Load image data

Image datasets have Image type columns, which contain PIL objects.

> [!TIP]
> To work with image datasets, you need to have the `vision` dependency installed. Check out the installation guide to learn how to install it.

When you load an image dataset and call the image column, the images are decoded as PIL Images:

```
>>> from datasets import load_dataset, Image

>>> dataset = load_dataset("beans", split="train")
>>> dataset[0]["image"]
```

> [!WARNING]
> Index into an image dataset using the row index first and then the `image` column - `dataset[0]["image"]` - to avoid decoding and resampling all the image objects in the dataset. Otherwise, this can be a slow and time-consuming process if you have a large dataset.

For a guide on how to load any type of dataset, take a look at the general loading guide.

Local files

You can load a dataset from the image path. Use the cast_column() function to accept a column of image file paths, and decode it into a PIL image with the Image feature:

```
>>> from datasets import Dataset, Image

>>> dataset = Dataset.from_dict({"image": ["path/to/image_1", "path/to/image_2", ..., "path/to/ima
>>> dataset[0]["image"]
<PIL.PngImagePlugin.PngImageFile image mode=RGBA size=1200x215 at 0x15E6D7160>]
```

If you only want to load the underlying path to the image dataset without decoding the image object, set `decode=False` in the Image feature:

```
>>> dataset = load_dataset("beans", split="train").cast_column("image", Image(decode=False))
>>> dataset[0]["image"]
{'bytes': None,
 'path': '/root/.cache/huggingface/datasets/downloads/extracted/b0a21163f78769a2cf11f58dfc767fb458
```

ImageFolder

You can also load a dataset with an `ImageFolder` dataset builder which does not require writing a custom dataloader. This makes `ImageFolder` ideal for quickly creating and loading image datasets with several thousand images for different vision tasks. Your image dataset structure should look like this:

```
folder/train/dog/golden_retriever.png
folder/train/dog/german_shepherd.png
folder/train/dog/chihuahua.png

folder/train/cat/maine_coon.png
folder/train/cat/bengal.png
folder/train/cat/birman.png
```

Alternatively it should have metadata, for example:

```
folder/train/metadata.csv
folder/train/0001.png
folder/train/0002.png
folder/train/0003.png
```

If the dataset follows the `ImageFolder` structure, then you can load it directly with load_dataset():

```
>>> from datasets import load_dataset

>>> dataset = load_dataset("username/dataset_name")
>>> # OR locally:
>>> dataset = load_dataset("/path/to/folder")
```

For local datasets, this is equivalent to passing `imagefolder` manually in load_dataset() and the directory in `data_dir`:

```
>>> dataset = load_dataset("imagefolder", data_dir="/path/to/folder")
```

Then you can access the videos as `PIL.Image` objects:

```
>>> dataset["train"][0]
{"image": <PIL.PngImagePlugin.PngImageFile image mode=RGBA size=1200x215 at 0x15E6D7160>, "label":

>>> dataset["train"][-1]
{"image": <PIL.PngImagePlugin.PngImageFile image mode=RGBA size=1200x215 at 0x15E8DAD30>, "label":
```

To ignore the information in the metadata file, set `drop_metadata=True` in load_dataset():

```
>>> from datasets import load_dataset

>>> dataset = load_dataset("username/dataset_with_metadata", drop_metadata=True)
```

If you don't have a metadata file, `ImageFolder` automatically infers the label name from the directory name.
If you want to drop automatically created labels, set `drop_labels=True`.
In this case, your dataset will only contain an image column:

```
>>> from datasets import load_dataset

>>> dataset = load_dataset("username/dataset_without_metadata", drop_labels=True)
```

Finally the `filters` argument lets you load only a subset of the dataset, based on a condition on the label or the metadata. This is especially useful if the metadata is in Parquet format, since this format enables fast filtering. It is also recommended to use this argument with `streaming=True`, because by default the dataset is fully downloaded before filtering.

```
>>> filters = [("label", "=", 0)]
>>> dataset = load_dataset("username/dataset_name", streaming=True, filters=filters)
```

> [!TIP]
> For more information about creating your own `ImageFolder` dataset, take a look at the Create an image dataset guide.

WebDataset

The WebDataset format is based on a folder of TAR archives and is suitable for big image datasets.
Because of their size, WebDatasets are generally loaded in streaming mode (using `streaming=True`).

You can load a WebDataset like this:

```
>>> from datasets import load_dataset

>>> dataset = load_dataset("webdataset", data_dir="/path/to/folder", streaming=True)
```

Image decoding

By default, images are decoded sequentially as `PIL.Images` when you iterate on a dataset. However it is possible to speed up the dataset significantly using multithreaded decoding:

```
>>> import os
>>> num_threads = num_threads = min(32, (os.cpu_count() or 1) + 4)
>>> dataset = dataset.decode(num_threads=num_threads)
>>> for example in dataset:  # up to 20 times faster !
...     ...
```

You can enable multithreading using `num_threads`. This is especially useful to speed up

remote data streaming.
However it can be slower than `num_threads=0` for local data on fast disks.

If you are not interested in the images decoded as `PIL.Images` and would like to access the path/bytes instead, you can disable decoding:

```
>>> dataset = dataset.decode(False)
```

Note: IterableDataset.decode() is only available for streaming datasets at the moment.