7.6. Convolutional Neural Networks (LeNet) (https://colab.research.google.com/github/d2l-ai/d2l-tensorflow-colab/blob/master/chapter_convolutional-neural-networks/lenet.ipynb)

COLAB [TENSORFLOW]

SAGEMAKER STUDIO LAB

(https://studiolab.sagemaker.aws/import/github/d2l-ai/d2l-pytorch-sagemaker-studio-lab/blob/main/GettingStarted-D2L.ipynb)

We now have all the ingredients required to assemble a fully-functional CNN. In our earlier encounter with image data, we applied a linear model with softmax regression (Section 4.4 (../chapter_linear-classification/softmax-regression-scratch.html#sec-softmax-scratch)) and an MLP (Section 5.2 (../chapter_multilayer-perceptrons/mlp-implementation.html#sec-mlp-implementation)) to pictures of clothing in the Fashion-MNIST dataset. To make such data amenable we first flattened each image from a 28×28 matrix into a fixed-length 784-dimensional vector, and thereafter processed them in fully connected layers. Now that we have a handle on convolutional layers, we can retain the spatial structure in our images. As an additional benefit of replacing fully connected layers with convolutional layers, we will enjoy more parsimonious models that require far fewer parameters.

In this section, we will introduce *LeNet*, among the first published CNNs to capture wide attention for its performance on computer vision tasks. The model was introduced by (and named for) Yann LeCun, then a researcher at AT&T Bell Labs, for the purpose of recognizing handwritten digits in images (LeCun *et al.*, 1998 (../chapter_references/zreferences.html#id161)). This work represented the culmination of a decade of research developing the technology; LeCun's team published the first study to successfully train CNNs via backpropagation (LeCun *et al.*, 1989 (../chapter_references/zreferences.html#id160)).

At the time LeNet achieved outstanding results matching the performance of support vector machines, then a dominant approach in supervised learning, achieving an error rate of less than 1% per digit. LeNet was eventually adapted to recognize digits for processing deposits in ATM machines. To this day, some ATMs still run the code that Yann LeCun and his colleague Leon Bottou wrote in the 1990s!

PYTORCH MXNET JAX TENSORFLOW

```
import tensorflow as tf
from d2l import tensorflow as d2l
```

7.6.1. LeNet

At a high level, LeNet (LeNet-5) consists of two parts: (i) a convolutional encoder consisting of two convolutional layers; and (ii) a dense block consisting of three fully connected layers. The architecture is summarized in Fig. 7.6.1.

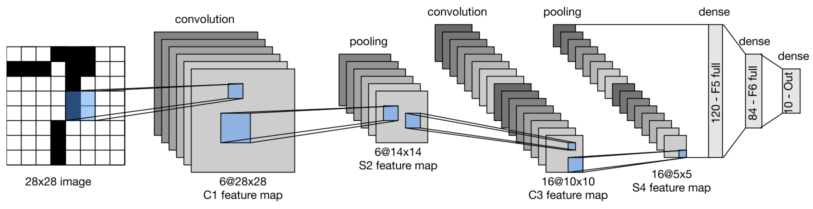

Fig. 7.6.1 Data flow in LeNet. The input is a handwritten digit, the output is a probability over 10 possible outcomes.

The basic units in each convolutional block are a convolutional layer, a sigmoid activation function, and a subsequent average pooling operation. Note that while ReLUs and max-pooling work better, they had not yet been discovered. Each convolutional layer uses a 5×5 kernel and a sigmoid activation function. These layers map spatially arranged inputs to a number of two-dimensional feature maps, typically increasing the number of channels. The first convolutional layer has 6 output channels, while the second has 16. Each 2×2 pooling operation (stride 2) reduces dimensionality by a factor of 4 via spatial downsampling. The convolutional block emits an output with shape given by (batch size, number of channel, height, width).

In order to pass output from the convolutional block to the dense block, we must flatten each example in the minibatch. In other words, we take this four-dimensional input and transform it into the two-dimensional input expected by fully connected layers: as a reminder, the two-dimensional representation that we desire uses the first dimension to index examples in the minibatch and the second to give the flat vector representation of each example. LeNet's dense block has three fully connected layers, with 120, 84, and 10 outputs, respectively. Because we are still performing classification, the 10-dimensional output layer corresponds to the number of possible output classes.

While getting to the point where you truly understand what is going on inside LeNet may have taken a bit of work, we hope that the following code snippet will convince you that implementing such models with modern deep learning frameworks is remarkably simple. We need only to instantiate a `Sequential` block and chain together the appropriate layers, using Xavier