mindee / doctr Public

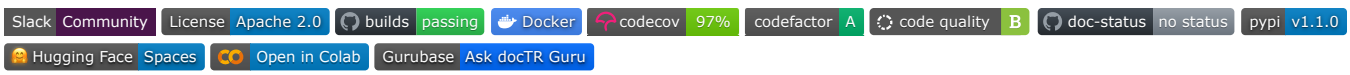
Code | Issues 18 | Pull requests 4 | Discussions | Actions

main | 9 Branches | 22 Tags | Go to file | Go to file | Code

felixdittrich92 [datasets] Correct vocabs (#2139) c7f7218 · 3 days ago

.github	build(deps): bump JamesIves/github-pag...	last month
api	[misc] post release v1.1.0 (#2130)	last month
demo	[TSR] table predictor integration (#2096)	3 months ago
docs	[datasets] Correct vocabs (#2139)	3 days ago
doctr	[datasets] Correct vocabs (#2139)	3 days ago
notebooks	[demo/docs] Update notebook docs & mi...	2 years ago
references	[Fix] DDP init with device id (#2136)	2 weeks ago
scripts	[Feat] Add layout scripts and update aug...	4 months ago
tests	[datasets] fall back to system fonts for ch...	2 weeks ago
.codacy.yml	Improve synthesize_page and font fallbac...	2 months ago
.gitignore	[reconstitution] Improve font scaling and r...	last month
.pre-commit-config.yaml	[datasets] Correct vocabs (#2139)	3 days ago
CODE_OF_CONDUCT.md	docs: Added Code of Conduct (#325)	5 years ago
CONTRIBUTING.md	[Docs] Fix typos (#1885)	last year
Dockerfile	[build] Drop py3.10 support (#2097)	3 months ago
LICENSE	Update LICENSE	4 years ago
Makefile	[CI/CD] Refactor jobs & improve caching (...	2 months ago
README.md	Update README	last month
pyproject.toml	Update root logging (#2129)	last month
setup.py	[misc] post release v1.1.0 (#2130)	last month

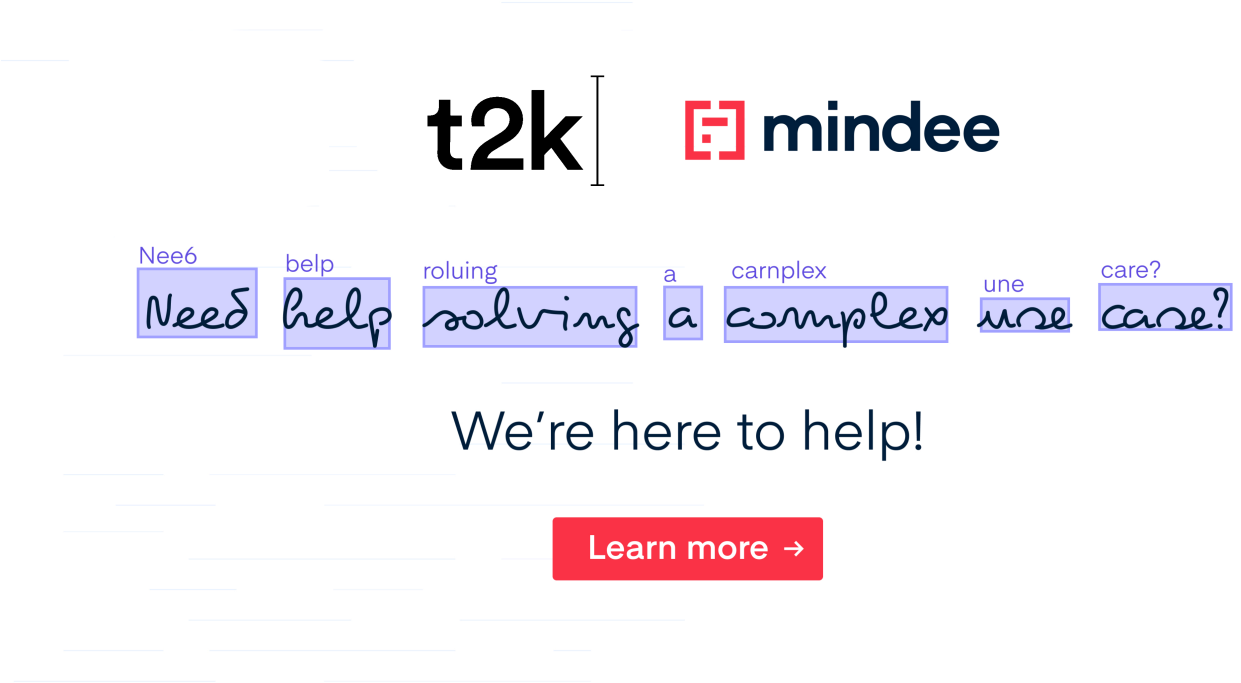
README | Code of conduct | Contributing | Apache-2.0 license



Optical Character Recognition made seamless & accessible to anyone, powered by PyTorch

Project responsibility

docTR was originally created by Mindee. It is now actively developed and maintained by t2k GmbH.



What you can expect from this repository:

- efficient ways to parse textual information (localize and identify each word) from your documents
- guidance on how to integrate this in your current architecture

Quick Tour

Getting your pretrained model

End-to-End OCR is achieved in docTR using a two-stage approach: text detection (localizing words), then text recognition (identify all characters in the word). As such, you can select the architecture used for text detection, and the one for text recognition from the list of available implementations.

```
from doctr.models import ocr_predictor

model = ocr_predictor(det_arch="db_resnet50", reco_arch="crnn_vgg16_bn", pretrained=True)
```

Reading files

Documents can be interpreted from PDF or images:

```
from doctr.io import DocumentFile

# PDF
pdf_doc = DocumentFile.from_pdf("path/to/your/doc.pdf")
# Image
single_img_doc = DocumentFile.from_images("path/to/your/img.jpg")
# Webpage (requires `weasyprint` to be installed)
webpage_doc = DocumentFile.from_url("https://www.yoursite.com")
# Multiple page images
multi_img_doc = DocumentFile.from_images(["path/to/page1.jpg", "path/to/page2.jpg"])
```

Putting it together

Let's use the default pretrained model for an example:

```
from doctr.io import DocumentFile
from doctr.models import ocr_predictor

model = ocr_predictor(pretrained=True)
# PDF
doc = DocumentFile.from_pdf("path/to/your/doc.pdf")
# Analyze
result = model(doc)
```

Detecting the document layout

You can additionally run a layout detection model as part of the pipeline by passing `detect_layout=True`. The detected regions (e.g. `Title`, `Text`, `Table`, `Page-header`, `Page-footer`) are attached to every page and rendered by `.show()`:

```
from doctr.io import DocumentFile
from doctr.models import ocr_predictor

model = ocr_predictor(pretrained=True, detect_layout=True)
doc = DocumentFile.from_images("path/to/your/doc.jpg")
result = model(doc)

# Access the detected layout regions of the first page
for region in result.pages[0].layout:
    print(region.type, region.confidence, region.geometry)
```

Dealing with rotated documents

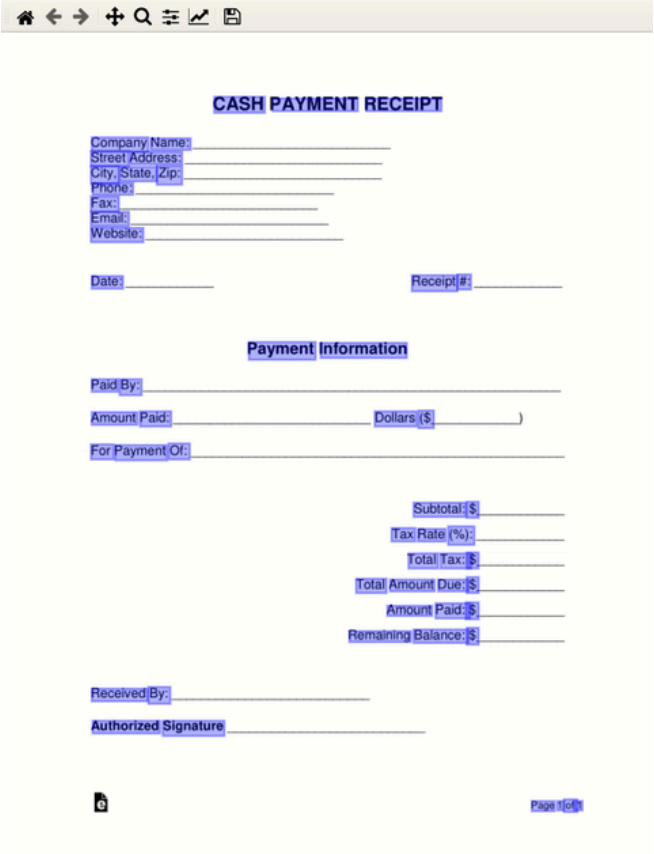
Should you use docTR on documents that include rotated pages, or pages with multiple box orientations, you have multiple options to handle it:

- If you only use straight document pages with straight words (horizontal, same reading direction), consider passing `assume_straight_pages=True` to the ocr_predictor. It will directly fit straight boxes on your page and return straight boxes, which makes it the fastest option.
- If you want the predictor to output straight boxes (no matter the orientation of your pages, the final localizations will be converted to straight boxes), you need to pass `export_as_straight_boxes=True` in the predictor. Otherwise, if `assume_straight_pages=False`, it will return rotated bounding boxes (potentially with an angle of 0°).

If both options are set to False, the predictor will always fit and return rotated boxes.

To interpret your model's predictions, you can visualize them interactively as follows:

```
# Display the result (requires matplotlib & mplcursors to be installed)
result.show()
```



Or even rebuild the original document from its predictions:

```
import matplotlib.pyplot as plt

synthetic_pages = result.synthesize()
plt.imshow(synthetic_pages[0])
plt.axis("off")
plt.show()
```

The `ocr_predictor` returns a `Document` object with a nested structure (with `Page`, `Block`, `Line`, `Word`, `Artefact`). To get a better understanding of our document model, check our documentation:

You can also export them as a nested dict, more appropriate for JSON format:

```
json_output = result.export()
```

Use the KIE predictor

The KIE predictor is a more flexible predictor compared to OCR as your detection model can detect multiple classes in a document. For example, you can have a detection model to detect just dates and addresses in a document.

The KIE predictor makes it possible to use detector with multiple classes with a recognition model and to have the whole pipeline already setup for you.

```
from doctr.io import DocumentFile
from doctr.models import kie_predictor

# Model
model = kie_predictor(det_arch="db_resnet50", reco_arch="crnn_vgg16_bn", pretrained=True)
# PDF
doc = DocumentFile.from_pdf("path/to/your/doc.pdf")
# Analyze
result = model(doc)

predictions = result.pages[0].predictions
for class_name in predictions.keys():
```

```
    list_predictions = predictions[class_name]
    for prediction in list_predictions:
        print(f"Prediction for {class_name}: {prediction}")
```

The KIE predictor results per page are in a dictionary format with each key representing a class name and it's value are the predictions for that class.

Installation

Prerequisites

Python 3.11 (or higher) and pip are required to install docTR.

Latest release

You can then install the latest release of the package using pypi as follows:

```
pip install python-doctr
```

We try to keep extra dependencies to a minimum. You can install specific builds as follows:

```
# standard build
pip install python-doctr
# optional dependencies for visualization, html, and contrib modules can be installed as follows:
pip install "python-doctr[viz,html,contrib]"
```

Developer mode

Alternatively, you can install it from source, which will require you to install Git. First clone the project repository:

```
git clone https://github.com/mindee/doctr.git
pip install -e doctr/.
```

Again, if you prefer to avoid the risk of missing dependencies, you can install the build:

```
pip install -e doctr/.
```

Models architectures

Credits where it's due: this repository is implementing, among others, architectures from published research papers.

Text Detection

- DBNet: Real-time Scene Text Detection with Differentiable Binarization.
- LinkNet: LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation
- FAST: FAST: Faster Arbitrarily-Shaped Text Detector with Minimalist Kernel Representation

Text Recognition

- CRNN: An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition
- SAR: Show, Attend and Read:A Simple and Strong Baseline for Irregular Text Recognition.
- MASTER: MASTER: Multi-Aspect Non-local Network for Scene Text Recognition.
- ViTSTR: Vision Transformer for Fast and Efficient Scene Text Recognition.
- PARSeq: Scene Text Recognition with Permuted Autoregressive Sequence Models.
- VIPTR: A Vision Permutable Extractor for Fast and Efficient Scene Text Recognition.

Layout Analysis

- LW-DETR: A Transformer Replacement to YOLO for Real-Time Detection.

Table Structure Recognition

- TableCenterNet: A one-stage network for table structure recognition.

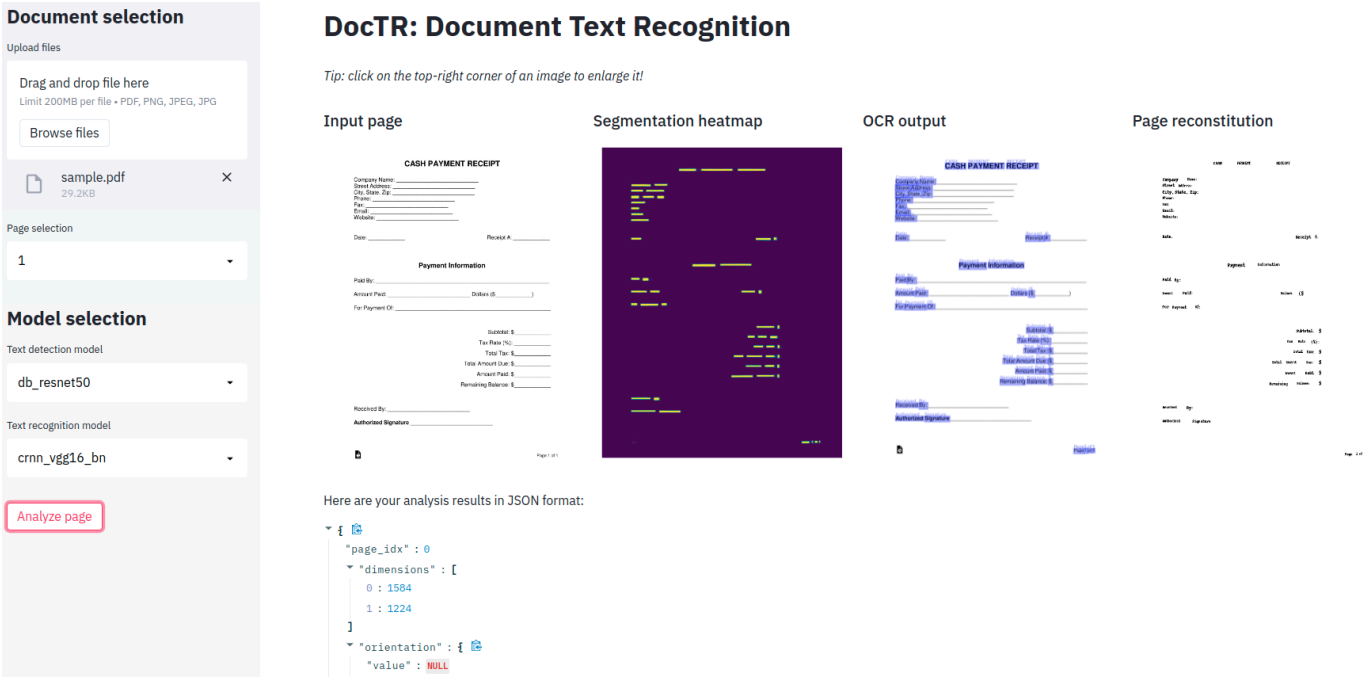
More goodies

Documentation

The full package documentation is available here for detailed specifications.

Demo app

A minimal demo app is provided for you to play with our end-to-end OCR models!



Live demo

Courtesy of 🤗 Hugging Face 🤗, docTR has now a fully deployed version available on Spaces! Check it out 🤗 Hugging Face Spaces

Running it locally

If you prefer to use it locally, there is an extra dependency (Streamlit) that is required.

```
pip install -r demo/pt-requirements.txt
```

Then run your app in your default browser with:

```
streamlit run demo/app.py
```

Docker container

We offer Docker container support for easy testing and deployment. Here are the available docker tags..

Using GPU with docTR Docker Images

The docTR Docker images are GPU-ready and based on CUDA `12.2`. Make sure your host is **at least** `12.2`, otherwise Torch won't be able to initialize the GPU. Please ensure that Docker is configured to use your GPU.

To verify and configure GPU support for Docker, please follow the instructions provided in the NVIDIA Container Toolkit Installation Guide.

Once Docker is configured to use GPUs, you can run docTR Docker containers with GPU support:

```
docker run -it --gpus all ghcr.io/mindee/doctr:torch-py3.9.18-2024-10 bash
```

Available Tags

The Docker images for docTR follow a specific tag nomenclature: `<deps>-py<python_version>-<doctr_version|YYYY-MM>`. Here's a breakdown of the tag structure:

- `<deps>`: `torch`, `torch-viz-html-contrib`.
- `<python_version>`: `3.10.13`, `3.11.8`, etc.
- `<doctr_version>`: a tag >= `v0.11.0`
- `<YYYY-MM>`: e.g. `2014-10`

Here are examples of different image tags:

Tag	Description
`torch-viz-html-contrib-py3.11.8-2024-10`	Torch with extra dependencies version `3.11.8` from latest commit on `main` in `2024-10`.
`torch-py3.11.8-2024-10`	PyTorch version `3.11.8` from latest commit on `main` in `2024-10`.

Building Docker Images Locally

You can also build docTR Docker images locally on your computer.

```
docker build -t doctr .
```

You can specify custom Python versions and docTR versions using build arguments. For example, to build a docTR image with PyTorch, Python version `3.9.10`, and docTR version `v0.7.0`, run the following command:

```
docker build -t doctr --build-arg FRAMEWORK=torch --build-arg PYTHON_VERSION=3.9.10 --build-arg DOCTR_VERSION=v0.7.0
```

Example script

An example script is provided for a simple documentation analysis of a PDF or image file:

```
python scripts/analyze.py path/to/your/doc.pdf
```

All script arguments can be checked using `python scripts/analyze.py --help`

Minimal API integration

Looking to integrate docTR into your API? Here is a template to get you started with a fully working API using the wonderful FastAPI framework.

Deploy your API locally

Specific dependencies are required to run the API template, which you can install as follows:

```
cd api/
pip install poetry
make lock
pip install -r requirements.txt
```

You can now run your API locally:

```
uvicorn --reload --workers 1 --host 0.0.0.0 --port=8002 --app-dir api/ app.main:app
```

Alternatively, you can run the same server on a docker container if you prefer using:

```
PORT=8002 docker-compose up -d --build
```

What you have deployed

Your API should now be running locally on your port 8002. Access your automatically-built documentation at http://localhost:8002/redoc and enjoy your three functional routes ("/detection", "/recognition", "/ocr", "/kie"). Here is an example with Python to send a request to the OCR route:

```python
import requests

params = {"det_arch": "db_resnet50", "reco_arch": "crnn_vgg16_bn"}

with open("/path/to/your/doc.jpg", "rb") as f:
    files = [  # application/pdf, image/jpeg, image/png supported
        ("files", ("doc.jpg", f.read(), "image/jpeg")),
    ]
print(requests.post("http://localhost:8080/ocr", params=params, files=files).json())
```

Example notebooks

Looking for more illustrations of docTR features? You might want to check the Jupyter notebooks designed to give you a broader overview.

Citation

If you wish to cite this project, feel free to use this BibTeX reference:

```
@misc{doctr2021,
    title={docTR: Document Text Recognition},
    author={Mindee},
    year={2021},
    publisher = {GitHub},
    howpublished = {\url{https://github.com/mindee/doctr}}
}
```

Releases 22

v1.1.0 Latest
last month

+ 21 releases

Used by 1.3K

Contributors 78

+ 64 contributors

Languages

Python 99.8% **Other** 0.2%