Load pdf data

> [!WARNING]
> Pdf support is experimental and is subject to change.

Pdf datasets have Pdf type columns, which contain `pdfplumber` objects.

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

When you load a pdf dataset and call the pdf column, the pdfs are decoded as `pdfplumber` Pdfs:

```
>>> from datasets import load_dataset, Pdf

>>> dataset = load_dataset("path/to/pdf/folder", split="train")
>>> dataset[0]["pdf"]
<pdfplumber.pdf.PDF at 0x1075bc320>
```

> [!WARNING]
> Index into a pdf dataset using the row index first and then the `pdf` column - `dataset[0]["pdf"]` - to avoid creating all the pdf 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.

Read pages

Access pages directly from a pdf using the `.pages` attribute.

Then you can use the `pdfplumber` functions to read texts, tables and images, e.g.:

```
>>> pdf = dataset[0]["pdf"]
>>> first_page = pdf.pages[0]
>>> first_page
<Page:1>
>>> first_page.extract_text()
Docling Technical Report
Version1.0
ChristophAuer MaksymLysak AhmedNassar MicheleDolfi NikolaosLivathinos
PanosVagenas CesarBerrospiRamis MatteoOmenetti FabianLindlbauer
KasperDinkla LokeshMishra YusikKim ShubhamGupta RafaelTeixeiradeLima
ValeryWeber LucasMorin IngmarMeijer ViktorKuropiatnyk PeterW.J.Staar
AI4KGroup,IBMResearch
Ru¨schlikon,Switzerland
Abstract
This technical report introduces Docling, an easy to use, self-contained, MIT-
licensed open-source package for PDF document conversion.
...
>>> first_page.images
In [24]: first_page.images
Out[24]:
[{'x0': 256.5,
  'y0': 621.0,
  'x1': 355.49519999999995,
  'y1': 719.9952,
  'width': 98.99519999999995,
  'height': 98.99519999999995,
  'name': 'Im1',
  'stream': <PDFStream(44): raw=88980, {'Type': /'XObject', 'Subtype': /'Image', 'BitsPerComponent
  'srcsize': (1024, 1024),
  'imagemask': None,
  'bits': 8,
  'colorspace': [/'DeviceRGB'],
  'mcid': None,
  'tag': None,
  'object_type': 'image',
  'page_number': 1,
  'top': 72.00480000000005,
  'bottom': 171.0,
  'doctop': 72.00480000000005}]
```

```
>>> first_page.extract_tables()
[]
```

You can also load each page as a `PIL.Image`:

```
>>> import PIL.Image
>>> import io
>>> first_page.to_image()
<pdfplumber.display.PageImage at 0x107d68dd0>
>>> buffer = io.BytesIO()
>>> first_page.to_image().save(buffer)
>>> img = PIL.Image.open(buffer)
>>> img
<PIL.PngImagePlugin.PngImageFile image mode=P size=612x792>
```

Note that you can pass `resolution=` to `.to_image()` to render the image in higher resolution that the default (72 ppi).

Local files

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

```
>>> from datasets import Dataset, Pdf

>>> dataset = Dataset.from_dict({"pdf": ["path/to/pdf_1", "path/to/pdf_2", ..., "path/to/pdf_n"]})
>>> dataset[0]["pdf"]
<pdfplumber.pdf.PDF at 0x1657d0280>
```

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

```
>>> dataset = dataset.cast_column("pdf", Pdf(decode=False))
>>> dataset[0]["pdf"]
{'bytes': None,
 'path': 'path/to/pdf/folder/pdf0.pdf'}
```

PdfFolder

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

```
folder/train/resume/0001.pdf
folder/train/resume/0002.pdf
folder/train/resume/0003.pdf

folder/train/invoice/0001.pdf
folder/train/invoice/0002.pdf
folder/train/invoice/0003.pdf
```

If the dataset follows the `PdfFolder` 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 `pdffolder` manually in load_dataset() and the directory in `data_dir`:

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

Then you can access the pdfs as `pdfplumber.pdf.PDF` objects:

```
>>> dataset["train"][0]
{"pdf": <pdfplumber.pdf.PDF at 0x161715e50>, "label": 0}

>>> dataset["train"][-1]
{"pdf": <pdfplumber.pdf.PDF at 0x16170bd90>, "label": 1}
```

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, `PdfFolder` 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 a pdf 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 `PdfFolder` dataset, take a look at the Create a pdf dataset guide.

Pdf decoding

By default, pdfs are decoded sequentially as pdfplumber `PDFs` when you iterate on a dataset. It sequentially decodes the metadata of the pdfs, and doesn't read the pdf pages until you access them.

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 documents decoded as pdfplumber `PDFs` 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.