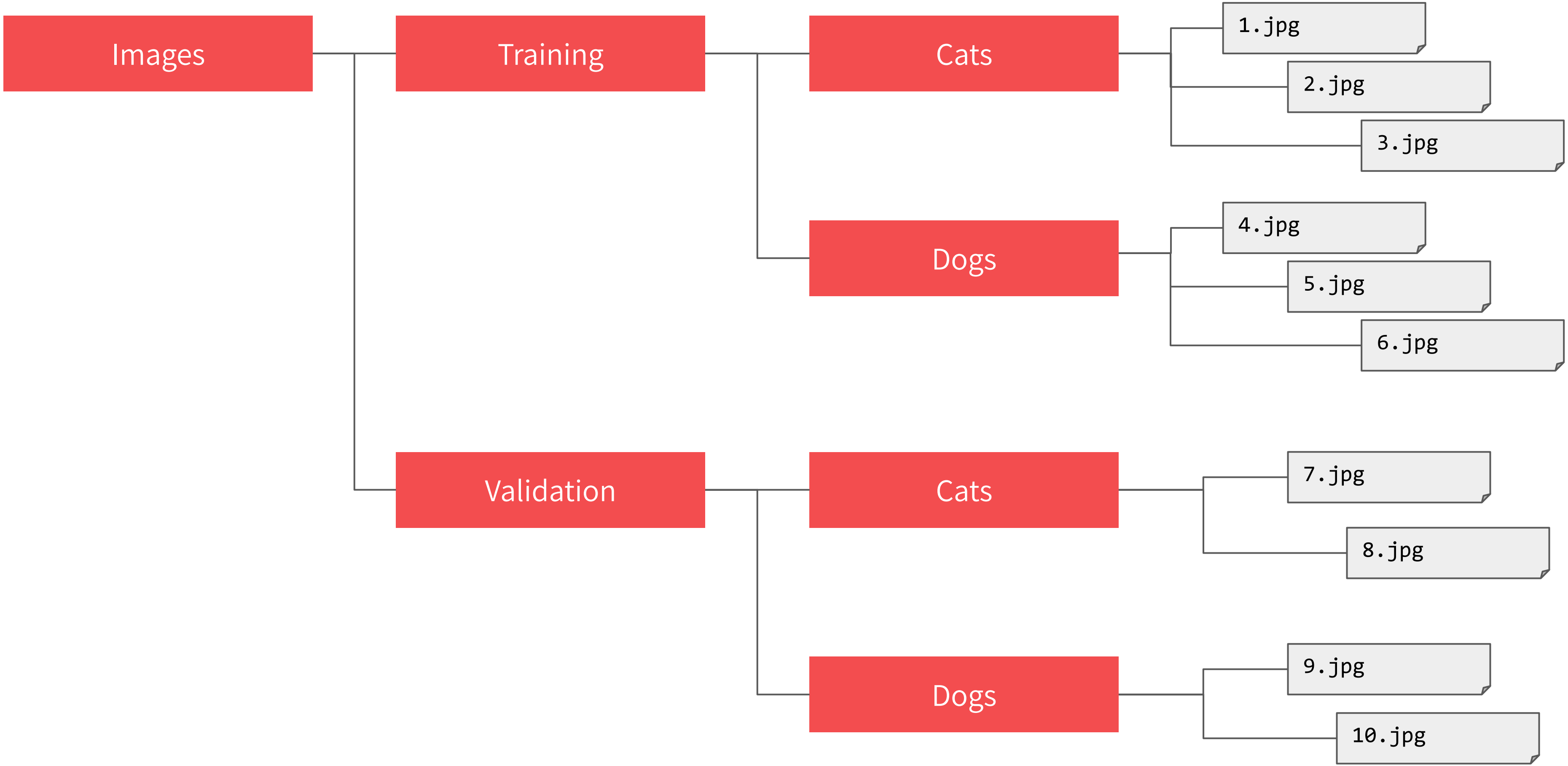
https://www.coursera.org/learn/convolutional-neural-networks-tensorflow



Convolutional Neural Networks in TensorFlow

TensorFlow Specialization

deeplearning.ai | TensorFlow


Images
Training
Cats
1.jpg
2.jpg
3.jpg
Dogs
4.jpg
5.jpg
6.jpg
Validation
Cats
7.jpg
8.jpg
Dogs
9.jpg
10.jpg

Keras ImageDataGenerator

```
def make_generator():
  train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1. / 255,
                     rotation_range=20, zoom_range=[0.8, 1.2])

  train_generator = train_datagen.flow_from_directory(catsdogs,
                     target_size=(224, 224), class_mode='categorical',batch_size=32)

  return train_generator


train_generator = tf.data.Dataset.from_generator(
                           make_generator,(tf.float32, tf.uint8))
```

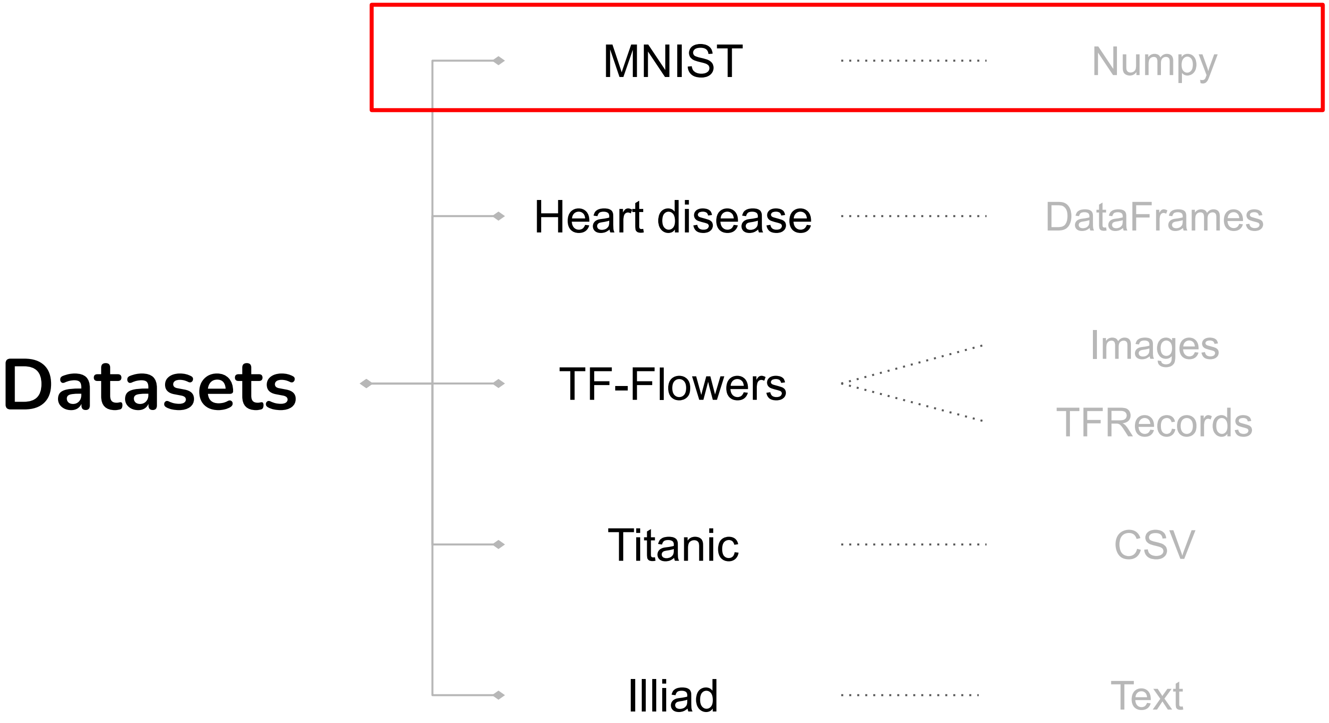
Keras ImageDataGenerator

```
def make_generator():
  train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1. / 255,
                     rotation_range=20, zoom_range=[0.8, 1.2])

  train_generator = train_datagen.flow_from_directory(catsdogs,
                     target_size=(224, 224), class_mode='categorical',batch_size=32)

  return train_generator


train_generator = tf.data.Dataset.from_generator(
                           make_generator,(tf.float32, tf.uint8))
```

Keras ImageDataGenerator

```
def make_generator():
  train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1. / 255,
                                   rotation_range=20, zoom_range=[0.8, 1.2])

  train_generator = train_datagen.flow_from_directory(catsdogs,
                                   target_size=(224, 224), class_mode='categorical',batch_size=32)

  return train_generator
```

```
train_generator = tf.data.Dataset.from_generator(
                                   make_generator,(tf.float32, tf.uint8))
```



Datasets

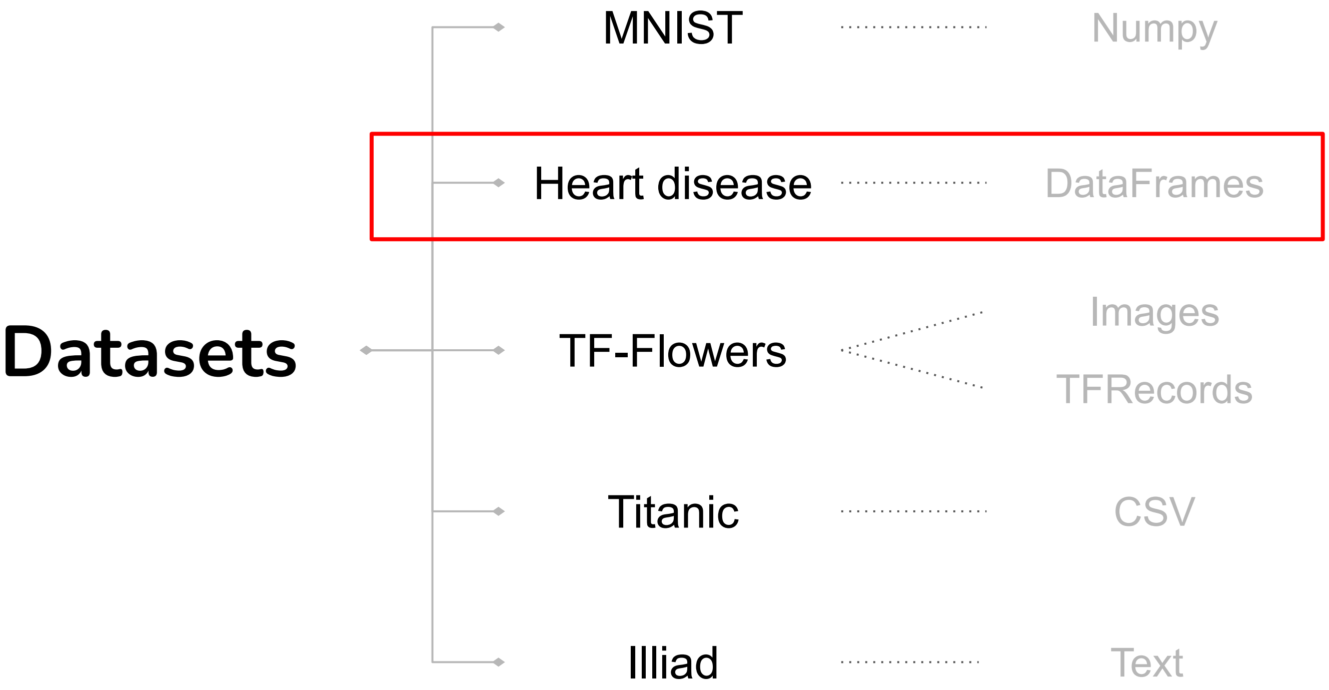
MNIST — Numpy

Heart disease — DataFrames

TF-Flowers — Images, TFRecords

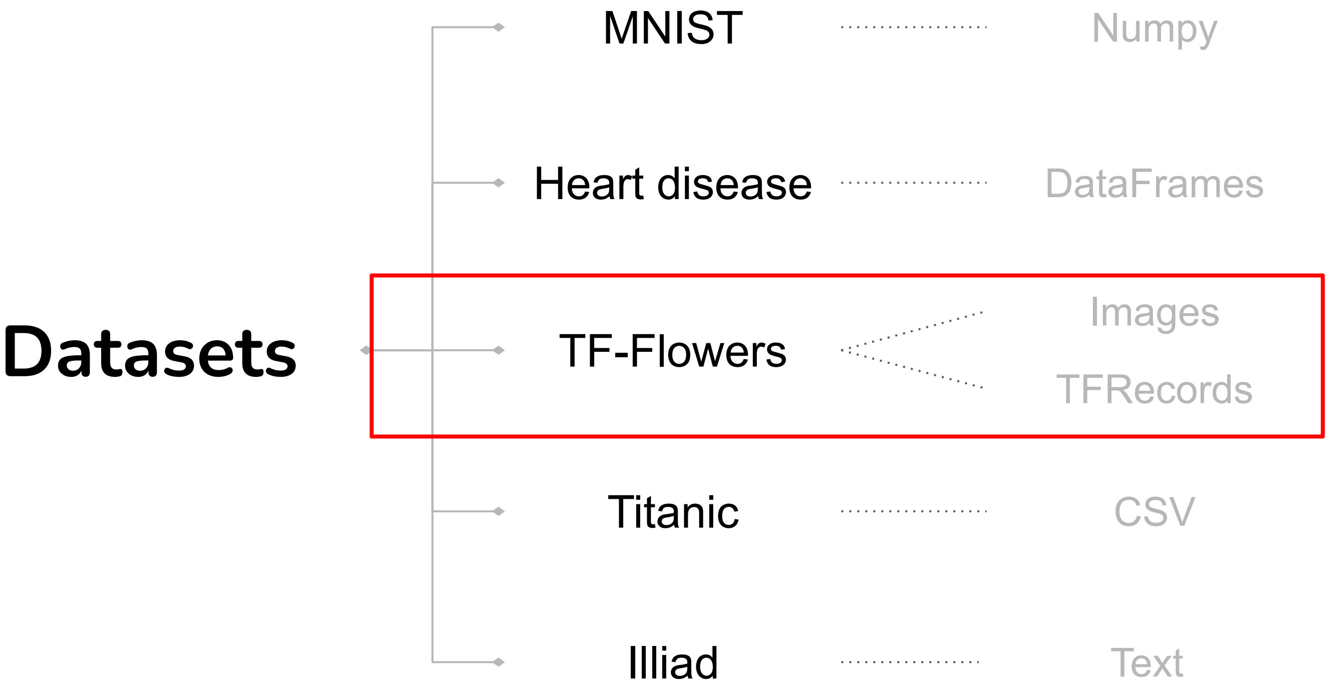
Titanic — CSV

Illiad — Text



Datasets

MNIST	Numpy
Heart disease	DataFrames
TF-Flowers	Images TFRecords
Titanic	CSV
Illiad	Text



Datasets

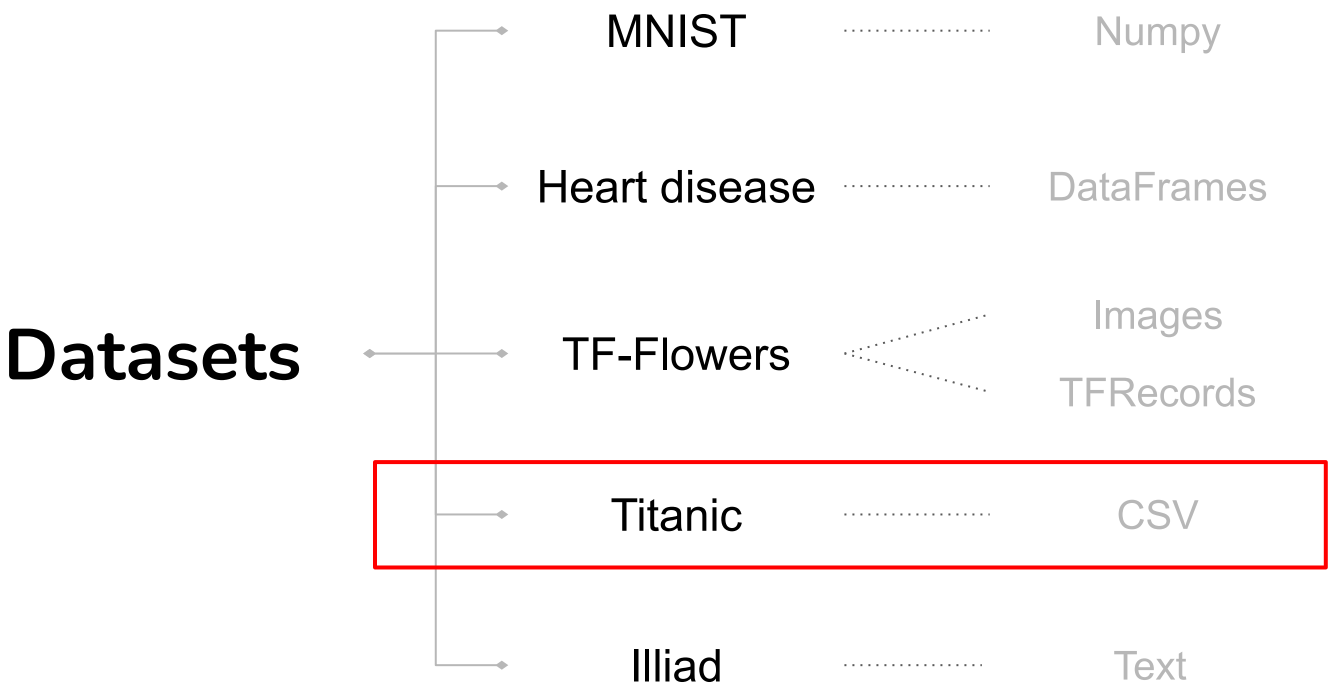
MNIST Numpy

Heart disease DataFrames

TF-Flowers Images TFRecords

Titanic CSV

Illiad Text



Datasets

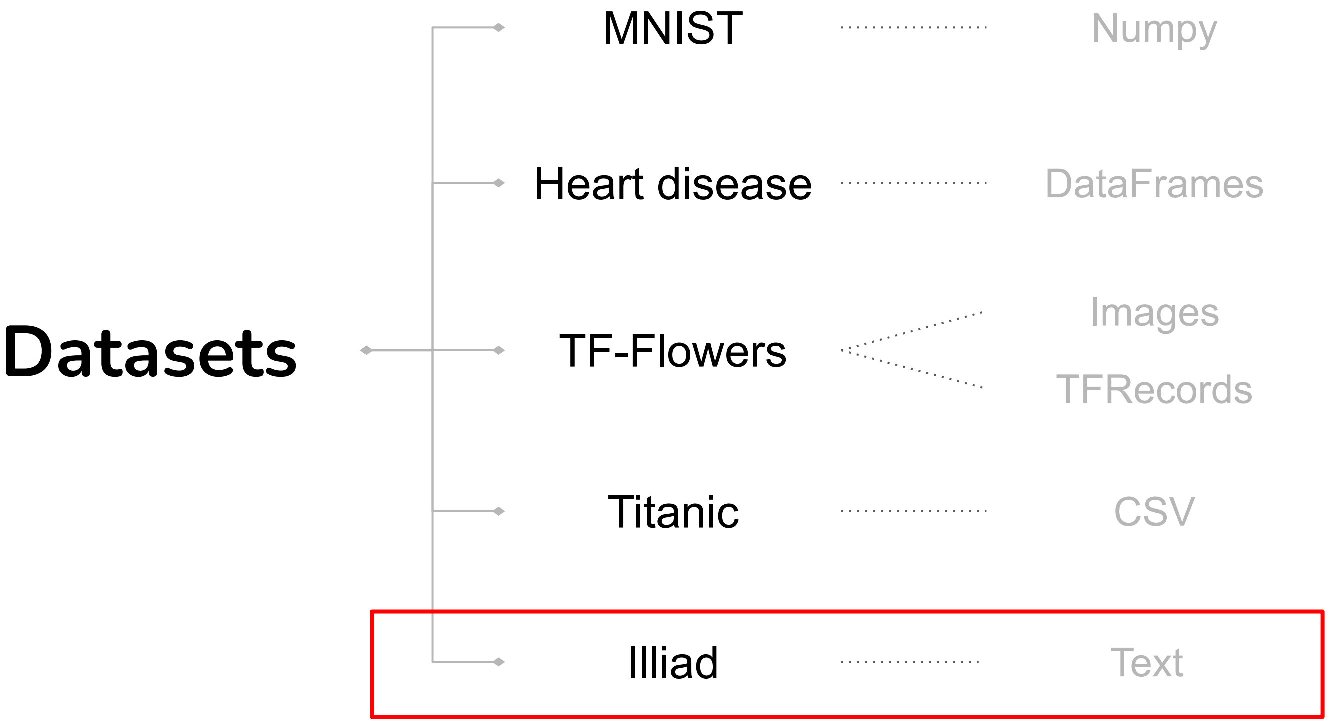
MNIST — Numpy

Heart disease — DataFrames

TF-Flowers — Images, TFRecords

Titanic — CSV

Illiad — Text



Datasets

MNIST Numpy

Heart disease DataFrames

TF-Flowers Images TFRecords

Titanic CSV

Illiad Text

MNIST

```
# Fetch the numpy dataset
DATA_URL =
'https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz'
path = tf.keras.utils.get_file('mnist.npz', DATA_URL)

# Extract train, test sets
with np.load(path) as data:
  train_examples = data['x_train']
  train_labels = data['y_train']
  test_examples = data['x_test']
  test_labels = data['y_test']
```

Numpy

MNIST

```
# Load them with tf.data
train_dataset = tf.data.Dataset.from_tensor_slices((train_examples, train_labels))
test_dataset = tf.data.Dataset.from_tensor_slices((test_examples, test_labels))

# Apply transformations like batch, shuffle to the dataset
train_dataset = train_dataset.shuffle(100).batch(64)
test_dataset = test_dataset.batch(64)
```

Training on MNIST

```
X, y = next(iter(train_dataset))
input_shape = X.numpy().shape[1:]


# Create a simple sequential model comprising of a Dense layer
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=input_shape),
    ...
    tf.keras.layers.Dense(10, activation='softmax')])

model.compile(optimizer=tf.keras.optimizers.RMSprop(), loss=...,
metrics=...])

# Train the model
model.fit(train_dataset, epochs=10)
```

Training on MNIST

```
X, y = next(iter(train_dataset))
input_shape = X.numpy().shape[1:]


# Create a simple sequential model comprising of a Dense layer
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=input_shape),
    ...
    tf.keras.layers.Dense(10, activation='softmax')])

model.compile(optimizer=tf.keras.optimizers.RMSprop(), loss=...,
metrics=...])

# Train the model
model.fit(train_dataset, epochs=10)
```

Identifying Heart Disease

```
csv_file = tf.keras.utils.get_file('heart.csv',
'https://storage.googleapis.com/applied-dl/heart.csv')

df = pd.read_csv(csv_file)
df['thal'] = pd.Categorical(df['thal'])
df['thal'] = df.thal.cat.codes

target = df.pop('target')
```

Pandas

Identifying Heart Disease

```
csv_file = tf.keras.utils.get_file('heart.csv', 'https://storage.googleapis.com/applied-dl/heart.csv')

df = pd.read_csv(csv_file)
df['thal'] = pd.Categorical(df['thal'])
df['thal'] = df.thal.cat.codes

target = df.pop('target')
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	1	145	233	1	2	150	0	2.3	3	0	fixed
1	67	1	4	160	286	0	2	108	1	1.5	2	3	normal
2	67	1	4	120	229	0	2	129	1	2.6	2	2	reversible

Training the model (Sequential)

```
dataset = tf.data.Dataset.from_tensor_slices((df.values, target.values))
train_dataset = dataset.shuffle(len(df)).batch(32)

model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(train_dataset, epochs=15)
```

Pandas

Constructing the dataset for the Functional API

```
dict_slices = tf.data.Dataset.from_tensor_slices((df.to_dict('list'),
                                                   target.values)).batch(16)
```

```
>>> for features, target in tfds.as_numpy(dict_slices.take(1)):
      for (feature, value), label in zip(features.items(), target):
         print('{} = {}\t Label = {}'.format(feature, value, label))

age  =  [63 67 67 37 41 56 62 57 63 53 57 56 56 44 52 57]     Label =  0
sex  =  [1 1 1 1 0 1 0 0 1 1 1 0 1 1 1 1]     Label =  1
cp  =  [1 4 4 3 2 2 4 4 4 4 4 2 3 2 3 3]     Label =  0
trestbps  =  [145 160 120 130 130 120 140 120 130 140 140 140 130 120 172 150]     Label =  0
chol  =  [233 286 229 250 204 236 268 354 254 203 192 294 256 263 199 168]     Label =  0
...
```

Pandas

Constructing the dataset for the Functional API

```
dict_slices = tf.data.Dataset.from_tensor_slices((df.to_dict('list'),
                                                  target.values)).batch(16)
```

```
>>> for features, target in tfds.as_numpy(dict_slices.take(1)):
      for (feature, value), label in zip(features.items(), target):
         print('{} = {}\t Label = {}'.format(feature, value, label))

age  =  [63 67 67 37 41 56 62 57 63 53 57 56 56 44 52 57]     Label =  0
sex  =  [1 1 1 1 0 1 0 0 1 1 1 0 1 1 1 1]     Label =  1
cp  =  [1 4 4 3 2 2 4 4 4 4 4 2 3 2 3 3]     Label =  0
trestbps  =  [145 160 120 130 130 120 140 120 130 140 140 140 130 120 172 150]     Label =  0
chol  =  [233 286 229 250 204 236 268 354 254 203 192 294 256 263 199 168]     Label =  0
...
```

Training the model (Functional)

```
# Constructing the inputs for all the dense features
inputs = {key: tf.keras.layers.Input(shape=(), name=key) for key in df.keys()}
x = tf.stack(list(inputs.values()), axis=-1)
x = tf.keras.layers.Dense(10, activation='relu')(x)

# The single output denoting the target's probability
output = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(inputs=inputs, outputs=output)
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(dict_slices, epochs=15)
```

Training the model (Functional)

```
# Constructing the inputs for all the dense features
inputs = {key: tf.keras.layers.Input(shape=(), name=key) for key in df.keys()}
x = tf.stack(list(inputs.values()), axis=-1)
x = tf.keras.layers.Dense(10, activation='relu')(x)

# The single output denoting the target's probability
output = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(inputs=inputs, outputs=output)
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(dict_slices, epochs=15)
```

Classifying species of flowers

```
DATA_URL = '[insert URL here]'
data_root_orig = tf.keras.utils.get_file(origin=DATA_URL,
                                         fname='flower_photos', untar=True)
data_root = pathlib.Path(data_root_orig)

# Load all the file paths in the directory
all_image_paths = list(data_root.glob('*/*'))
all_image_paths = [str(path) for path in all_image_paths]

# Gather the list of labels and create a labelmap
label_names = sorted(item.name for item in data_root.glob('*/') if item.is_dir())
label_to_index = dict((name, index) for index, name in enumerate(label_names))

# Use the label map to fetch all categorical labels
all_image_labels = [label_to_index[pathlib.Path(path).parent.name]
                    for path in all_image_paths]
```

Classifying species of flowers

```
DATA_URL = '[insert URL here]'
data_root_orig = tf.keras.utils.get_file(origin=DATA_URL,
                                         fname='flower_photos', untar=True)
data_root = pathlib.Path(data_root_orig)

# Load all the file paths in the directory
all_image_paths = list(data_root.glob('*/*'))
all_image_paths = [str(path) for path in all_image_paths]

# Gather the list of labels and create a labelmap
label_names = sorted(item.name for item in data_root.glob('*/') if item.is_dir())
label_to_index = dict((name, index) for index, name in enumerate(label_names))

# Use the label map to fetch all categorical labels
all_image_labels = [label_to_index[pathlib.Path(path).parent.name]
                    for path in all_image_paths]
```

Classifying species of flowers

```python
DATA_URL = '[insert URL here]'
data_root_orig = tf.keras.utils.get_file(origin=DATA_URL,
                                         fname='flower_photos', untar=True)
data_root = pathlib.Path(data_root_orig)

# Load all the file paths in the directory
all_image_paths = list(data_root.glob('*/*'))
all_image_paths = [str(path) for path in all_image_paths]

# Gather the list of labels and create a labelmap
label_names = sorted(item.name for item in data_root.glob('*/') if item.is_dir())
label_to_index = dict((name, index) for index, name in enumerate(label_names))

# Use the label map to fetch all categorical labels
all_image_labels = [label_to_index[pathlib.Path(path).parent.name]
                    for path in all_image_paths]
```

Classifying species of flowers

```
DATA_URL = '[insert URL here]'
data_root_orig = tf.keras.utils.get_file(origin=DATA_URL,
                                         fname='flower_photos', untar=True)
data_root = pathlib.Path(data_root_orig)

# Load all the file paths in the directory
all_image_paths = list(data_root.glob('*/*'))
all_image_paths = [str(path) for path in all_image_paths]

# Gather the list of labels and create a labelmap
label_names = sorted(item.name for item in data_root.glob('*/') if item.is_dir())
label_to_index = dict((name, index) for index, name in enumerate(label_names))

# Use the label map to fetch all categorical labels
all_image_labels = [label_to_index[pathlib.Path(path).parent.name]
                    for path in all_image_paths]
```

Preprocessing and creating the dataset

```python
path_ds = tf.data.Dataset.from_tensor_slices(all_image_paths)
label_ds = tf.data.Dataset.from_tensor_slices(all_image_labels)

def preprocess_image(path):
  image = tf.io.read_file(path)
  image = tf.image.decode_jpeg(image, channels=3)
  image = tf.image.resize(image, [192, 192])
  image /= 255.0  # normalize to [0,1] range
  return image

image_ds = path_ds.map(preprocess_image)
image_label_ds = tf.data.Dataset.zip((image_ds, label_ds))
```

Preprocessing and creating the dataset

```
path_ds = tf.data.Dataset.from_tensor_slices(all_image_paths)
label_ds = tf.data.Dataset.from_tensor_slices(all_image_labels)

def preprocess_image(path):
  image = tf.io.read_file(path)
  image = tf.image.decode_jpeg(image, channels=3)
  image = tf.image.resize(image, [192, 192])
  image /= 255.0  # normalize to [0,1] range
  return image

image_ds = path_ds.map(preprocess_image)
image_label_ds = tf.data.Dataset.zip((image_ds, label_ds))
```

Preprocessing and creating the dataset

```
path_ds = tf.data.Dataset.from_tensor_slices(all_image_paths)
label_ds = tf.data.Dataset.from_tensor_slices(all_image_labels)

def preprocess_image(path):
  image = tf.io.read_file(path)
  image = tf.image.decode_jpeg(image, channels=3)
  image = tf.image.resize(image, [192, 192])
  image /= 255.0  # normalize to [0,1] range
  return image

image_ds = path_ds.map(preprocess_image)
image_label_ds = tf.data.Dataset.zip((image_ds, label_ds))
```

Preprocessing and creating the dataset

```
path_ds = tf.data.Dataset.from_tensor_slices(all_image_paths)
label_ds = tf.data.Dataset.from_tensor_slices(all_image_labels)

def preprocess_image(path):
  image = tf.io.read_file(path)
  image = tf.image.decode_jpeg(image, channels=3)
  image = tf.image.resize(image, [192, 192])
  image /= 255.0  # normalize to [0,1] range
  return image

image_ds = path_ds.map(preprocess_image)
image_label_ds = tf.data.Dataset.zip((image_ds, label_ds))
```

Training the model

```
BATCH_SIZE = 32
ds = image_label_ds.shuffle(
                buffer_size=len(all_image_paths)).repeat().batch(BATCH_SIZE)

steps_per_epoch=tf.math.ceil(len(all_image_paths) / BATCH_SIZE).numpy()

model.fit(ds, epochs=1, steps_per_epoch=steps_per_epoch)
```

Predicting survivors with Titanic

```
train_file_path = tf.keras.utils.get_file(
    "train.csv","https://storage.googleapis.com/tf-datasets/titanic/train.csv")
test_file_path = tf.keras.utils.get_file(
    "Eval.csv", "https://storage.googleapis.com/tf-datasets/titanic/eval.csv")

def get_dataset(file_path, **kwargs):
  dataset = tf.data.experimental.make_csv_dataset(
      file_path,
      batch_size=5, # Artificially small to make examples easier to show.
      label_name='survived',
      na_value="?",
      num_epochs=1,
      ignore_errors=True,
      **kwargs)
  return dataset

raw_train_data = get_dataset(train_file_path)
raw_test_data = get_dataset(test_file_path)
```

CSV

Predicting survivors with Titanic

```
train_file_path = tf.keras.utils.get_file(
    "train.csv","https://storage.googleapis.com/tf-datasets/titanic/train.csv")
test_file_path = tf.keras.utils.get_file(
    "Eval.csv", "https://storage.googleapis.com/tf-datasets/titanic/eval.csv")

def get_dataset(file_path, **kwargs):
  dataset = tf.data.experimental.make_csv_dataset(
      file_path,
      batch_size=5, # Artificially small to make examples easier to show.
      label_name='survived',
      na_value="?",
      num_epochs=1,
      ignore_errors=True,
      **kwargs)
  return dataset

raw_train_data = get_dataset(train_file_path)
raw_test_data = get_dataset(test_file_path)
```

Predicting survivors with Titanic

```
train_file_path = tf.keras.utils.get_file(
    "train.csv","https://storage.googleapis.com/tf-datasets/titanic/train.csv")
test_file_path = tf.keras.utils.get_file(
    "Eval.csv", "https://storage.googleapis.com/tf-datasets/titanic/eval.csv")

def get_dataset(file_path, **kwargs):
  dataset = tf.data.experimental.make_csv_dataset(
      file_path,
      batch_size=5, # Artificially small to make examples easier to show.
      label_name='survived',
      na_value="?",
      num_epochs=1,
      ignore_errors=True,
      **kwargs)
  return dataset

raw_train_data = get_dataset(train_file_path)
raw_test_data = get_dataset(test_file_path)
```

What happens after loading the CSV?

```
def show_batch(dataset):
  for batch, label in dataset.take(1):
    for key, value in batch.items():
      print("{:20s}: {}".format(key,value.numpy()))

>>> show_batch(get_dataset(train_file_path))
sex                 : [b'female' b'female' b'female' b'male' b'male']
age                 : [40. 28. 52. 50. 34.]
n_siblings_spouses  : [0 0 1 0 1]
parch               : [0 0 0 0 0]
fare                : [13.      7.75   78.2667 13.      21.    ]
class               : [b'Second' b'Third' b'First' b'Second' b'Second']
deck                : [b'unknown' b'unknown' b'D' b'unknown' b'unknown']
embark_town         : [b'Southampton' b'Queenstown' b'Cherbourg' b'Southampton' ..]
alone               : [b'y' b'y' b'n' b'y' b'n']
```

Getting data from named columns

```
CSV_COLUMNS = ['survived', 'sex', 'age', 'n_siblings_spouses', 'parch', 'fare',
'class', 'deck', 'embark_town', 'alone']

temp_dataset = get_dataset(train_file_path, column_names=CSV_COLUMNS)

>>> show_batch(temp_dataset)

sex                 : [b'female' b'male' b'male' b'male' b'male']
age                 : [15. 29. 49. 35. 22.]
n_siblings_spouses  : [1 1 1 0 0]
parch               : [0 0 1 0 0]
fare                : [ 14.4542  21.     110.8833   7.125    7.125 ]
class               : [b'Third' b'Second' b'First' b'Third' b'Third']
deck                : [b'unknown' b'unknown' b'C' b'unknown' b'unknown']
embark_town         : [b'Cherbourg' b'Southampton' b'Cherbourg' b'Southampton'..]
alone               : [b'n' b'n' b'n' b'y' b'y']
```

Omitting some columns

```
SELECT_COLUMNS =['survived','age','n_siblings_spouses','class','deck','alone']
temp_dataset = get_dataset(train_file_path, select_columns=SELECT_COLUMNS)

>>> show_batch(temp_dataset)
age                  : [60. 34. 28. 40. 28.]
n_siblings_spouses   : [1 1 1 0 0]
class                : [b'Second' b'Third' b'Third' b'First' b'Third']
deck                 : [b'unknown' b'unknown' b'unknown' b'B' b'unknown']
alone                : [b'n' b'n' b'n' b'y' b'y']
```

CSV

Extracting features

```
 SELECT_COLUMNS = ['survived', 'age', 'n_siblings_spouses', 'parch', 'fare']

DEFAULTS = [0, 0.0, 0.0, 0.0, 0.0]
temp_dataset = get_dataset(train_file_path,
                           select_columns=SELECT_COLUMNS,
                           column_defaults=DEFAULTS)

# Function that will pack together all the columns:
def pack(features, label):
  return tf.stack(list(features.values()), axis=-1), label

packed_dataset = temp_dataset.map(pack)
```

Extracting features

```
 SELECT_COLUMNS = ['survived', 'age', 'n_siblings_spouses', 'parch', 'fare']


DEFAULTS = [0, 0.0, 0.0, 0.0, 0.0]
temp_dataset = get_dataset(train_file_path,
                           select_columns=SELECT_COLUMNS,
                           column_defaults=DEFAULTS)


# Function that will pack together all the columns:
def pack(features, label):
  return tf.stack(list(features.values()), axis=-1), label


packed_dataset = temp_dataset.map(pack)
```

Packing numeric features

```
 NUMERIC_FEATURES = ['age','n_siblings_spouses','parch', 'fare']

class PackNumericFeatures(object):
  def __init__(self, names):
    self.names = names

  def __call__(self, features, labels):
    numeric_freatures = [features.pop(name) for name in self.names]
    numeric_features = [tf.cast(feat, tf.float32)
             for feat in numeric_freatures]
    numeric_features = tf.stack(numeric_features, axis=-1)
    features['numeric'] = numeric_features

    return features, labels

packed_train_data = raw_train_data.map(
    PackNumericFeatures(NUMERIC_FEATURES))
packed_test_data = raw_test_data.map(
    PackNumericFeatures(NUMERIC_FEATURES))
```

CSV

Packing numeric features

```
 NUMERIC_FEATURES = ['age','n_siblings_spouses','parch', 'fare']

class PackNumericFeatures(object):
  def __init__(self, names):
    self.names = names

  def __call__(self, features, labels):
    numeric_freatures = [features.pop(name) for name in self.names]
    numeric_features = [tf.cast(feat, tf.float32)
             for feat in numeric_freatures]
    numeric_features = tf.stack(numeric_features, axis=-1)
    features['numeric'] = numeric_features

    return features, labels

packed_train_data = raw_train_data.map(
    PackNumericFeatures(NUMERIC_FEATURES))
packed_test_data = raw_test_data.map(
    PackNumericFeatures(NUMERIC_FEATURES))
```

Packing numeric features

```
 NUMERIC_FEATURES = ['age','n_siblings_spouses','parch', 'fare']

class PackNumericFeatures(object):
  def __init__(self, names):
    self.names = names

  def __call__(self, features, labels):
    numeric_freatures = [features.pop(name) for name in self.names]
    numeric_features = [tf.cast(feat, tf.float32)
             for feat in numeric_freatures]
    numeric_features = tf.stack(numeric_features, axis=-1)
    features['numeric'] = numeric_features

    return features, labels

packed_train_data = raw_train_data.map(
    PackNumericFeatures(NUMERIC_FEATURES))
packed_test_data = raw_test_data.map(
    PackNumericFeatures(NUMERIC_FEATURES))
```

Showing packed features

```
>>> show_batch(packed_train_data)
sex         : [b'male' b'male' ...]
class       : [b'First' b'Third' ...]
deck        : [b'unknown' b'unknown' ...]
embark_town : [b'Cherbourg' b'Southampton' ...]
alone       : [b'n' b'y'    ...]
numeric     : [[28.      1.  ...]
               [49.      0.  ...]
               [27.      0.  ...]
               [0.83     0.  ...]
               [28.      0.  ...]]
```

Normalizing features

```
 NUMERIC_FEATURES = ['age','n_siblings_spouses','parch', 'fare']

def normalize_numeric_data(data, mean, std):
  # Center the data
  return (data-mean)/std

desc = pd.read_csv(train_file_path)[NUMERIC_FEATURES].describe()

MEAN, STD = np.array(desc.T['mean']), np.array(desc.T['std'])

normalizer = functools.partial(normalize_numeric_data,
                               mean=MEAN,
                               std=STD)

numeric_column = tf.feature_column.numeric_column(
                                   'numeric',
                                   normalizer_fn=normalizer,
                                   shape=[len(NUMERIC_FEATURES)])
```

Normalizing features

```
 NUMERIC_FEATURES = ['age','n_siblings_spouses','parch', 'fare']

def normalize_numeric_data(data, mean, std):
  # Center the data
  return (data-mean)/std

desc = pd.read_csv(train_file_path)[NUMERIC_FEATURES].describe()

MEAN, STD = np.array(desc.T['mean']), np.array(desc.T['std'])

normalizer = functools.partial(normalize_numeric_data,
                               mean=MEAN,
                               std=STD)

numeric_column = tf.feature_column.numeric_column(
                                   'numeric',
                                   normalizer_fn=normalizer,
                                   shape=[len(NUMERIC_FEATURES)])
```

CSV

Normalizing features

```
 NUMERIC_FEATURES = ['age','n_siblings_spouses','parch', 'fare']

def normalize_numeric_data(data, mean, std):
  # Center the data
  return (data-mean)/std

desc = pd.read_csv(train_file_path)[NUMERIC_FEATURES].describe()

MEAN, STD = np.array(desc.T['mean']), np.array(desc.T['std'])

normalizer = functools.partial(normalize_numeric_data,
                               mean=MEAN,
                               std=STD)

numeric_column = tf.feature_column.numeric_column(
                                   'numeric',
                                   normalizer_fn=normalizer,
                                   shape=[len(NUMERIC_FEATURES)])
```

Normalizing features

```
 NUMERIC_FEATURES = ['age','n_siblings_spouses','parch', 'fare']

def normalize_numeric_data(data, mean, std):
  # Center the data
  return (data-mean)/std

desc = pd.read_csv(train_file_path)[NUMERIC_FEATURES].describe()

MEAN, STD = np.array(desc.T['mean']), np.array(desc.T['std'])

normalizer = functools.partial(normalize_numeric_data,
                               mean=MEAN,
                               std=STD)

numeric_column = tf.feature_column.numeric_column(
                                   'numeric',
                                   normalizer_fn=normalizer,
                                   shape=[len(NUMERIC_FEATURES)])
```

Now for the categorical features

```
    CATEGORIES = {
        'sex': ['male', 'female'],
        'class' : ['First', 'Second', 'Third'],
        'deck' : ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'],
        'embark_town' : ['Cherbourg', 'Southhampton', 'Queenstown'],
        'alone' : ['y', 'n']
    }
```

```
cat_feature_col = tf.feature_column.categorical_column_with_vocabulary_list(
                     key='class',
                     vocabulary_list=['First', 'Second', 'Third'])

categorical_column = tf.feature_column.indicator_column(cat_feature_col))
```

Training the model

```
dense_features= tf.keras.layers.DenseFeatures(categorical_columns+numeric_columns)

model = tf.keras.Sequential([
  dense_features,
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid'),
])

model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy'])

model.fit(packed_train_data, epochs=20)
```

Training the model

```
dense_features= tf.keras.layers.DenseFeatures(categorical_columns+numeric_columns)

model = tf.keras.Sequential([
  dense_features,
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid'),
])

model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy'])

model.fit(packed_train_data, epochs=20)
```

Training the model

```
dense_features= tf.keras.layers.DenseFeatures(categorical_columns+numeric_columns)

model = tf.keras.Sequential([
  dense_features,
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid'),
])

model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy'])

model.fit(packed_train_data, epochs=20)
```

Text

Identifying translators of a work

```
DIRECTORY_URL = 'https://storage.googleapis.com/download.tensorflow.org/data/illiad/'
FILE_NAMES = ['cowper.txt', 'derby.txt', 'butler.txt']

def labeler(example, index):
  return example, tf.cast(index, tf.int64)

labeled_data_sets = []
for i, file_name in enumerate(FILE_NAMES):
  file_path = tf.keras.utils.get_file(name, origin=DIRECTORY_URL+file_name)
  lines_dataset = tf.data.TextLineDataset(file_path)
  labeled_dataset = lines_dataset.map(lambda ex: labeler(ex, i))
  labeled_data_sets.append(labeled_dataset)
```

Identifying translators of a work

```
DIRECTORY_URL = 'https://storage.googleapis.com/download.tensorflow.org/data/illiad/'
FILE_NAMES = ['cowper.txt', 'derby.txt', 'butler.txt']

def labeler(example, index):
  return example, tf.cast(index, tf.int64)

labeled_data_sets = []
for i, file_name in enumerate(FILE_NAMES):
  file_path = tf.keras.utils.get_file(name, origin=DIRECTORY_URL+file_name)
  lines_dataset = tf.data.TextLineDataset(file_path)
  labeled_dataset = lines_dataset.map(lambda ex: labeler(ex, i))
  labeled_data_sets.append(labeled_dataset)
```

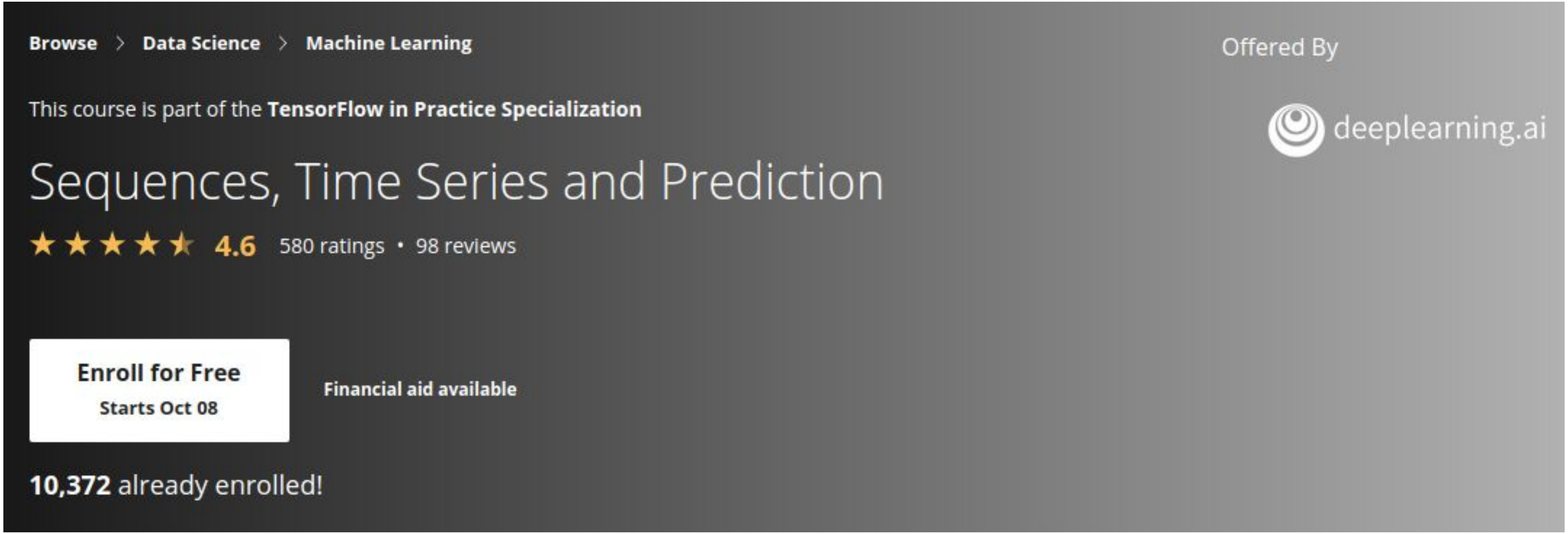
Preparing the dataset

```
dataset = labeled_data_sets[0]
for labeled_dataset in labeled_data_sets[1:]:
  dataset = dataset.concatenate(labeled_dataset)

dataset = dataset.shuffle(buffer_size=50000)

>>> for ex in dataset.take(5):
      print(ex[0].numpy(), ex[1].numpy())
b"Eight barbed arrows have I shot e'en now," 1
b'In thy own band; the Achaians shall for him,' 0
b"Upon their well-mann'd ships, should Heaven vouchsafe" 1
b'He shall not cozen me! Of him, enough!' 1
b'Turns flying, marks him with a steadfast eye,' 0
```

Text encoding

```
tokenizer = tfds.features.text.Tokenizer()

vocabulary_set = set()
for text_tensor, _ in all_labeled_data:
  some_tokens = tokenizer.tokenize(text_tensor.numpy())
  vocabulary_set.update(some_tokens)

vocab_size = len(vocabulary_set)
>>> vocab_size
17178
```

https://www.coursera.org/learn/natural-language-processing-tensorflow



Browse > Data Science > Machine Learning

Offered By

deeplearning.ai

This course is part of the **TensorFlow in Practice Specialization**

Sequences, Time Series and Prediction

★★★★☆ 4.6 580 ratings • 98 reviews

Enroll for Free
Starts Oct 08

Financial aid available

10,372 already enrolled!

Encode an example

```
# Show one of the labeled data
original_text = next(iter(all_labeled_data))[0].numpy()


# Create an text encoder with a fixed vocabulary set
encoder = tfds.features.text.TokenTextEncoder(vocabulary_set)


# Encode an example
encoded_text = encoder.encode(original_text)
```

```
Original text  b"As honour's meed, the mighty monarch gave."
Encoded text  [16814, 4289, 11591, 15925, 177, 10357, 11207, 16715]
```

Encoding all the examples

```
def encode(text_tensor, label):
  encoded_text = encoder.encode(text_tensor.numpy())
  return encoded_text, label

def encode_map_fn(text, label):
  return tf.py_function(encode, inp=[text, label],
                                  Tout=(tf.int64, tf.int64))

all_encoded_data = all_labeled_data.map(encode_map_fn)
```

Prepare the dataset

```
BUFFER_SIZE = 50000
BATCH_SIZE = 64
TAKE_SIZE = 5000

train_data = all_encoded_data.skip(TAKE_SIZE).shuffle(BUFFER_SIZE)
train_data = train_data.padded_batch(BATCH_SIZE, padded_shapes=([-1],[]))

test_data = all_encoded_data.take(TAKE_SIZE)
test_data = test_data.padded_batch(BATCH_SIZE, padded_shapes=([-1],[]))
```

Training the model

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.Sequential([
        tf.keras.layers.Dense(units, activation='relu') for units in [64, 64]
    ]),
    tf.keras.layers.Dense(3, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_data, epochs=3)
```