```
!pip install d2l==1.0.3
```

Show hidden output

COSE474-2024F: Deep Learning

4.2. The Image Classification Dataset

One widely used dataset for image classification is the MNIST dataset (LeCun et al., 1998) of handwritten digits. At the time of its release in the 1990s it posed a formidable challenge to most machine learning algorithms, consisting of 60,000 images of pixels resolution (plus a test dataset of 10,000 images). To put things into perspective, back in 1995, a Sun SPARCStation 5 with a whopping 64MB of RAM and a blistering 5 MFLOPs was considered state of the art equipment for machine learning at AT&T Bell Laboratories. Achieving high accuracy on digit recognition was a key component in automating letter sorting for the USPS in the 1990s. Deep networks such as LeNet-5 (LeCun et al., 1995), support vector machines with invariances (Schölkopf et al., 1996), and tangent distance classifiers (Simard et al., 1998) all could reach error rates below 1%.

For over a decade, MNIST served as the point of reference for comparing machine learning algorithms. While it had a good run as a benchmark dataset, even simple models by today's standards achieve classification accuracy over 95%, making it unsuitable for distinguishing between strong models and weaker ones. Even more, the dataset allows for very high levels of accuracy, not typically seen in many classification problems. This skewed algorithmic development towards specific families of algorithms that can take advantage of clean datasets, such as active set methods and boundary-seeking active set algorithms. Today, MNIST serves as more of a sanity check than as a benchmark. ImageNet (Deng et al., 2009) poses a much more relevant challenge. Unfortunately, ImageNet is too large for many of the examples and illustrations in this book, as it would take too long to train to make the examples interactive. As a substitute we will focus our discussion in the coming sections on the qualitatively similar, but much smaller Fashion-MNIST dataset (Xiao et al., 2017) which was released in 2017. It contains images of 10 categories of clothing at 28×28 pixels resolution.

```
%matplotlib inline
import time
import torch
import torchvision
from torchvision import transforms
from d2l import torch as d2l

d2l.use_svg_display()
```

4.2.1. Loading the Dataset

Since the Fashion-MNIST dataset is so useful, all major frameworks provide preprocessed versions of it. We can download and read it into memory using built-in framework utilities.

```
class FashionMNIST(d2l.DataModule):
    """The Fashion-MNIST dataset."""
    def __init__(self, batch_size=64, resize=(28, 28)):
        super().__init__()
        self.save_hyperparameters()
        trans = transforms.Compose([transforms.Resize(resize),
                                    transforms.ToTensor()])
        self.train = torchvision.datasets.FashionMNIST(
            root=self.root, train=True, transform=trans, download=True)
        self.val = torchvision.datasets.FashionMNIST(
            root=self.root, train=False, transform=trans, download=True)
```

Fashion-MNIST consists of images from 10 categories, each represented by 6000 images in the training dataset and by 1000 in the test dataset. A test dataset is used for evaluating model performance (it must not be used for training). Consequently the training set and the test set contain 60,000 and 10,000 images, respectively.

```
data = FashionMNIST(resize=(32, 32))
len(data.train), len(data.val)
```

```
(60000, 10000)
```

The images are grayscale and upscaled to 32×32 pixels in resolution above. This is similar to the original MNIST dataset which consisted of (binary) black and white images. Note, though, that most modern image data has three channels (red, green, blue) and that hyperspectral images can have in excess of 100 channels (the HyMap sensor has 126 channels). By convention we store an image as a $c \times h \times w$ tensor, where c is the number of color channels, h is the height and w is the width.

```
data.train[0][0].shape
```

```
torch.Size([1, 32, 32])
```

The categories of Fashion-MNIST have human-understandable names. The following convenience method converts between numeric labels and their names.

4.2.2. Reading a Minibatch

To make our life easier when reading from the training and test sets, we use the built-in data iterator rather than creating one from scratch. Recall that at each iteration, a data iterator reads a minibatch of data with size **batch_size**. We also randomly shuffle the examples for the training data iterator.

```
@d2l.add_to_class(FashionMNIST)
def get_dataloader(self, train):
    data = self.train if train else self.val
    return torch.utils.data.DataLoader(data, self.batch_size, shuffle=train,
                                       num_workers=self.num_workers)
```

To see how this works, let's load a minibatch of images by invoking the **train_dataloade**r method. It contains 64 images.

```
X, y = next(iter(data.train_dataloader()))
print(X.shape, X.dtype, y.shape, y.dtype)
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes
  warnings.warn(_create_warning_msg(
torch.Size([64, 1, 32, 32]) torch.float32 torch.Size([64]) torch.int64
```

Let's look at the time it takes to read the images. Even though it is a built-in loader, it is not blazingly fast. Nonetheless, this is sufficient since processing images with a deep network takes quite a bit longer. Hence it is good enough that training a network will not be I/O constrained.

```
tic = time.time()
for X, y in data.train_dataloader():
    continue
f'{time.time() - tic:.2f} sec'
```

```
'12.37 sec'
```

4.2.3. Visualization

We will often be using the Fashion-MNIST dataset. A convenience function `show_images` can be used to visualize the images and the associated labels. Skipping implementation details, we just show the interface below: we only need to know how to invoke `d2l.show_images` rather than how it works for such utility functions.

```
def show_images(imgs, num_rows, num_cols, titles=None, scale=1.5):  #
    """Plot a list of images."""
    raise NotImplementedError
```

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

Let's put it to good use. In general, it is a good idea to visualize and inspect data that you are training on. Humans are very good at spotting oddities and because of that, visualization serves as an additional safeguard against mistakes and errors in the design of experiments. Here are the images and their corresponding labels (in text) for the first few examples in the training dataset.

```
@d2l.add_to_class(FashionMNIST)
def visualize(self, batch, nrows=1, ncols=8, labels=[]):
    X, y = batch
    if not labels:
        labels = self.text_labels(y)
    d2l.show_images(X.squeeze(1), nrows, ncols, titles=labels)
batch = next(iter(data.val_dataloader()))
data.visualize(batch)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-10-d872326f59fa> in <cell line: 8>()
      6     d2l.show_images(X.squeeze(1), nrows, ncols, titles=labels)
      7 batch = next(iter(data.val_dataloader()))
----> 8 data.visualize(batch)

<ipython-input-10-d872326f59fa> in visualize(self, batch, nrows, ncols, labels)
      3     X, y = batch
      4     if not labels:
----> 5         labels = self.text_labels(y)
      6     d2l.show_images(X.squeeze(1), nrows, ncols, titles=labels)
      7 batch = next(iter(data.val_dataloader()))

AttributeError: 'FashionMNIST' object has no attribute 'text_labels'
```

4.2.4. Summary

We now have a slightly more realistic dataset to use for classification. Fashion-MNIST is an apparel classification dataset consisting of images representing 10 categories. We will use this dataset in subsequent sections and chapters to evaluate various network designs, from a simple linear model to advanced residual networks. As we commonly do with images, we read them as a tensor of shape (batch size, number of channels, height, width). For now, we only have one channel as the images are grayscale (the visualization above uses a false color palette for improved visibility).

Lastly, data iterators are a key component for efficient performance. For instance, we might use GPUs for efficient image decompression, video transcoding, or other preprocessing. Whenever possible, you should rely on well-implemented data iterators that exploit high-performance computing to avoid slowing down your training loop.

Start coding or generate with AI.