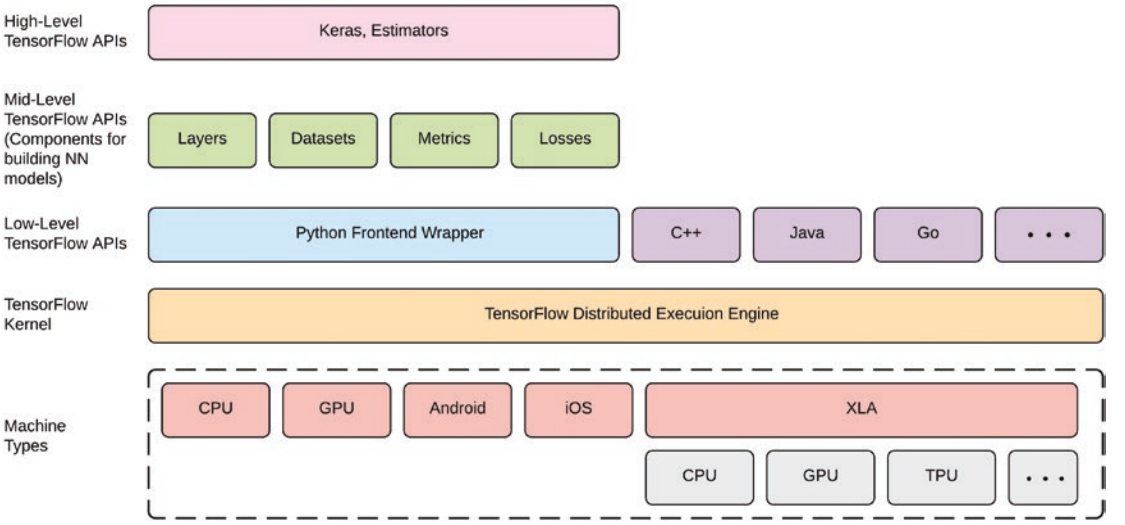

Figure 30-1. *TensorFlow API hierarchy*

The Low-Level TensorFlow APIs

The low-level API gives the tools for building network graphs from the ground up using mathematical operations. This API level affords the greatest level of flexibility to tweak and tune the model as desired. Moreover, the higher-level APIs implement low-level operations under the hood.

The Mid-Level TensorFlow APIs

TensorFlow provides a set of reusable packages for simplifying the process involved in creating neural network models. Some examples of these functions include the layers **(tf.keras.layers)**, Datasets **(tf.data)**, metrics **(tf.keras.metrics)**, loss **(tf.keras.losses)**, and FeatureColumns **(tf.feature_column)** packages.

Layers

The layers package **(tf.keras.layers)** provides a handy set of functions to simplify the construction of layers in a neural network architecture. For example, consider the convolutional network architecture in Figure 30-2 and how the layers API simplifies the creation of the network layers.

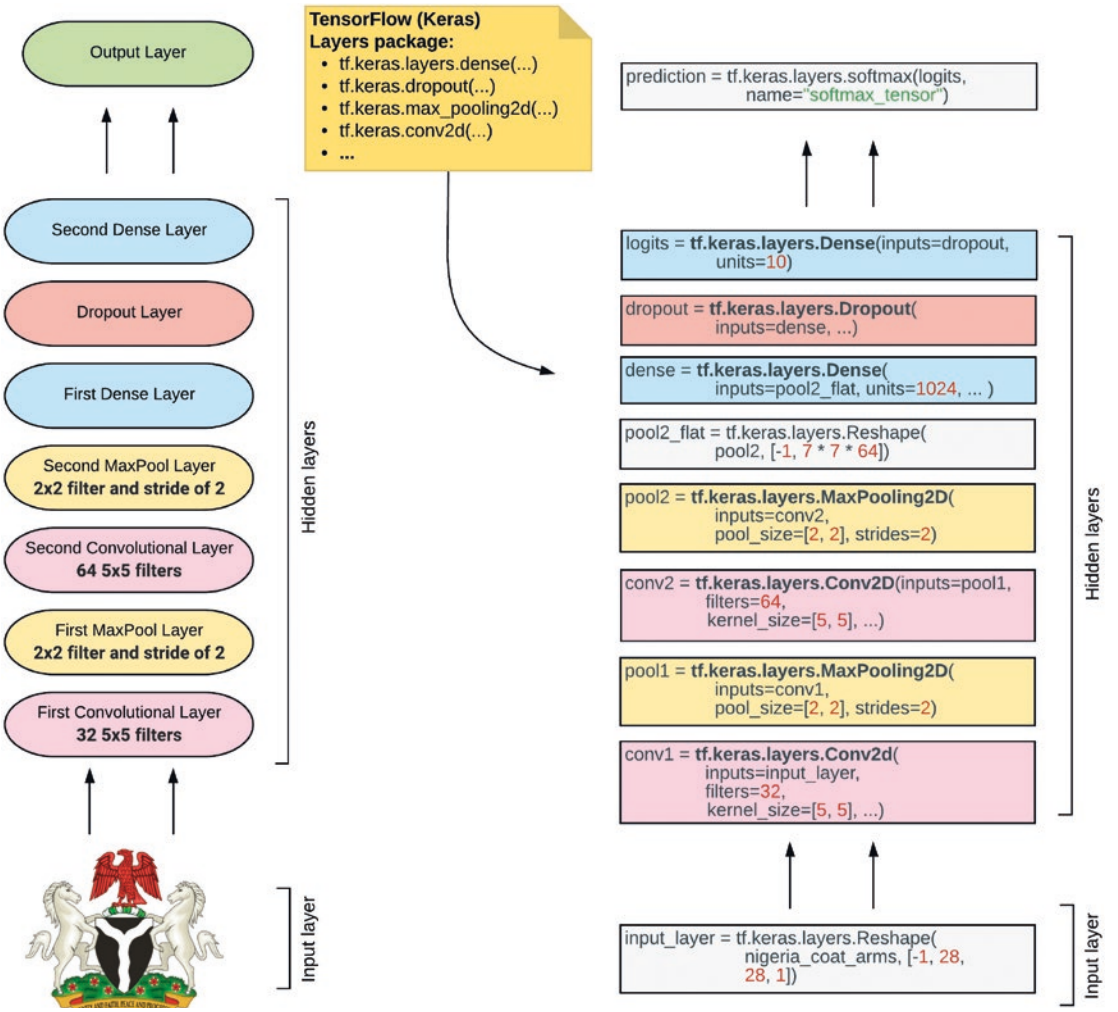

Figure 30-2. *Using the layers API to simplify creating the layers of a neural network*

Datasets

The Dataset package **(tf.data)** provides a convenient set of high-level functions for creating complex dataset input pipelines. The goal of the Dataset package is to have a fast, flexible, and easy-to-use interface for fetching data from various data sources, performing data transform operations on them before passing them as inputs to the learning model. The Dataset API provides a more efficient means of fetching records from a dataset. The major classes of the Dataset API are illustrated in Figure 30-3.

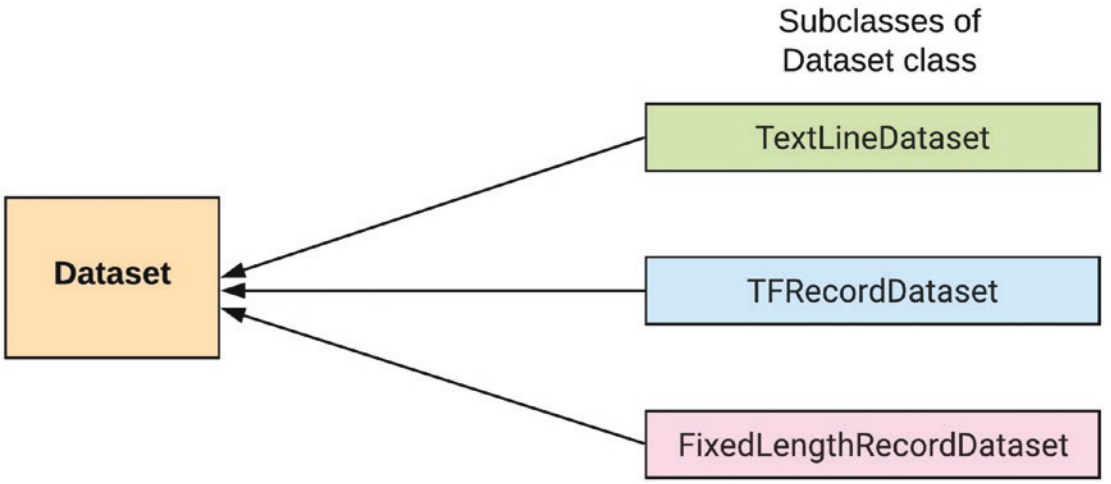

Figure 30-3. *Dataset API class hierarchy*

From the illustration in Figure 30-3, the subclasses perform the following functions:

- TextLineDataset: This class is used for reading lines from text files.
- TFRecordDataset: This class is responsible for reading records from TFRecord files. A TFRecord file is a TensorFlow binary storage format. It is faster and easier to work with data stored as TFRecord files as opposed to raw data files. Working with TFRecord also makes the data input pipeline more easily aligned for applying vital transformations such as shuffling and returning data in batches.
- FixedLengthRecordDataset: This class is responsible for reading records of fixed sizes from binary files.

FeatureColumns

FeatureColumns **tf.feature_column** is a TensorFlow functionality for describing the features of the dataset that will be fed into a high-level Keras or Estimator models for training and validation. FeatureColumns makes it easy to prepare data for modeling by carrying out tasks such as the conversion of categorical features of the dataset into a one-hot encoded vector.

The **feature_column** API is broadly divided into two categories; they are the categorical and dense columns. The categories and subsequent functions are illustrated in Figure 30-4.

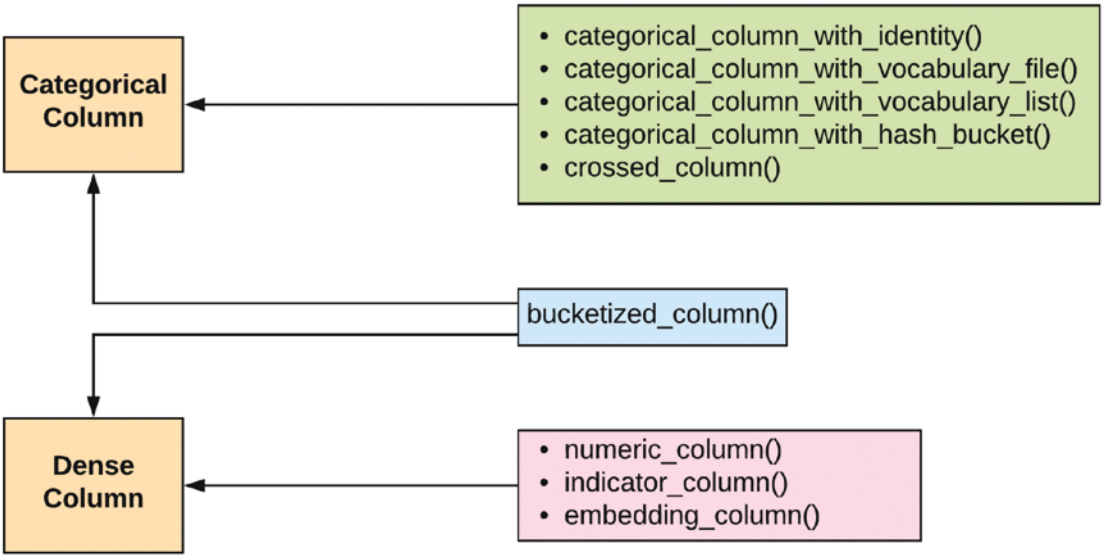

Figure 30-4. *Function calls of the Feature Column API*

Let's go through each API function briefly in Table 30-1.

***Table 30-1. tf.feature_column** API Functions*

Function name	Description
Numeric column – **tf.feature_column.numeric_column()**	This is a high-level wrapper for numeric features in the dataset.
Indicator column – **tf.feature_column.indicator_column()**	The indicator column takes as input a categorical column and transforms it into a one-hot encoded vector.
Embedding column – **tf.feature_column.embedding_column()**	The embedding column function transforms a categorical column with multiple levels or classes into a lower-dimensional numeric representation that captures the relationships between the categories. Using embeddings mitigates the problem of a large sparse vector (an array with mostly zeros) created via one-hot encoding for a dataset feature with lots of different classes.

(*continued*)

Table 30-1. (*continued*)

Function name	Description
Categorical column with identity – **tf.feature_column.categorical_column_with_identity()**	This function creates a one-hot encoded output of a categorical column containing identities, e.g, [‘0’, ‘1’, ‘2’, ‘3’].
Categorical column with vocabulary list – **tf.feature_column.categorical_ column_with_vocabulary_list()**	This function creates a one-hot encoded output of a categorical column with strings. It maps each string to an integer based on a vocabulary list. However, if the vocabulary list is long, it is best to create a file containing the vocabulary and use the function **tf.feature_column.categorical_ column_with_vocabulary_file().**
Categorical column with hash bucket – **tf.feature_column.categorical_ column_with_hash_buckets()**	This function specifies the number of categories by using the hash of the inputs. It is used when it is not possible to create a vocabulary for the number of categories due to memory considerations.
Crossed column – **tf.feature_columns.crossed_column()**	The function gives the ability to combine multiple input features into a single input feature.
Bucketized column – **tf.feature_column.bucketized_column()**	The function splits a column of numerical inputs into buckets to form new classes based on a specified set of numerical ranges.

The High-Level TensorFlow APIs

The high-level API provides simplified API calls that encapsulate lots of the details that are typically involved in creating a deep learning TensorFlow model. These high-level abstractions make it easier to develop powerful deep learning models quickly with fewer lines of code.