Chapter 4

Dipping Toes in Deep Learning

This chapter introduces three fundamental types of deep neural networks—fully connected networks (FCNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs)—and demonstrates how to implement them in TensorFlow 2 using Keras. Each model is aligned with a different type of data: FCNs are suited for general vector-based tasks, CNNs excel in image-related problems, and RNNs specialize in handling sequential and time-dependent data.

The discussion begins with FCNs, illustrated through the task of restoring corrupted images of handwritten digits using the MNIST dataset. An autoencoder is introduced as a special type of FCN designed for unsupervised learning, where the goal is to reconstruct inputs rather than classify them. The model architecture consists of four dense layers: an input layer of 784 nodes (corresponding to the 28×28 image pixels), two hidden layers of 64 and 32 nodes with ReLU activation, a mirrored hidden layer of 64 nodes, and finally an output layer of 784 nodes with tanh activation to ensure outputs fall between –1 and 1. Training is carried out by feeding synthetically corrupted images as input and using the clean images as targets, optimizing the mean squared error loss with the Adam optimizer. After ten epochs, the reconstruction error reduces significantly, demonstrating the model's ability to restore digits effectively. Figure 1 show clean digits, corrupted versions, the structure of an autoencoder, and the restored outputs, highlighting how autoencoders can learn robust feature representations without labels.

Figure 1 Images restored by the model.

The chapter then transitions to CNNs, using the CIFAR-10 dataset to model vehicle classification as an example. CNNs are introduced as architectures designed to preserve spatial information in images while being parameter-efficient through local convolution kernels. Their structure typically alternates between convolutional layers, which extract features such as edges and shapes, and pooling layers, which reduce dimensionality while

introducing translation invariance. Fully connected layers at the end map features to class probabilities. The first CNN implementation attempts to use large kernels but fails with a negative dimension error, which arises when layer configurations shrink the output too much. A corrected version employs smaller 3×3 kernels, controlled strides, and interleaved max-pooling, achieving around 72% training accuracy after 25 epochs. Throughout this section, the text emphasizes hyperparameters such as kernel size, stride, and padding, as well as the risk of dense layers after convolutions becoming parameter bottlenecks. Figures 2 and 3 illustrate the CIFAR-10 data, the mechanics of convolutions and pooling, and how parameter choices affect dimensionality.

Figure 2 Sample images from cifar-10 data set along with their labels

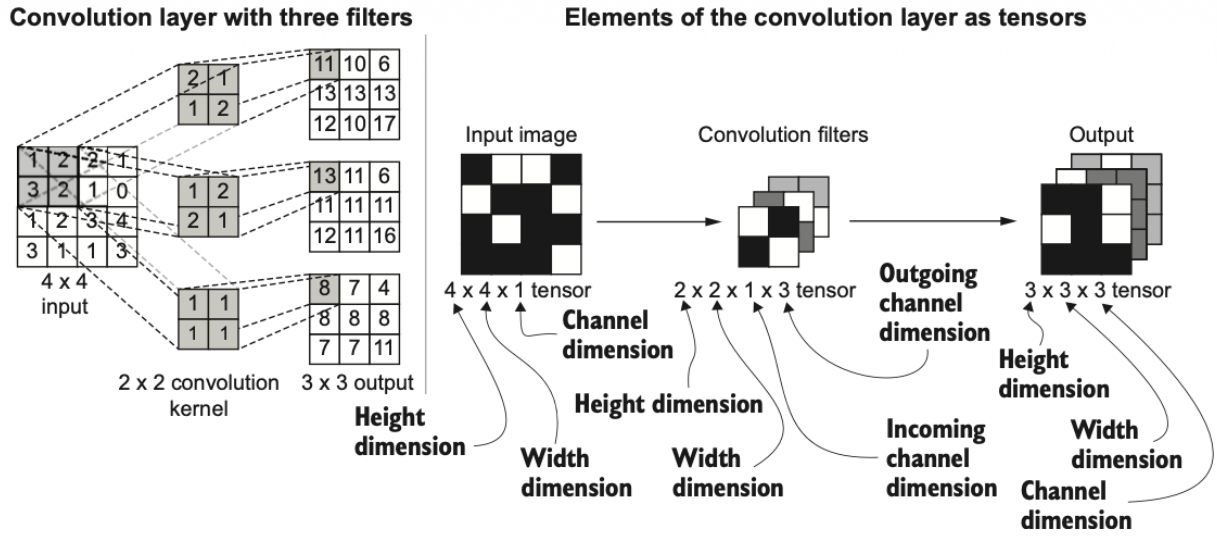
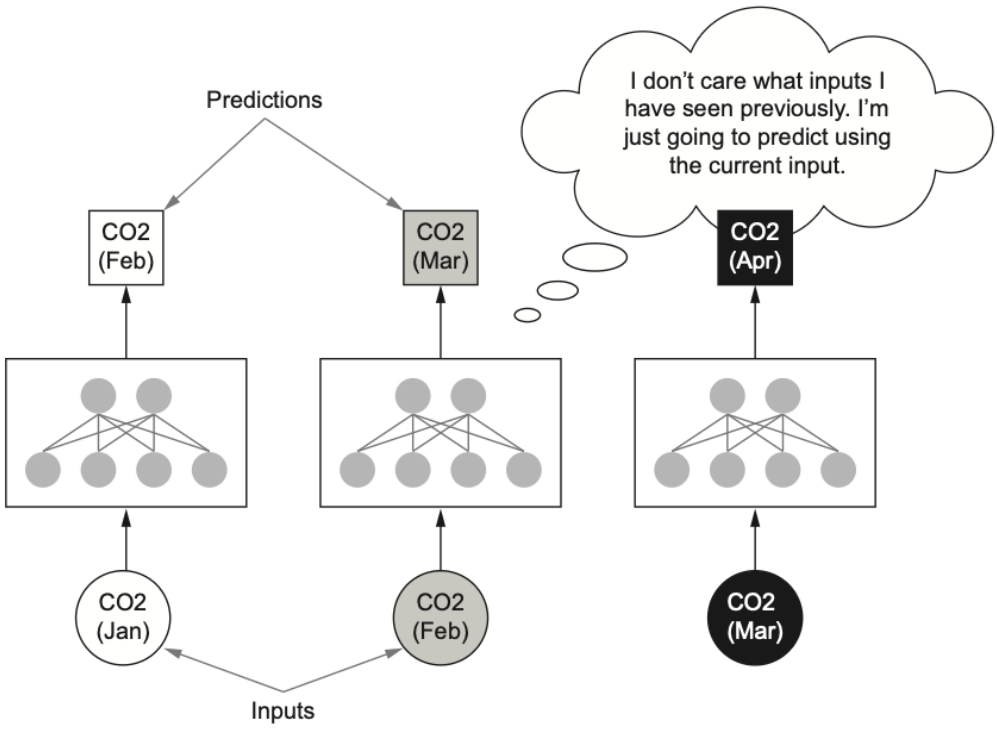


Figure 3 A computation of a convolution layer with multiple filters (randomly initialized)

Finally, the chapter explores RNNs through the task of predicting future atmospheric CO_2 concentrations based on historical monthly data spanning over three decades. Unlike feed-forward networks, which treat inputs as independent, RNNs incorporate memory of past inputs, making them ideal for sequential data. The raw dataset consists of date and average CO_2 values, which exhibit a long-term upward trend. To stabilize the data and make it more suitable for modeling, the series is transformed into relative differences between consecutive months, reducing the values to a narrower range. Input sequences are then generated by taking the previous twelve months of data to predict the next month, ensuring temporal dependencies are preserved. The text demonstrates how batching can be applied while maintaining sequence order, and Figures 4 and 5 visualize the operational differences between feed-forward and recurrent models, and the transformation of the data

How a feed-forward neural network operates



How a recurrent neural network operates

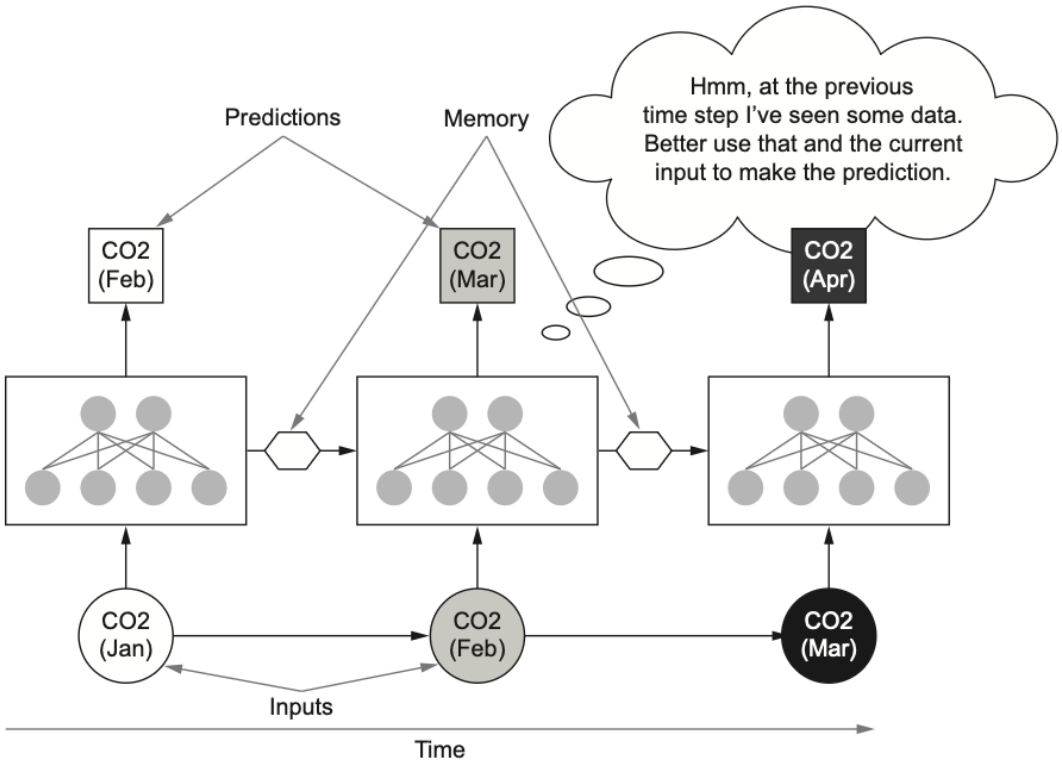


Figure 4 The operational difference between a feed-forward network and an RNN

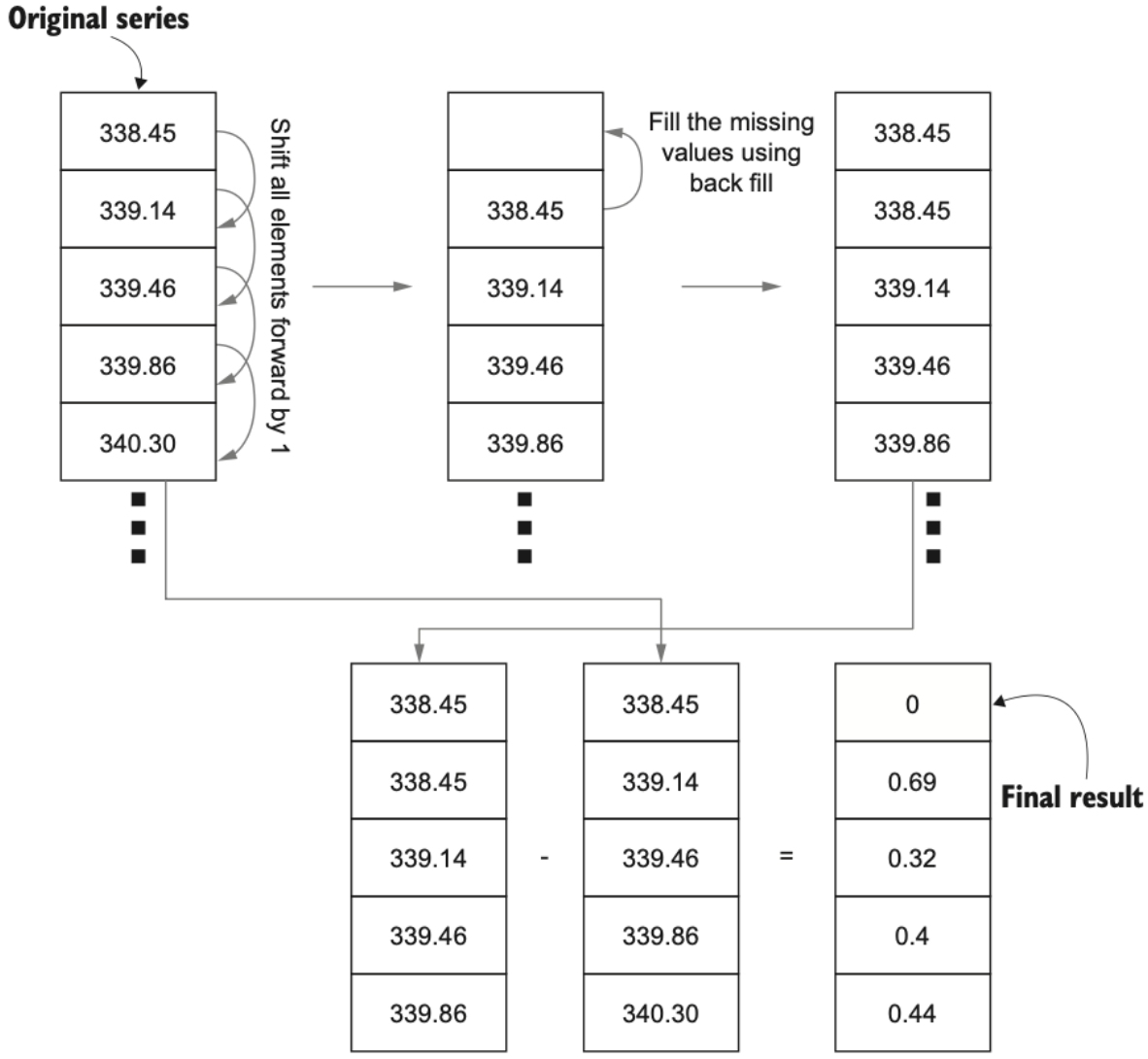


Figure 5 Transformations taking place going from the original Average series to the Average Diff series

In summary, this chapter illustrates how different deep learning architectures are matched to specific problem types. Autoencoders as FCNs provide a foundation for unsupervised feature learning and data reconstruction, CNNs dominate computer vision tasks by capturing hierarchical spatial patterns, and RNNs shine in sequential domains such as time series prediction. Together, they form the building blocks for more advanced architectures that will be explored in subsequent chapters.

Reproduces code for MNIST preprocessing:

```
import numpy as np

from tensorflow.keras.datasets.mnist import load_data
```

```
(x_train, y_train), (x_test, y_test) = load_data()

norm_x_train = ((x_train - 128.0) / 128.0).reshape([-1, 784])

def generate_masked_inputs(x, p, seed=None):
    if seed:
        np.random.seed(seed)
    mask = np.random.binomial(n=1, p=p, size=x.shape).astype('float32')
    return x * mask

masked_x_train = generate_masked_inputs(norm_x_train, 0.5)
```