Keras and Data Retrieval in TensorFlow 2

Chapter ini comprehensive guide Keras APIs Sequential, Functional, Subclassing) dan data loading strategies production-grade. Keras adalah high-level API yang membuat deep learning accessible sambil tetap memberikan low-level control. [1]

Keras Model-Building APIs: Complete Comparison

1. Sequential API Linear Stack)

Untuk simple architectures:

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(784,)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

Limitations: Single input/output, no layer sharing.

2. Functional API Complex Architectures)

Multi-input/output, layer sharing, dynamic architectures:

```
inputs = tf.keras.Input(shape=(784,))
x1 = tf.keras.layers.Dense(128, activation='relu')(inputs)
x2 = tf.keras.layers.Dense(64, activation='relu')(inputs)
x = tf.keras.layers.concatenate([x1, x2])
outputs = tf.keras.layers.Dense(10, activation='softmax')(x)
model = tf.keras.Model(inputs, outputs)
```

Use cases: Multi-modal data, Siamese networks, encoder-decoder.

3. Subclassing API Maximum Flexibility)

Custom layers/models seperti PyTorch `nn.Module`:

```
class CustomCNN(tf.keras.Model):
    def __init__(self, num_classes=10):
        super().__init__()
        self.conv1 = tf.keras.layers.Conv2D(32, 3, activation='relu')
        self.pool1 = tf.keras.layers.MaxPool2D()
        self.dense = tf.keras.layers.Dense(num_classes)

    def call(self, inputs, training=False):
        x = self.conv1(inputs)
        x = self.pool1(x)
        x = tf.keras.layers.GlobalAvgPool2D()(x)
        return self.dense(x)
```

Best for: Research, custom layers, complex control flow.[1]

Production Data Retrieval Strategies

1. `tf.data` API (High-Performance)

Production-grade pipelines:

```
def preprocess(image, label):
    image = tf.cast(image, tf.float32) / 255.0
    image = tf.image.resize(image, (224, 224))
    return image, label

train_ds = tf.data.Dataset.from_tensor_slices((images, labels))
train_ds = train_ds.map(preprocess).batch(32).prefetch(tf.data.AUTOTUNE)
```

Key optimizations:

- `.cache()`: Cache preprocessed data in memory
- `.shuffle(buffer_size)`: Randomize order
- `.prefetch(AUTOTUNE)`: Overlap preprocessing + training
- `.repeat()`: Infinite dataset looping

2. Keras DataGenerators

Legacy API untuk image augmentation:

```
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=20, width_shift_range=0.2, zoom_range=0.2
)
```

3. `tensorflow-datasets`

60+ clean datasets siap pakai:

```
import tensorflow_datasets as tfds
(ds_train, ds_test), info = tfds.load('mnist', split=['train', 'test'], as_supervised=Tru
```

Kesimpulan

Functional API + `tf.data` adalah gold standard untuk production TensorFlow workflows. Hindari Keras generators untuk large-scale training.[1]

```
encoder = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(32, activation='relu')  # Latent space
])

decoder = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(784, activation='sigmoid')
])

autoencoder = tf.keras.Model(noisy_input, decoder(encoder(noisy_input)))
autoencoder.compile(loss='mse', optimizer='adam')
```