Chapter 3

Keras and Data Retrieval in TensorFlow 2

Implementing networks directly with the low-level API would require repeatedly coding common operations, which is inefficient. Keras simplifies this process by offering intuitive abstractions for layers and models, enabling developers to build, train, and evaluate deep learning systems more quickly.

Keras provides three main APIs: Sequential, Functional, and Sub-classing. The Sequential API is the simplest, supporting models with a single input and output in a straight layer-by-layer stack. The Functional API allows more flexibility, such as handling multiple inputs or outputs, and parallel connections. Finally, the Sub-classing API is the most advanced, enabling custom layers and models defined as Python classes. Figure 1 compares these three APIs in terms of ease of use and flexibility.

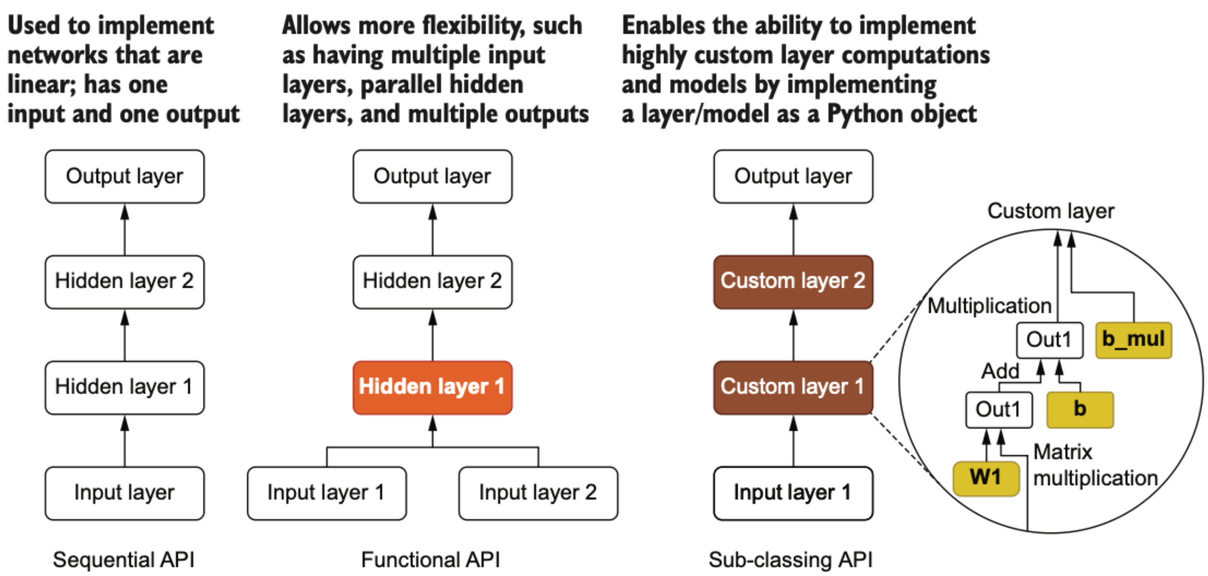

Figure 1 Sequential, functional, and sub-classing APIs in comparison

To demonstrate, the Iris dataset is introduced. It contains four flower measurements (sepal length, sepal width, petal length, petal width) for three iris species. After downloading and preprocessing—renaming columns, encoding labels, centering features, and applying one-hot encoding—the dataset is used to train different models. With the Sequential API, a simple multilayer perceptron (Model A) is implemented, achieving around 74% training accuracy in 25 epochs. With the Functional API, a second model (Model B) is created to take both original features and principal components (from PCA) as inputs. Although the architecture is more flexible, accuracy improvements are marginal. Finally, the Sub-classing API is used to implement a custom layer with an additional multiplicative bias (Model C). While results didn't improve significantly, this exercise illustrates the power of defining entirely new layer behaviors. Table 1 summarizes the pros and cons of the three Keras APIs: Sequential is concise but limited, Functional is flexible but requires careful wiring of layers, and Sub-classing is most powerful but more complex to debug.

Table 1 Pros and cons of using various Keras APIs

Sequential API	Pros	Models implemented with the Sequential API are easy to under-stand and are concise
	Cons	Cannot implement models having complex architectural characteris-tics such as multiple inputs/outputs
Functional API	Pros	Can be used to implement models with complex architectural ele-ments such as multiple inputs/outputs.
	Cons	The developer needs to manually connect various layers correctly and create a model.
Sub-classing API	Pros	Can create custom layers and models that are not provided as standard layers.
	Cons	Requires thorough understanding of low-level functionality provided by TensorFlow. Due to the user-defined nature, it can lead to instabilities and diffi-culties in debugging.

Reproduce code for Iris preprocessing + sequential model:

```
# imports

import requests

import pandas as pd

import tensorflow as tf

from tensorflow.keras.layers import Dense
```

```
from tensorflow.keras.models import Sequential
import tensorflow.keras.backend as K
url      =      "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
r = requests.get(url)
with open('iris.data', 'wb') as f:
    f.write(r.content)


iris_df = pd.read_csv('iris.data', header=None)
iris_df.columns = ['sepal_length', 'sepal_width', 'petal_width', 'petal_length', 'label']
iris_df["label"] = iris_df["label"].map({
    'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2
})


iris_df = iris_df.sample(frac=1.0, random_state=4321)
x   =   iris_df[["sepal_length",   "sepal_width",   "petal_width", "petal_length"]]
x = x - x.mean(axis=0)
y = tf.one_hot(iris_df["label"], depth=3)


K.clear_session()
model = Sequential([
    Dense(32, activation='relu', input_shape=(4,)),
    Dense(16, activation='relu'),
    Dense(3, activation='softmax')
])
```

```
model.compile(loss='categorical_crossentropy',  optimizer='adam',
metrics=['acc'])

model.summary()

model.fit(x, y, batch_size=64, epochs=25)
```

A model is only as good as the data fed into it, so TensorFlow provides multiple input pipeline solutions. The tf.data API enables construction of efficient pipelines, for example reading flower images and labels from a CSV, decoding and resizing them, applying one-hot encoding, shuffling, and batching before feeding into a model. Figure 2 illustrates such a pipeline from raw files to tensors ready for training. Alternatively, Keras DataGenerators (e.g., ImageDataGenerator) provide a quicker, less customizable method, often sufficient for small to medium-scale projects. Finally, the tensorflow-datasets (tfds) library offers ready-to-use standard datasets like CIFAR-10, IMDB reviews, and ImageNet.

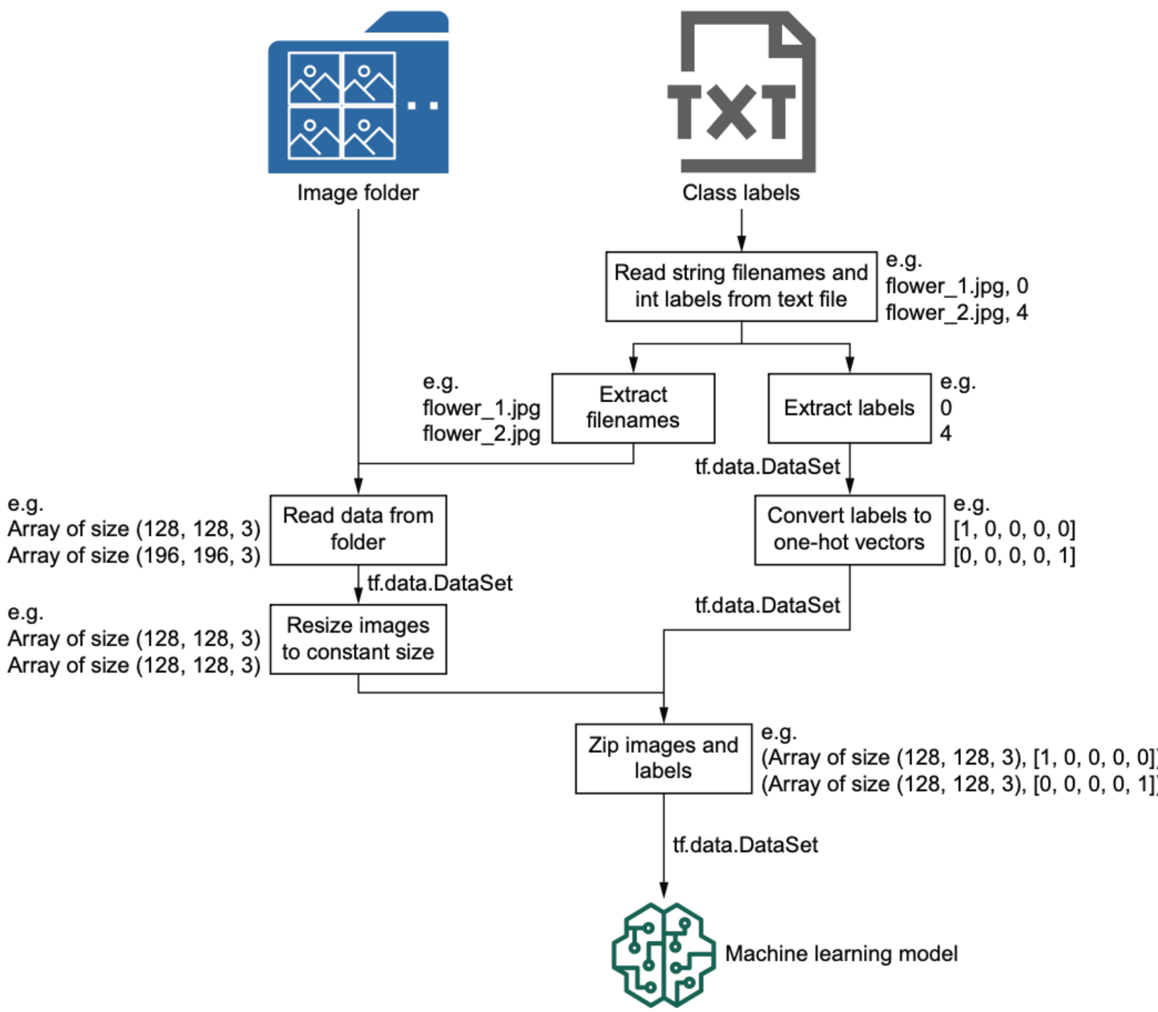

Figure 2 The input pipeline that you'll be developing using the tf.data API

Each of these approaches to modeling and data retrieval comes with strengths and limitations. Sequential modeling is concise but rigid. The functional API balances flexibility with clarity, supporting most real-world architectures. Subclassing provides ultimate control at the cost of complexity. Likewise, tf.data is powerful and efficient but requires more code; data generators are concise but limited; and tensorflow-datasets is convenient but restricted to supported datasets.

Together, these tools create a layered ecosystem. Beginners can build simple models with minimal effort, while advanced users can extend the framework to create highly specialized models and pipelines. By understanding when to choose simplicity and when to embrace complexity, practitioners can design effective systems that are both maintainable and scalable.