# TensorFlow in Action (Part 1)

## Chapter 3 : Keras and Data Retrieval in TensorFlow 2

This chapter introduces **Keras** as the high-level API in TensorFlow 2 for defining and training neural networks, and then focuses on how to build robust input pipelines for those models. It starts with a small image-classification example and uses it to compare three Keras model-building styles: the Sequential API, the Functional API, and the subclassing API. With Sequential, you stack layers in a straight line; with the Functional API, you define arbitrary DAG-style architectures (multiple inputs/outputs, skip connections); and with subclassing, you write custom tf.keras.Model classes for maximum flexibility.

Once the basic model-building patterns are clear, the chapter turns to **data ingestion**, which is crucial because deep learning models expect large, well-structured batches rather than ad-hoc Python loops. First, it uses the tf.data API: you create tf.data.Dataset objects from file paths and labels, then chain transformations like map, shuffle, batch, and prefetch to build efficient pipelines that can decode images, normalize them, and feed them to model.fit in a streaming fashion. The example with a flower image dataset demonstrates how you can assemble an end-to-end pipeline (folder structure → dataset → preprocessing → batched tensors) that trains a small ConvNet with just a few lines of glue code.

The chapter then shows two alternative data-loading tools. Keras **DataGenerators**, such as ImageDataGenerator, provide a simpler but less flexible way to load and optionally augment images directly from directories or from a pandas DataFrame using helpers like flow_from_dataframe, which can quickly produce batches of tensors and labels from a CSV mapping filenames to classes. Finally, the tensorflow-datasets (tfds) package is introduced as the easiest option when you use standard benchmark datasets; you can load data like CIFAR-10 or IMDB reviews with one call to tfds.load, getting pre-split, preformatted datasets ready to plug into tf.data and Keras. The trade-off is clear: tf.data offers maximum flexibility and performance, Keras generators are convenient for many image tasks, and tfds is perfect when your dataset is already supported.