Jekyll theme for documentation — mydoc product

version 6.0

Last generated: June 07, 2021

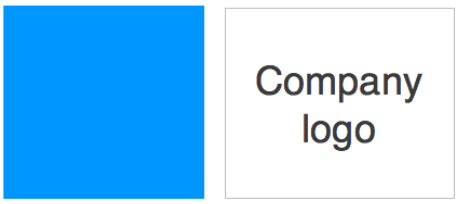


Table of Contents

Introduction

Summary: Description of the documentation web page.

Overview

This site provides all documentation that is needed for setting up, configuring and running the DocVisor tool. The tool contains layouts for common document analysis tasks: OCR, Weakly Supervised Region Segmentation and Fully Automatic Region Segmentation. The tool is designed such that the user only needs to provide their data and set up the configuration files for loading the tool.

The instructions provided in the document should be sufficient to help you set up the tool. In case you face any issues, you can feel free to contact any of the Contributors (page 0).

Page-Layouts

OCR

The OCR task involves recognizing text from images. Our tool is particularly designed to interact with and visualize the outputs of models which are trained using the attention mechanism.

In particular, the OCR layout is meant to visualize the results of OCR models trained for **line images** using the attention mechanism. While the layout is best experienced if the user provides the data with attentions, the layouts can be loaded even with simple images and prediction/ground truth data wand does not require the user to have attentions for it to load.

In compliance with the design of the whole tool, the OCR layout is extremely easy to use, and has loads of features that help the user focus on their analysis of the trained models.

There are two main features of the OCR layout:

Text Selection:

Text Selection is a feature that allows the user to select any substring of the predicted text and see its corresponding portion highlighted in the image.

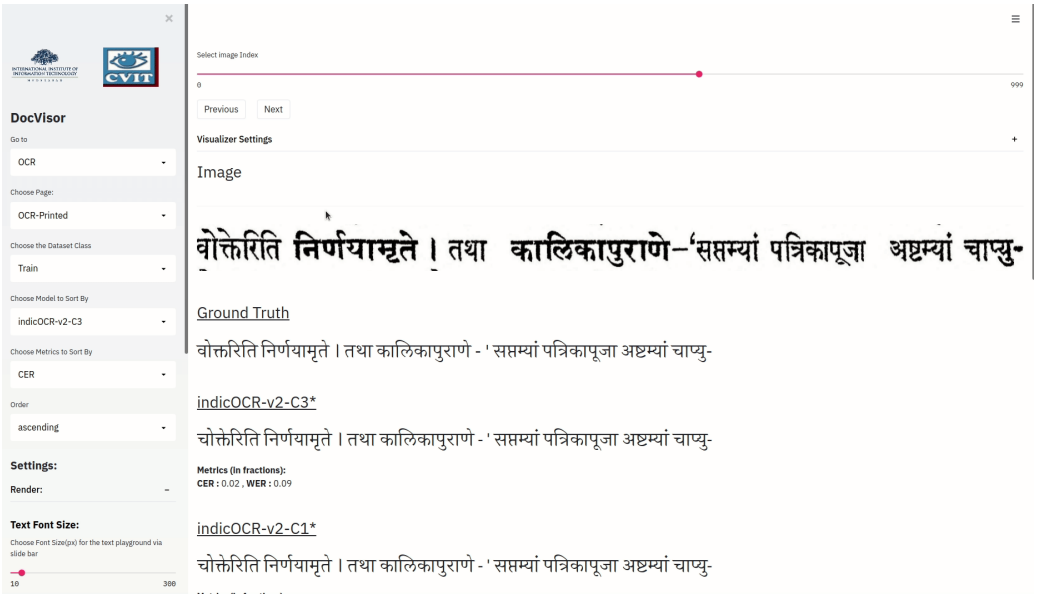
The following gif should give a good idea of the feature.

Note: This feature requires that the attention matrix is provided as an input by the user. For more details check the OCR layout page.

Image Selection:

The Image Selection is a feature that allows the user to select any portion of the line image and have the corresponding predicted text highlighted.

The following gif should give a good idea of the Image Selection Feature:



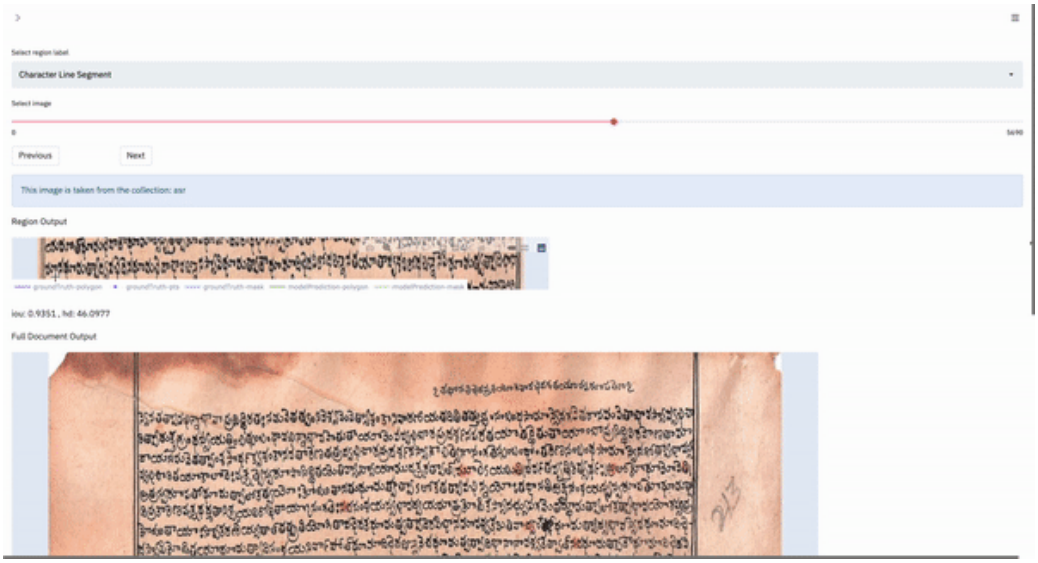
The layout is designed to be highly customizable. All the relevant settings can be adjusted via the setting panel in the left side panel of the OCR layout. For more details about getting started with the OCR tool, visit **add reference**.

Fully-Automatic Layout:

Those region segmentation approaches which do not have supervision during inference come under the family of fully-automatic region segmentation. Here, the model identifies the various regions, along with their corresponding labels.

DocVisor provides the utility of output visualization of the model by being able to simultaneously view the ground truth and model output for every region, as well as the entire document. Additionally, the user can also visualize outputs on a full-document basis.

The following gif shows the output selection and visualization process:



The user can click once on the output to be visualized, and it is then overlayed on the existing plot. This way, visualized outputs can be stacked on another for a better visual comparison. The user can also double-click on a single output to isolate it, i.e. only that output is visualized.

In the full-document output, the current region being shown above is highlighted in red, so that the user knows where the reigon is with respect to the document as a whole.

Box-Supervised Layout:

This layout is primarily useful for those tasks which involve weakly-supervised region segmentation with the help of bounding boxes. Specifically, the tool enables the user to view the various possible outputs within the pipeline of the model on a per-region basis.

For a single region, the app renders a plot with only the current region of interest shown. The user can select the outputs they wish to visualize and so the different outputs can be compared.

The following gif shows the output selection and visualization process:

We can see that the visualization process is the same as in the fully-automatic layout.

Getting Started

Summary: All necessary instructions to use the DocVisor tool.

Step 1 : Download or Clone DocVisor tool

Clone Using Git

If you have git installed on your local machine, run the following command to clone the docvisor repository.

```
git clone https://github.com/ihdia/docvisor
```

Download Zip

If you do not have git or you want to download the zip file, download the zip file from here and unzip the tool to any location on your divice.

Step 2: Data Preperation

There three main layouts of the DocVisor tool:

1. Fully Automatic
2. Box Supervised

3. OCR

You can load one or more of these tools to the DocVisor tool at any given point in time.

- To load the Fully Automatic tool, prepare your datafiles as described here (page 0)
- To load the Box Supervised tool, prepare your datafiles as described here (page 0)
- To load the OCR tool, prepare your datafiles as described here (page 0)

Step 3: Setting up your environment

1. Create a conda environment using the following command:

```
conda create --name docvisor
```

2. Ensure that the pip points to the docvisor environment by running `which pip`. If it does not, then run the following command:

```
conda install pip
```

3. Install the requirements necessary:

```
pip install -r requirements.txt
```

Step 4: Modify Config File

1. Place all the metaData files in one directory

The metaData directory will look like:

```
```
metaData/
    - ocr_handwritten.json
    - ocr_printed.json
    - fullyAutomatic.json
    - boxSupervised.json

```
```

1. Change the path of the metaData file in the docvisor/config.py file.

Step 5: Launch the tool

Launch the tool by running `./run.sh` script.

Load Example Data

Paper

Summary: About our Submission to ICDAR-OST 2021

We have submitted the DocVisor tool to ICDAR on June 7th 2021.

We will upload the link to the paper soon ....

WATCH THIS SPACE!

Setting Up OCR Layout

Summary: This page carries all information required to load the OCR-layout of the DocVisor tool.

Note: Installation of the DocVisor tool is a pre-requisite to be successful in loading the OCR-Layout. If you have not installed the tool, kindly install the tool following the instructions provided here.

To get the tool up and running, the tool requires some metadata files to be ready. The following section should help you set up the metadata files.

Step 1 : Configuring the OCR layout

Step 1.1 Generation of Meta Data File

Meta Data file is a configuration file that tells the OCR Layout of the DocVisor tool the exact location of the datafiles along with other important details. It is in `json` format. The following is an example of what a typical metadata file for the ocr-layout looks like.

```
{
    "metaData": {
        "pageLayout": "OCR",
        "pageName": "Name of your OCR Page Layout",
        "dataPaths": {
            "Class 1": "/path/to/class_1.json",
            "Class 2": "/path/to/class_2.json",
                .               .
                .               .
                .               .
            "Class N":"/path/to/class_n.json"
        }
    }
}
```

1. `pageLayout`: **string**

   The value of `pageLayout` for this page is “OCR”. This tells the DocVisor tool to use all the other `metaData` information and initialize the **OCR** layout of the DocVisor tool. The value of `pageLayout` key can be one of the following:

   a. OCR

   b. Fully Automatic Region Parsing

   c. Box Supervised

   The instructions in this page pertain to setting up and loading OCR(1). Checkout this link (page 0) for 2 and **this link** for 3.

2. `pageName`: **string**

   `pageName` : Any name of your interest. If you are having multiple OCR-layouts to be loaded, then the `pageName` variable **must** be unique., i.e. no two OCR layout meta data files should have the same `pageName` value.

3. `dataPaths`: **dictionary**

   `dataPaths` is a dictionary having the structure shown above.

   Each key : “class i” is a unique user-defined data class that the user wants to load. A common way of classifying a dataset in most deep learning scenarios is to classify it as *train*, *test* and *validation* sets. The value of each of these “class i” keys is a path to a json data file that has all the ocr data information for that particular class.

Example of Meta Data File:

```
{
    "metaData": {
        "pageLayout": "OCR",
        "pageName": "OCR-Printed",
        "dataPaths": {
            "Train": "/home/user/Desktop/docvisor/train/train_d
ata.json",
            "Validation":"/home/user/Desktop/docvisor/val/val_d
ata.json",
            "Test": "/home/user/Desktop/docvisor/test/test_dat
a.json"
        }
    }
}
```

Step 1.2 Setup Directory

Create a directory containing only config files. Each instance of the layout should have a json file. For example, if we are trying to visualize the outputs of a line OCR for both typed and hand-written images, the following is the directory structure:

```
metadata/
    - ocr_handwritten.json
    - ocr_printed.json
```

With each of the json files following the format as described above. Note that no two instances of the layout should have the same `pageName` key.

Multiple Layouts

If you are trying to visualize results of either Fully Automatic or Box Supervised Layouts, you can do so, by creating similar metaData files for them as described in their corresponding documentations and then place all of them into this metaData directory.

Example of directory containing all three layouts:

```
metaData/
    - ocr_handwritten.json
    - fullyAutomatic.json
    - boxSupervised.json
```

The docVisor tool will launch single instances of the OCR, Fully Automatic and Box Supervised tools.

Multiple Instances of the Same Layout

The DocVisor tool allows the user to view multiple instances of the same layout in a single session. For example, if you have a handwritten dataset and printed dataset and you are wanting to analyze the OCR

outputs of both of these datasets, you can just create mulitple a seperate json file with unique `pageName` key and place the file in the metaData directory.

The metaData directory will look like:

```
metaData/
    - ocr_handwritten.json
    - ocr_printed.json
    - fullyAutomatic.json
    - boxSupervised.json
```

For the above directory structure, the DocVisor tool will load two instances of the OCR layout and a single layout of fully automatic and box supervised layouts.

Step 1.3 Updating the Config File

In docVisor/config.py file, change the `metaDataDir` to point to the metaData Directory that you have created.

Step 2: Prepare your Data

The data file required to launch your OCR layout is a json file. At any given point in time you can visualize a single image and its corresponding ground truth and

(multi)-model predictions. Hence the json data provided will be an array of images with each object of the array having the following format:

```
{
        "id":"<some-unique-image-id>",  # MANDATORY
        "imagePath":"/path/to/line/image/1.jpg", # MANDATORY
        "groundTruth":"annotated-ground-truth-string",
        "outputs":{

            model1,
            model2,
              .   ,
              .   ,
              .   ,
            modeln
        },
        "info":{
            "key1":value1
            "key2":value2
               .     .
               .     .
               .     .
            "keyn":valuen
        }
}
```

1. `id`:
    - **data-type**: `string`
    - **mandatory**: **YES**
    - **Description**: `id` is a Unique string
2. `imagePath` :

- **data-type**: `string`
- **mandatory**: **YES**
- **Description**: `imagePath` is a string that has the path to the line image file.

> ☑ **Tip:** Give absolute paths as the value to imagePaths.

3. `groundTruth` :
    - **data-type**: `string`
    - **mandatory**: **NO**
    - **Description**: `groundTruth` expects a string. It represents the text ground truth of the line image. This is **not mandatory**. You can either load the tool with just images or images and predictions if you do not have the ground truth. If passed on, it will be displayed, else it will be omitted. In case you don’t have ground truth for your dataset, just ommit the `groundTruth` field.

4. `outputs` :
    - **data-type**: `dictionary`
    - **mandatory**: **NO**
    - **Description**: If you have outputs of models (with or without attention), you can specify their details in this dictionay. The `outputs` field is not mandatory. In case you just want to load the OCR layout just to analyze the ground truth (for purpose of correction) or just view the images in your dataset, you can ommit the `outputs` field.

    The `outputs` field is a disctionary of models. Each model has the following structure:

```
 "model_name":{
         "attentions":[], # array of attentions
         "prediction":"predicted string from the model m
odel_name",
         "metrics":{
             "metric 1": value_1              #value_1
: int,
             "metric 2": value_2             #value_2 :
int,
                          .
                          .
                          .
             "metric n": value_n             #value_n :
int

         }
 },
```

`model_name` :

- **data-type**: `dictionary`
- **mandatory**: **Yes of there are models, otherwise No**
- **Description**: Replace the string `model_name` with any unique-name of your interest.

- `attentions`:

- **data-type**: `list/array`
- **mandatory**: **NO**
- **Description**: `attentions` field expects an list of attention numbers. The attentions from

your model, should be pre-processed. If you have attentions, you most probably are it getting it as an intermediate output from a Seq2Seq architecture where the attentions are being used by the decoder. We expect that you pre-process the attention matrix such that, for every character of your predicted string, you save the range of the x co-ordinates that your model is attending to for that character. Note that since we are dealing with line images, we only need the x co-ordinates.

After pre-processing, it is expected that your array be of shape (length_of_your_predicted_sequence,2)After that just ravel that array. i.e. the ith row of your numpy array has the x-co-ordinates of the image (x1,x2) where the model has attended to.

Using the numpy.ravel() command, ravel that array and store and then set it as the value for the `attentions` field.

Many a times, the sequence predicted and the final sequence displayed is different. For example if you are using some kind of endcoding or your predicted sequence is LaTeX code and you want to visualize the actual compiled output.For the indicOCR case, the sequence predicted was a string of concatenations of the ords of each character. So this line was used for creating the character-Image-pixels mapping. If your encoding structure is different or if you do not have any encoding, feel free to modify/delete the call of `ord` function.

For more clarification contack `docvisor.iiith@gmail.com`

`predictions`:

- **data-type**: `string`
- **mandatory**: **NO**
- **Description**: Predicted string for that particular model.

`metrics`:

- **data-type**: `dictionary`
- **mandatory**: **NO**
- **Description**: We expect that there will be metrics if you have both ground truth and predicted text. The metrics is a dictionary with key value pairs. Each key is a string – name of the metric defined by the user, and value is a number – integer or float.

Example of a metrics dictionary:

```
    "metrics":{
        "CER":0.12,
        "WER":0.23
    }
```

Where CER and WER correspond to the Character Error Rate and Word Error Rate. You can define your own metrics and load them.

5. `info`:
    - **data-type**: `dictionary`
    - **mandatory**: **NO**
    - **Description**: Many times, our data has some meta information, such as the origin of the file, or which collection it belongs to or any other information. You may even want to have notes on that particular image. So, we have provied the info field which is not mandatory. But if you want to look at individual image’s meta information, store the information as key value pairs in a dictionary.

    An example of the `info` key for images taken from classical texts of Indian Languages could be:

```
    "info":{
        "collection":"Nirnaya Sindhu"
    }
```

The above info dictionary has a collection key with a value of Nirnaya Sindhu, indicating that the image is taken from the collection Nirnaya Sindhu. You can define your own keys and their corresponding values and any number of them as well. However the values and keys have to be strings. This Information will then be displayed in the bottom of the OCR layout.

The above keys and fields represent data for a single image. To add details of multiple images, store them in the json file as a list.

> ℹ **Note:** Although you can avoid the non-mandatory fields, the tool expects some form of uniformity. i.e. if a particular non-mandatory field is not present/ is present in the data of the first image, then it should not be

present / should be present in all the images of that particular json file.

☑ **Tip:** If you have data which is non-uniform, like say, for some images you have ground truth and for others you don’t, then classify your dataset such that it is uniform and load each of the classes simultaneously in your metaData file as described above (page 14)

Step 3: Launch the Tool

To launch the tool, you need to run the ./run.sh file. The tool will load on localhost with `port 8501`. If `8501` is pre-occupied, check the terminal to know which exact port in which it has been loaded.

Help and Feedback

For Feedback or queries, you can either visit the github repo and create issues or use the discussion format. For more details, you can mail, `docvisor.iiith@gmail.com`.

Text to Image Mapping

Summary: This page will help you understand how to use the Text Selection component of the tool. It will also describe how to use the various settings provided by the OCR-Layout of the DocVisor tool to get a better experience using the text selection component.

Note: Note that you can make use of this component only if you have provided attentions as a part of the data in the json file. For more details of how to load the data, visit this page (page 0)

What is Text to Image Mapping

In the process of learning to map a line image to its corresponding text, in intermediate stage produces the alignment matrix via the attention component. We leverage this attention/ alignment matrix to introduce a cool feature of Text to Image and Image to Text Mapping.

Alignment Matrix

The attentions produced at each time step, can be viewed as a probability distributions indicating the decoder to pay-attention to a certain features of the image. Using this matrix we find the location the model is paying attention at that time step. To see how this is done, visit this page:

Usage

Step 1: In the **Visualize Settings** of the OCR-Layout, select the **Text Selection** Component.

Step 2: Select a any substring of the predicted text of **any of the models that have attention** in them.

Step 3: View the highlighted image.

The following gif should give you an idea of the Text Selection Feature and how cool it is.

☑ **Tip:** To know if a model has attentions or not, the OCR-Layout places a *(star symbol) at the end of the name of the tool.

Highlighting Color

If the user wants to change the color used for highlighting the image, to a color that they like or to the one that is best suited for their image dataset, use the tool's sidebar settings to do so.

1. Open the sidebar (if it is not open)
2. Expand the Render component under **Settings**

3. Go to **highlighted image color** and choose the color of your interest.

The tool will now use this color for highlighting the image.

The following gif should help you understand what exactly needs to be done to use this feature.

Change Image Threshold

Many times the dataset may have images that are very blurr. For highlighting, we have used an image threshold that best suites a black and white image. However, it is possible that users may have images that are in gray scale, and on selection of

a substring of the predicted text, you loose importan character pixels to the highlighted color. To avoid this we have allowed the user to tweak this and set it to a value best for their dataset/purpose.

To do so, follow the below steps:

1. Open the sidebar (if it is not open)
2. Expand the Render component under **Settings**
3. Go to **Image Threshold** and choose the threshold value that bests suits your dataset.

Text Font Size

For purpose of easy-visualization, users might want to aling text along with the characters in the image. To do so, follow the steps below:

To do so, follow the below steps:

1. Open the sidebar (if it is not open)
2. Expand the Render component under **Settings**
3. Go to **Text Font Size** and choose the

font-size value that bests suits the image/dataset.

The following gif should help you understand the feature.

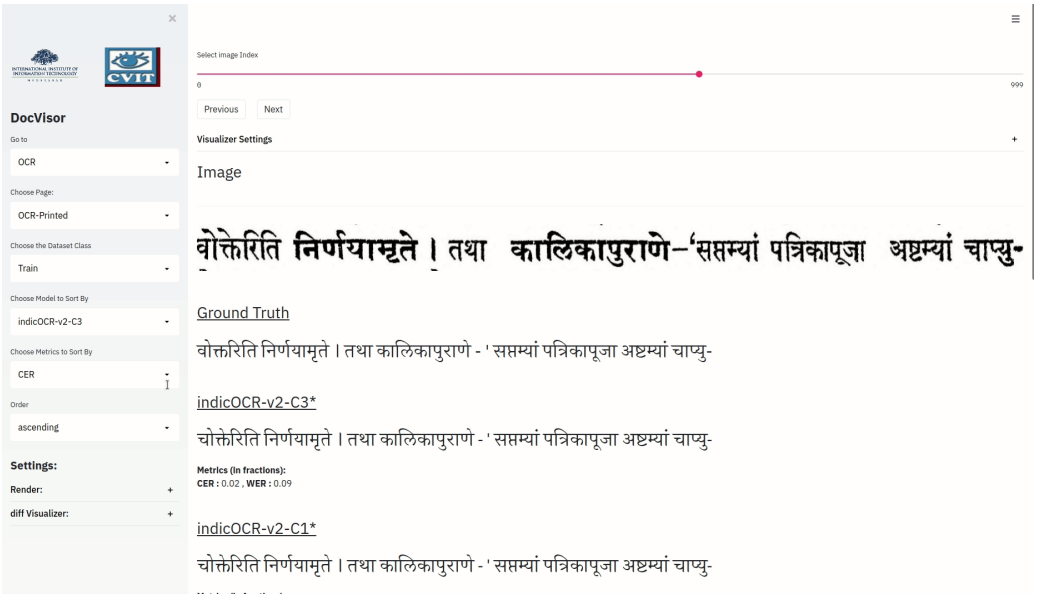


Image to Text Mapping

Summary: This page will help you understand how to use the Image Selection component of the tool. It will also describe how to use the various settings provided by the OCR-Layout of the DocVisor tool to get a better experience using the Image selection component.

Note: Note that you can make use of this component only if you have provided attentions as a part of the data in the json file. For more details of how to load the data, visit this page (page 0)

What is Image to Text Mapping

In the process of learning to map a line image to its corresponding text, in intermediate stage produces the alignment matrix via the attention component. We leverage this attention/ alignment matrix to introduce a cool feature of Image to Text Mapping and Text to Image mapping.

Alignment Matrix

The attentions produced at each time step, can be viewed as a probability distributions indicating the decoder to pay-attention to a certain features of the image. Using this matrix we find the location the model is paying attention at that time step. To see how this is done, visit this page:

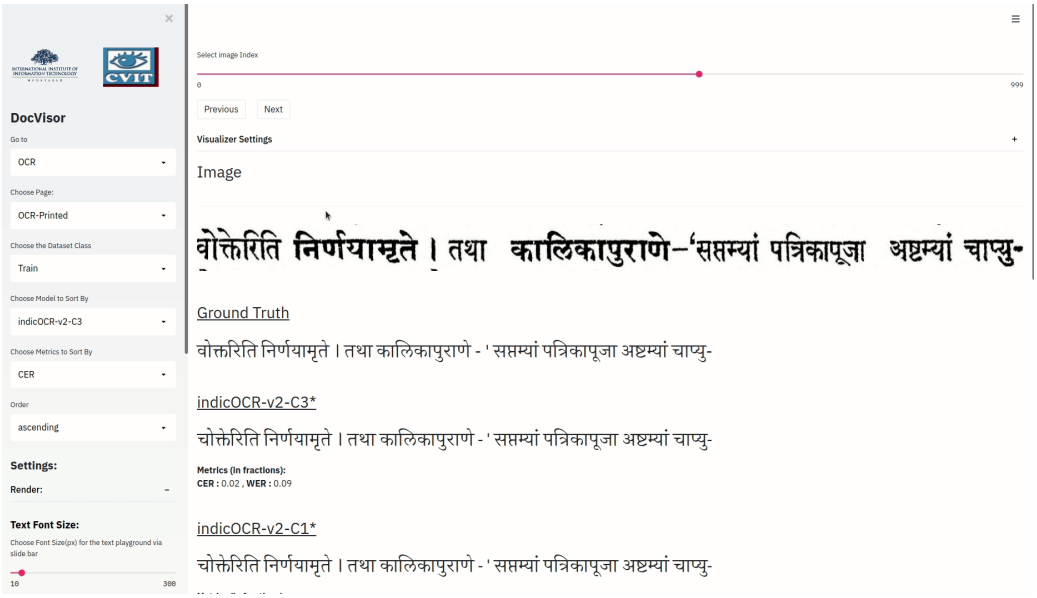
Usage

Step 1: In the **Visualize Settings** of the OCR-Layout, select the **Image Selection** Component.

Step 2: Select a any sub-portion of the image.

Step 3: View the the corresponding text highlighted on all the models that have attentions provided.

The following gif should give you an idea of the Image Selection Feature and how cool it is.



☑ **Tip:** To know if a model has attentions or not, the OCR-Layout places a *(star symbol) at the end of the name of the tool.

🛈 **Note:** Notice that on selection of a range of the image via a crop like feature, the corresponding the substring of all models that have attentions get highlighted. In the above gif, notice that only indicOCR-v2-C3 and incdicOCR-v2-C1 (which are both conactenated with a * at the end) have their corresponding substrings highlighted. The Google-OCR model,

which does not have attentions, will not be able to avail this feature.

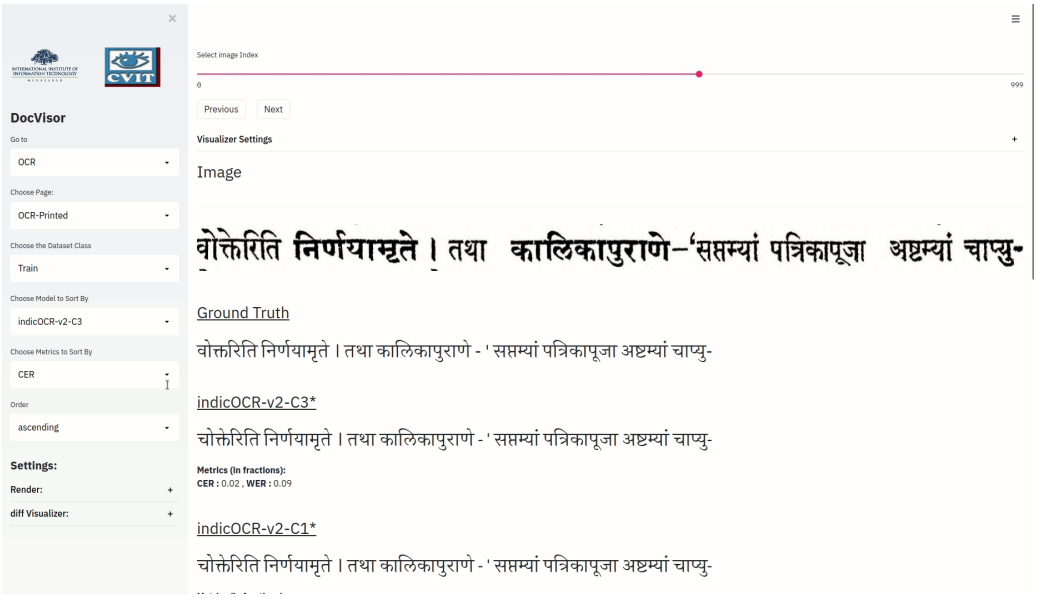
Text Font Size

For purpose of easy-visualization, users might want to aling text along with the characters in the image. To do so, follow the steps below:

To do so, follow the below steps:

1. Open the sidebar (if it is not open)
2. Expand the Render component under **Settings**
3. Go to **Text Font Size** and choose the font-size value that bests suits the image/dataset.

The following gif should help you understand the feature.



Notice that the predicted text words are almost in line with the lines image words, which can help in visualization.

Navigation

Summary: This page should help the user understand all the possible ways, he can navigate through his dataset so that he/she can be analyze the dataset in a focused manner.

Metrics

Note: Note that you can make use of this component only if you have provided metrics as a part of the data in the json file. For more details of how to load the data, visit this page (page 0)

- The user can navigate based on the metrics provided either by ascending or descending order of the mertics.
- To do so, use the side navigation bar to select by which metric and how the metric should be sorted for navigation.
- In case you have multiple model outputs, the user can also select by which model’s metric the sorting should be done.

- In order to view the metrics in the same order as provided in the json data file, choose the metric “None” using the drop down provided for choosing the metrics.

Bookmarks

In order to bookmark an image, for further analysis, click the bookmark button at the bottom of the page. Once you have bookmarked all the images, select bookmark in the sidebar navigation from the **Select Dataset Class** section of the drop down. Your image is already bookmarked if there is a 🔖 symbol next to the Image title.

The below gif should help you understand the bookmarks feature:

Save

To save the particular image index on disk for further analysis, click the save button on the bottom of the image. The index will be saved to disk. To know if your image is already saved, look for the 💾 symbol agains the title IMAGE above your line image.

While navigating through your bookmarks, you can choose to either save a single image onto disk or all bookmarks onto disk at once.

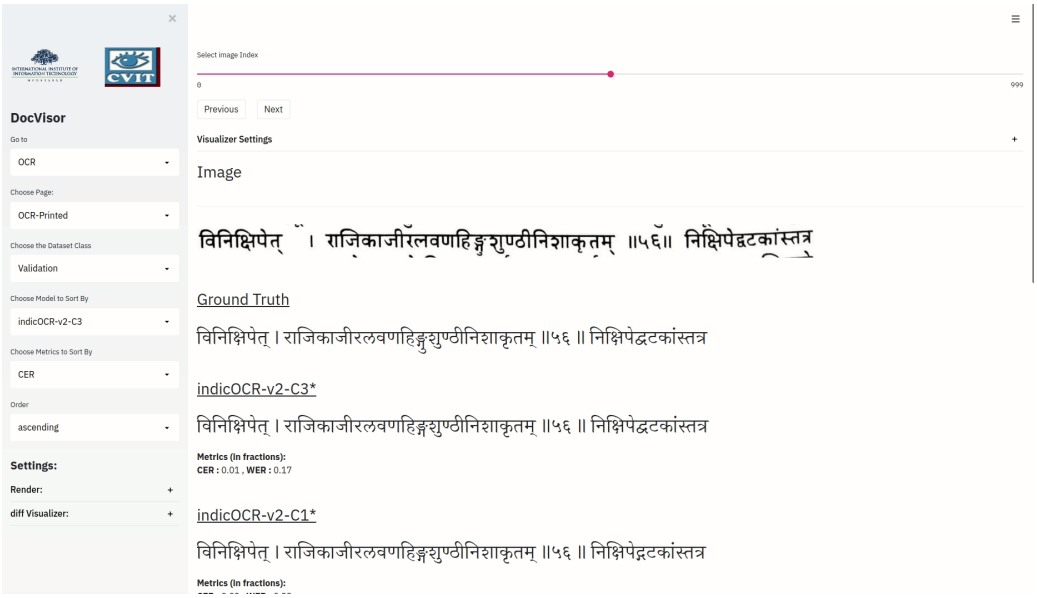
The navigation will save a folder with the following structure:

```
Date/
└── N
    ├── Box-supervised Region Parsing
    │   └── BoundaryNet
    ├── Fully Automatic Region Parsing
    │   ├── DocBank
    │   ├── FullyAutomatic
    │   └── PubTab
    │
    └── OCR
        ├── OCR-Handwritten
        └── OCR-Printed
            └── save.txt
```

N : Nth time you have used the docVisor tool on that particular date.

Info

If you have provided the info key in the json data, you will find a info field at the bottom of the page. Expand it to see all the key,value pairs provided in the info field to the ocr layout printed.


DocVisor
Go to
OCR
Choose Page:
OCR-Printed
Choose the Dataset Class
Validation
Choose Model to Sort By
indicOCR-v2-C3
Choose Metrics to Sort By
CER
Order
ascending
Settings:
Render:
diff Visualizer:
Select image Index
0
999
Previous
Next
Visualizer Settings
Image
Ground Truth
विनिक्षिपेत् । राजिकाजीरलवणहिङ्गुशुण्ठीनिशाकृतम् ॥५६॥ निक्षिपेद्वटकांस्तत्र
indicOCR-v2-C3*
विनिक्षिपेत् । राजिकाजीरलवणहिङ्गुशुण्ठीनिशाकृतम् ॥५६ ॥ निक्षिपेद्वटकांस्तत्र
Metrics (in fractions):
CER : 0.01 , WER : 0.17
indicOCR-v2-C1*
विनिक्षिपेत् । राजिकाजीरलवणहिङ्गुशुण्ठीनिशाकृतम् ॥५६ ॥ निक्षिपेद्वटकांस्तत्र
Metrics (in fractions):

Diff Visualizer

Summary: This page will help you understand the diff Visualizer component of the OCR Layout tool.

Note: Note that you can make use of this component only if you have atleast ground truth and one prediction or you have atleast 2 model output predictions

Ground Truth vs Model

In the OCR setting, our goal is to train our models such that the predicted text is as close to the ground truth. Sometimes, it might be useful to visualize the exact characters that have been wrongly predicted by our models. To do so the OCR tool has provided a diff visualizer.

The gif below should help understand how to use the feature.

Model Truth vs Model

When you have outputs from multiple models, you may want to see the difference of two strings between those two models. The OCR layout provides a mechanism to do so.

Notation

The above visualization, depicts the minimal operations required to convert the target string to the reference string. Notice that every distinct color bounded-text is also annotated with numbers. Also notice that for a color bounded component in the reference string with annotation i, there could be a corresponding color bounded text annotated with the same number i in the target string. (the color of these two color-bounded texts need not be the same).

Color of reference Component i	Color of Target Component i	What does it Mean?
‘<some-reference-text>(i)’	‘<some-target-text>(i)’	Component i of reference and component i of the target are **Equal**
‘<some-reference-text>(i)’	non-existent	**Insert** Component i of reference after component i-1 of target
non-existent	‘<some-target-text>(i)’	**Delete** Component i of target

Color of reference Component i	Color of Target Component i	What does it Mean?
‘<some-reference-text>(i)’	‘<some-target-text>(i)’	**Replace** component i of target with component i of reference

Layout Setup

Summary: Description of how to set up Fully Automatic Layout.

Step 1: Configuring Fully Automatic Layout

Step 1.1 Generation of Meta Data File

The metadata for the fully automatic tool essentially acts as a configuration file, through which the user can specify the paths of the json data for the layout, as well as the layout type which the data belongs to (Fully Automatic Region Parsing in this case). Additionally, the user can also specify other configuration data for the layout which would be rendered from this metadata, such as whether certain outputs should be shown in the plot with a mask or not. The metadata is expected to lie in the `metaData` folder within the `streamlit` folder, and must be a `json` file which follows a specific format.

Shown below is an example metadata file:

```
{
    "metaData":{
        "pageLayout": "Fully Automatic Region Parsing",
        "pageName": "FullyAutomatic",
        "dataPaths": {
            "Class 1":"/path/to/class_1.json",
            "Class 2":"/path/to/class_2.json",
                    .
                    .
                    .
            "Class N":"/path/to/class_N.json"
        },
        "outputMasks": {"output1":1,"output2":0, ... , "output
N":1}

    }
}
```

1. `metaData`:**dict (required)**

   All the relevant data is within the `metaData` attribute of the json data.

2. `pageLayout`:**string (required)**

   `pageLayout` tells the app which layout the data should be rendered in. “Fully Automatic Region Parsing” is the expected value for the data to be rendered in the Fully Automatic Page Layout.

3. `pageName`:**string (required)**

`pageName` is the name of the instance of the page layout for the data. Since there can be multiple instances of the same layout for different data, the `pageName` attribute is essential to identify the particular page of interest.

4. `dataPaths`:**dict (required)**

   `dataPaths` is a dictionary with the keys being the name of the group of data (train/test/val for example), and the values are the path to that `json` data.

5. `outputMasks`:**dict**

   `outputMasks` is a dictionary with the keys `groundTruth` and `modelPrediction`, and their corresponding values a boolean - whether the output should have a mask or not in the visualization plot.

Step 1.2 Setup Directory

Create a directory containing only config files. Each instance of the layout should have a json file. For example, if we are

trying to visualize data for two fully automatic models, the following is a possible directory structure:

```
metadata/
    - fullyAutomatic_model1.json
    - fullyAutomatic_model2.json
```

With each of the json files following the format as described above. Note that no two instances of the layout should have the same `pageName` key.

Multiple Layouts

If you are trying to visualize results of either OCR or Box Supervised Layouts, you can do so, by creating similar metaData files for them as described in their corresponding documentations and then place all of them into this metaData directory.

Example of directory containing all three layouts:

```
metaData/
    - ocr_handwritten.json
    - fullyAutomatic.json
    - boxSupervised.json
```

The docVisor tool will launch single instances of the OCR, Fully Automatic and Box Supervised layouts.

Multiple Instances of the Same Layout

The DocVisor tool allows the user to view multiple instances of the same layout in a single session. For example, if you would like to visualize outputs for two fully automatic models, as well as a box supervised layout and an OCR layout, you can just create mulitple a seperate json file with unique `pageName` key and place the file in the metaData directory.

The metaData directory will look like:

```
metaData/

    - fullyAutomatic_model1.json
    - fullyAutomatic_model2.json
    - ocr_handwritten.json
    - boxSupervised.json
```

For the above directory structure, the DocVisor tool will load two instances of the Fully Automatic layout and a single layout of OCR and Box Supervised layouts.

Step 1.3: Formatting Data Files

The data files are `json` files which contain the data that is to be visualized. Shown below is an example data file:

```
{
    "bhoomi-4422268662325975730":{
        "imagePath":"/data1/hdia_dataset/bhoomi/RGARTHA DIPIKA/
GOML/3076/72.jpg",
        "regions":[
            {
                "groundTruth": [],
                "modelPrediction": [],
                "regionLabel": "Character Line Segment",
                "metrics": {
                    "iou": 0.8205464552270203,
                    "hd": 20.518284528683193
                },
                "id": "bhoomi-4422268662325975730",
                "collection": "bhoomi"
            },
            {
                "groundTruth": [],
                "modelPrediction": [],
                "regionLabel": "Character Line Segment",
                "metrics": {
                    "iou": 0.8043249199353717,
                    "hd": 29.120439557122072
                },
                "id": "bhoomi-4422268662325975730",
                "collection": "bhoomi"
            }
        ]
    }
}
```

1. `Document ID`:**string (required)**

The data is stored on a per-document basis, with an ID being the key for that document's data. In the image shown above, data is shown for only one document. The ID of the document here is `bhoomi-44222686623259757 30`.

2. `imagePath`:**string (required)**

   `imagePath` is the path to the image of the document associated with the outputs to be visualized.

3. `regions`:**dict (required)**

   `regions` is an list of dictionaries. Each dictionary in turn stores the regionwise data of the document. Every region which is to be visualized should be present in `regions` in the above format.

4. a. `groundTruth`:**list (required)**
   b. `modelPrediction`:**list**
   c. `regionLabel`:**string (required)**
   d. `id`:**string (required)**
   e. `metrics`:**dict**

f. `collection`:**string**

`groundTruth` and `modelPrediction` are lists of points, and are of the form `[[x1,y1],[x2,y2], ... , [xn,yn]]`. `groundTruth` is required, but `modelPrediction` need not exist for every region. Similarly, `regionLabel` and `id` are required, but `metrics` and `collection` are not. `regionLabel` is the region class label for that particular region. `metrics` can have the values of various metrics with which the user can sort the data by during visualization. `collection` is additional information pertaining to the collection from which the document was obtained.

Step 1.4 Updating the Config File

In docVisor/config.py file, change the `metaDataDir` to point to the metaData Directory that you have created.

Step 2: Launch the Tool

To launch the tool, you need to run the ./run.sh file. The tool will load on localhost with `port 8501`. If `8501` is pre-occupied, check the terminal to know which exact port in which it has been loaded.

Help and Feedback

For Feedback or queries, you can either visit the github repo and create issues or use the discussion format. For more details, you can mail, `docvisor.iiith@gmail.com.`

Navigation

Summary: Description of Navigation process in Fully Automatic Layout.

The Navigation Process

The navigation system of the Fully Automatic layout allows the user to browse through the various documents in their dataset in an intuitive manner.

The user is served document images from the region they’ve selected to browse through in the `Select Region Label` dropdown at the top of the page.

Once they've done so, the user can use the `previous` and `next` buttons to move back and forth between the images. Additionally, a slider is available for moving across large numbers of images. The number shown above the slider position (the circle that moves) is the current position of the document being served in the image carousel.

Initially, the order of the documents served is random. The user can select metrics to sort the data by, so that the navigation experience would be more logical. Refer to Metrics (page 71) for more information.

Full Document

In addition to the region labels that exist within the data, DocVisor also provides another option called `Full Document` in the `Select Region Label` dropdown. Here, only images on a full-length document basis are served. The metrics used here

for sorting are on the basis of the entire document, since there is not particular region.

Visualization

Summary: Description of Visualization in Fully Automatic layout.

Overview

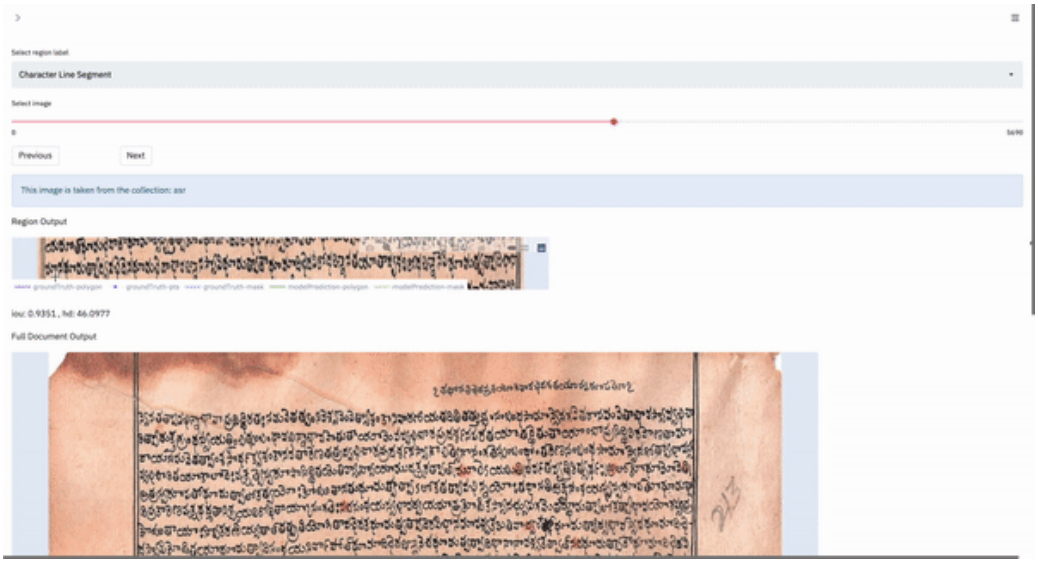
The main feature of the Fully Automatic layout is the visualization of the model's outputs it enables the user to have. The visualization plot offers a visual comparison to the ground-truth data for the same region, as well as for the entire document. The user can visualize the region in a more focused plot, as well as a full-length plot of the document it belongs to. Additionally, the user can also choose to view only the full-document outputs. Refer to Navigation (page 61) for more information.

The plot is interactive, and the user can select or de-select the outputs (polygons, points and masks). These outputs can be toggled from the legend of the plot, which has the color-coded labels for each output. Double-clicking a single output in the

legend would hide all other outputs, and show only the clicked output. Double-clicking on an output twice would display all outputs simultaneously in the plot.

The plot also has features such as zooming in and out, panning and cropping. These options appear in the top-right of the plot, when the user hovers their mouse over it.

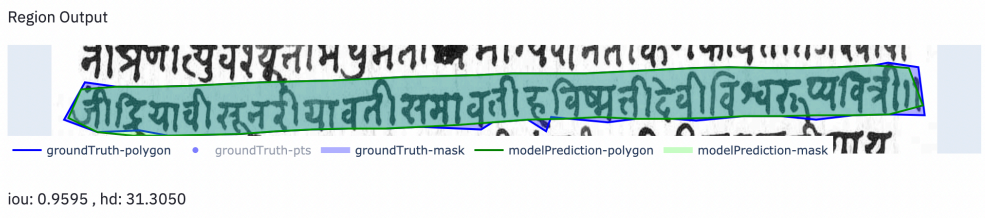
Shown below is a gif of how a user would visualize outputs for a region as well as a full document:



Region Output

The `Region Output` area of the layout displays the visualization plot for the current region served. In `Full Document`, there is no `Region Output` area. The metrics displayed here (`iou` and `hd` in the image below) are of the region only.

Shown below is an example of the `Region Output`:

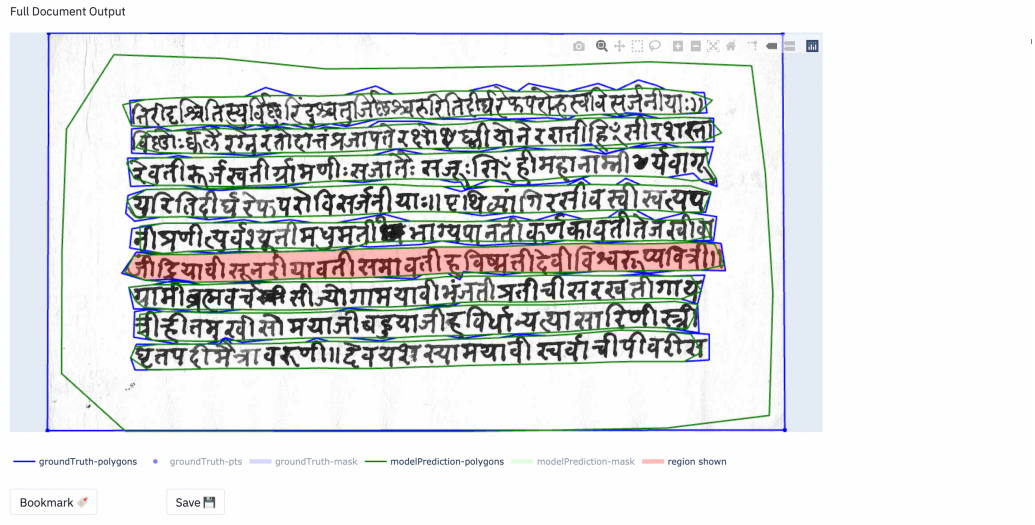


Full Document Output

The `Full Document Output` area of the layout displays the visualization plot for the full document which the current region belongs to. The region in a red highlited area is the region being shown above in the `Region Output` area.

In `Full Document`, the metrics displayed are of the full document.

Shown below is an example of `Full Document Output` that is displayed along with a `Region Output`:



Shown below is an example of `Full Document Output` that is displayed when the user selects `Full Document`:

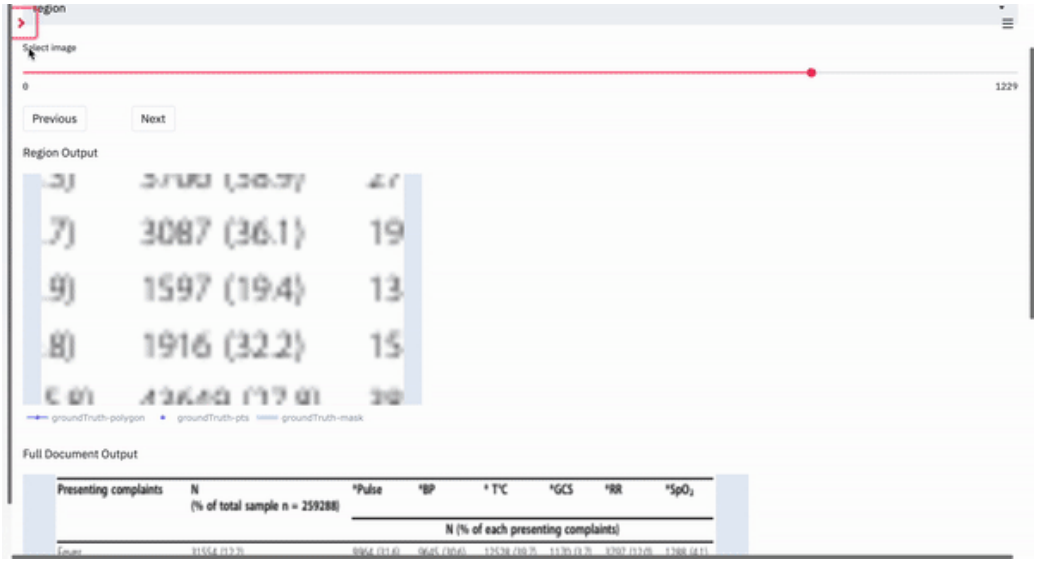
Settings

Summary: Description of Settings in Fully Automatic layout.

Overview

Some settings of the current layout instance (Fully Automatic in this case) can be configured on the left-navbar of the app. The navbar can be hidden by clicking the `x` on the top-right of the navbar, and can be expanded by clicking the `>` in the top-left of the page when it is hidden.

Shown below is a gif of how a user would go about changing the settings:



1. The user can select the main page layout type they would like to choose

(OCR, Fully Automatic Region Parsing and Box-Supervised Region Parsing) in the `Go To` dropdown.

2. In the selected layout, the user can select the instance of the layout they would like to view (in the gif example, the instances were PubNet, FullyAutomatic and DocBank) under the `Choose Page` dropdown.
3. If the user has different data splits in their dataset (train/test/val), the user can select the desired split under the `Select dataset` dropdown. If there are none, the app creates a default `data` split.
4. If the data has metrics computed for the regions as well as the full document, the user can sort the data being served in ascending or descending order according to that metric. Once selected, the documents served during navigation will be in that order. These can be selected under the `Sort By` and `Sort Order`

dropdowns.

Display Options

The outputs selected on the plot legend are temporary selections, i.e. they do not persist between images. Hence, the user can select those outputs they wish to be displayed constantly under the `Display Options` setting. Every output will have a checkbox, and those outputs selected will be continuously displayed between images served during navigation. For the `Region Output`, they are available under `Region Display Options` and for the `Full Document Output` they are available under `Document Display Options`.

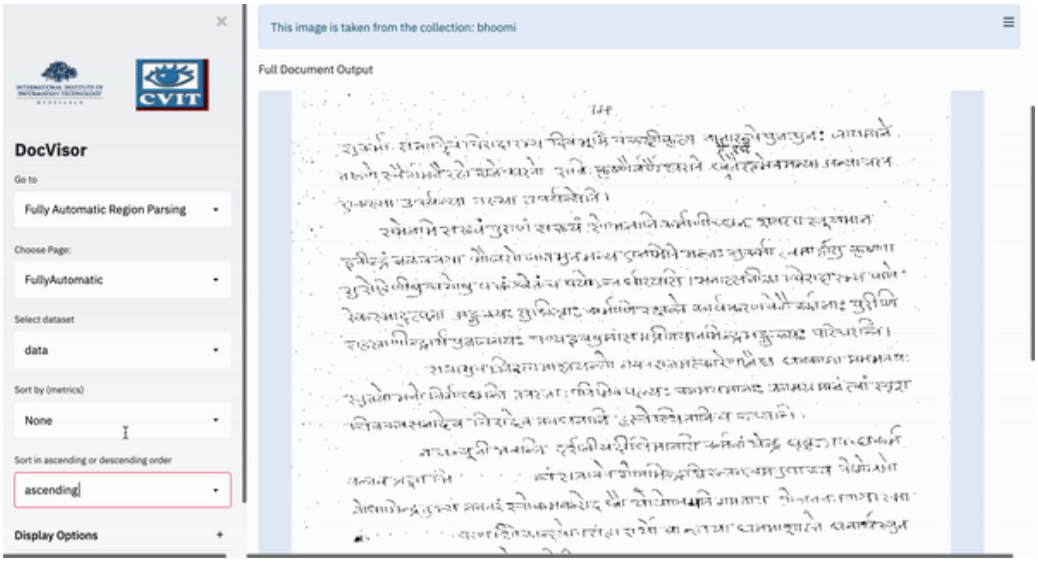
Metrics

Summary: Description of Metrics in Fully Automatic layout.

Sorting by metrics during navigation

If the data has metrics computed for the regions as well as the full document, the user can sort the data being served in ascending or descending order according to that metric. However, if there is any metric that is present for one region/ document, that metric must be present in all regions/documents. Refer to Layout Setup (page 0) for more information on the format of the data.

Once selected, the documents served during navigation will be in the order of that metric. These can be selected under the `Sort By` and `Sort Order` dropdowns. In addition, There is a default `None` option in the cases where the user may not have metrics to sort the data by.



Metrics displayed for each image

For those images which have metrics values present in the data, the values are displayed as text below the visualization plot. When the data is being visualized per region, the metrics shown are for that region only. On the other hand, when `Full Document` is selected as the region, the metrics shown are naturally for the entire document.

Bookmark and Save

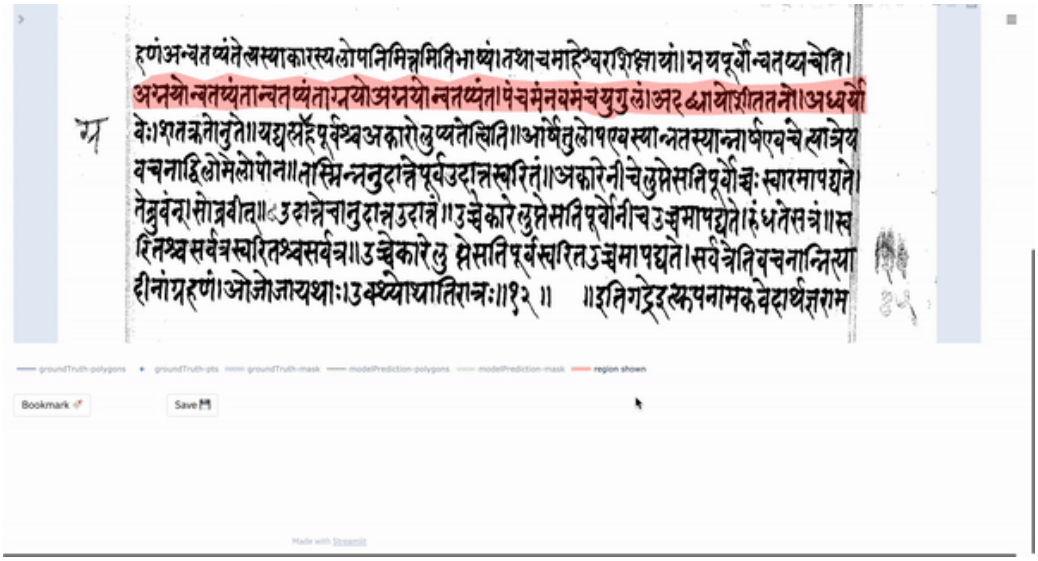
Summary: Description of Bookmarking and Saving in Fully Automatic layout.

Bookmark

During analysis, the user may find an image they would want to come back to later. It would be tedious for the user to have to remember the index of the image, and this is not feasible for a large number of images. Hence, DocVisor offers a bookmarking feature, where the user can bookmark a particular region/document. The steps would be as follows:

1. Click on the `Bookmark 🔖` button at the bottom of the page.
2. Once you have bookmarked atleast one image, the `bookmarks` option is available under the `Select dataset` dropdown.
3. Users can browse through all the bookmarked pages once the `bookmarks` option is selected.

Shown below is a gif of the process. In the gif below, the user bookmarks one image, and then adds another to the bookmarks.



Save

If the user wants to reference the information of a region/document they are interested in later, bookmarking is only a temporary solution: it only persists for a single session. For that reason, the user can save data of the region/document. The steps are as follows:

1. Click the `Save` 💾 button at the bottom of the page.
2. Once saved, a folder is created for the current date, as well as a number

denoting the session of the user. Every new session would create a new folder in the date folder.

3. Within the number folder, three folders `OCR`, `Fully Automatic Region Parsing` and `Box-supervised Region Parsing` can be seen. The desired file will be present in the folder with the name of the instance of the `Fully Automatic Region Parsing` layout.

In bookmarks, there is an additional option of being able to `Save All` 💾 the bookmarked images.

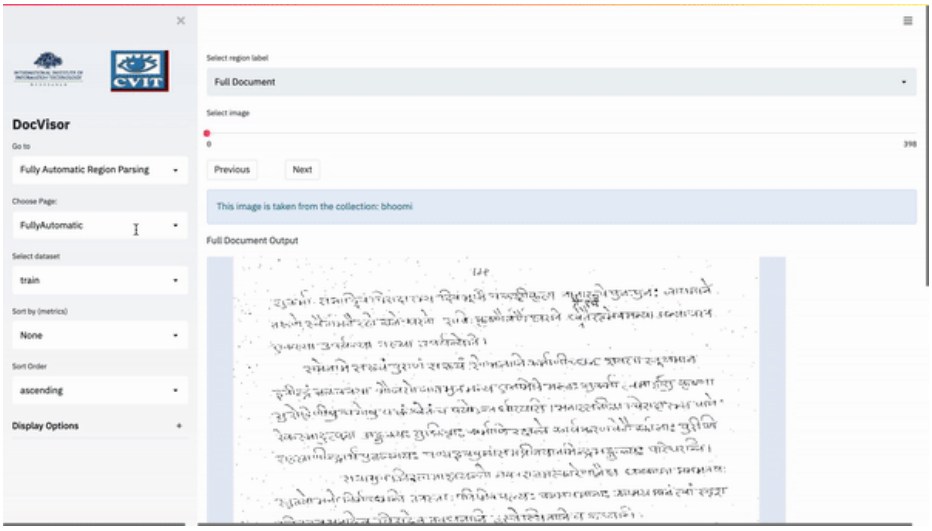
Shown below is an example of how the directory structure would look for the saved files:

```
31_05_2021/
└── 5
    ├── Box-supervised Region Parsing
    │   └── BoundaryNet
    ├── Fully Automatic Region Parsing
    │   ├── DocBank
    │   ├── FullyAutomatic
    │   └── PubTab
    │       └── save.txt
    └── OCR
        ├── OCR-Handwritten
        └── OCR-Printed
```

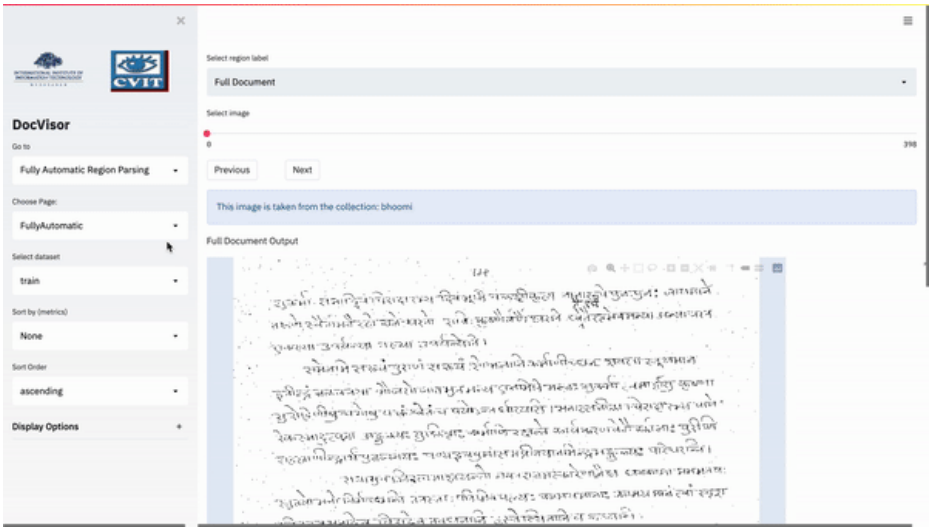
Gallery

Summary: Sample gifs of various scenarios in Fully Automatic layout

1. A gif showing an example visualization of another Fully Automatic Layout instance (PubTabNet in this case):



2. A gif showing an example visualization of another Fully Automatic Layout instance (DocBank in this case):


DocVisor
Go to
Fully Automatic Region Parsing
Choose Page:
FullyAutomatic
Select dataset
train
Sort by (metrics)
None
Sort Order
ascending
Display Options
Select region label
Full Document
Select image
0
398
Previous
Next
This image is taken from the collection: bhoomi
Full Document Output

Layout Setup

Summary: Description of how to set up Box Supervised Layout.

Step 1: Configuring Box Supervised Layout

Step 1.1 Generation of Meta Data File

The metadata for the box supervised tool essentially acts as a configuration file, through which the user can specify the paths of the json data for the layout, as well as the layout type which the data belongs to (Box Supervised Region Parsing in this case). Additionally, the user can also specify other configuration data for the layout which would be rendered from this metadata, such as whether certain outputs should be shown in the plot with a mask or not. The metadata is expected to lie in the `metaData` folder within the `streamlit` folder, and must be a `json` file which follows a specific format.

Shown below is an example metadata file:

```
{
    "metaData":{
        "pageLayout": "Box-supervised Region Parsing",
        "pageName": "BoundaryNet",
        "dataPaths": {
            "Class 1":"/path/to/class_1.json",
            "Class 2":"/path/to/class_2.json",
                    .
                    .
                    .
            "Class N":"/path/to/class_N.json"
        },
        "outputMasks": {"output1":1,"output2":0, ... , "output
N":1}

    }
}
```

1. `metaData`:**dict (required)**

   All the relevant data is within the `metaData` attribute of the json data.

2. `pageLayout`:**string (required)**

   `pageLayout` tells the app which layout the data should be rendered in. “box supervised Region Parsing” is the expected value for the data to be rendered in the box supervised Page Layout.

3. `pageName`:**string (required)**

`pageName` is the name of the instance of the page layout for the data. Since there can be multiple instances of the same layout for different data, the `pageName` attribute is essential to identify the particular page of interest.

4. `dataPaths`:**dict (required)**

   `dataPaths` is a dictionary with the keys being the name of the group of data (train/test/val for example), and the values are the path to that `json` data.

5. `outputMasks`:**dict**

   `outputMasks` is a dictionary with the keys `groundTruth` and `modelPrediction`, and their corresponding values a boolean - whether the output should have a mask or not in the visualization plot.

Step 1.2 Setup Directory

Create a directory containing only config files. Each instance of the layout should have a json file. For example, if we are

trying to visualize data for two box supervised models, the following is a possible directory structure:

```
metadata/
    - boxSupervised_model1.json
    - boxSupervised_model2.json
```

With each of the json files following the format as described above. Note that no two instances of the layout should have the same `pageName` key.

Multiple Layouts

If you are trying to visualize results of either OCR or Fully Automatic Layouts, you can do so, by creating similar metaData files for them as described in their corresponding documentations and then place all of them into this metaData directory.

Example of directory containing all three layouts:

```
metaData/
    - ocr_handwritten.json
    - fullyAutomatic.json
    - boxSupervised.json
```

The docVisor tool will launch single instances of the OCR, Fully Automatic and Box Supervised layouts.

Multiple Instances of the Same Layout

The DocVisor tool allows the user to view multiple instances of the same layout in a single session. For example, if you would like to visualize outputs for two box supervised models, as well as a fully automatic layout and an OCR layout, you can just create mulitple a seperate json file with unique `pageName` key and place the file in the metaData directory.

The metaData directory will look like:

```
metaData/

    - ocr_handwritten.json
    - fullyAutomatic.json
    - boxSupervised_model1.json
    - boxSupervised_model2.json
```

For the above directory structure, the DocVisor tool will load two instances of the Box Supervised layout and a single layout of OCR and Fully Automatic layouts.

Step 1.3: Formatting Data Files

The data files are `json` files which contain the data that is to be visualized. Shown below is an example data file:

```
[
    {
        "imagePath": "/home/user/new_jpg_data/Bhoomi_data/image
s/AAVARNI VYAKHYA/GOML/991/19.jpg",
        "outputs": {
            "ground_truth": [],
            "gcn_output": [],
            "encoder_output": []
        },
        "metrics": {
            "iou": 0.6072467801928089,
            "hd": 115.10864433221339
        },
        "regionLabel": "Physical Degradation",
        "bbox": [256, 7, 302, 165],
        "collection": "bhoomi"
    }
]
```

The json file with the data for the box-supervised layout is a list of dictionaries. Each dictionary represents the data of a single region of a document. The structure of each dictionary is as follows:

1. `imagePath`:**string (required)**

`imagePath` is the path to the image of the document associated with the outputs to be visualized.

2. `outputs`:**dict (required)**

   `outputs` is an dictionary of lists. Each list in turn stores the points of the output polygon to be visualized. In the above example, there are three outputs to be visualized: `ground_truth`, `gcn_output` and `encoder_output`.

3. `metrics`:**dict**

   `metrics` is a dictionary containing the values of all the metrics computed for the data. In the above example, there are two metrics present for the data: `iou` and `hd`.

4. `regionLabel`:**string (required)**

   `regionLabel` contains the label of the region class for the data. In the above example, the `regionLabel` is `Physical Degradation`.

5. `bbox`:**list (required)**

`bbox` is the user-annotated bounding box for the current region.

6. `collection`:**string**

   `collection` is additional information such as the collection the current data belongs to in the dataset.

Step 1.4 Updating the Config File

In docVisor/config.py file, change the `metaDataDir` to point to the metaData Directory that you have created.

Step 2: Launch the Tool

To launch the tool, you need to run the ./run.sh file. The tool will load on localhost with `port 8501`. If `8501` is pre-occupied, check the terminal to know which exact port in which it has been loaded.

Help and Feedback

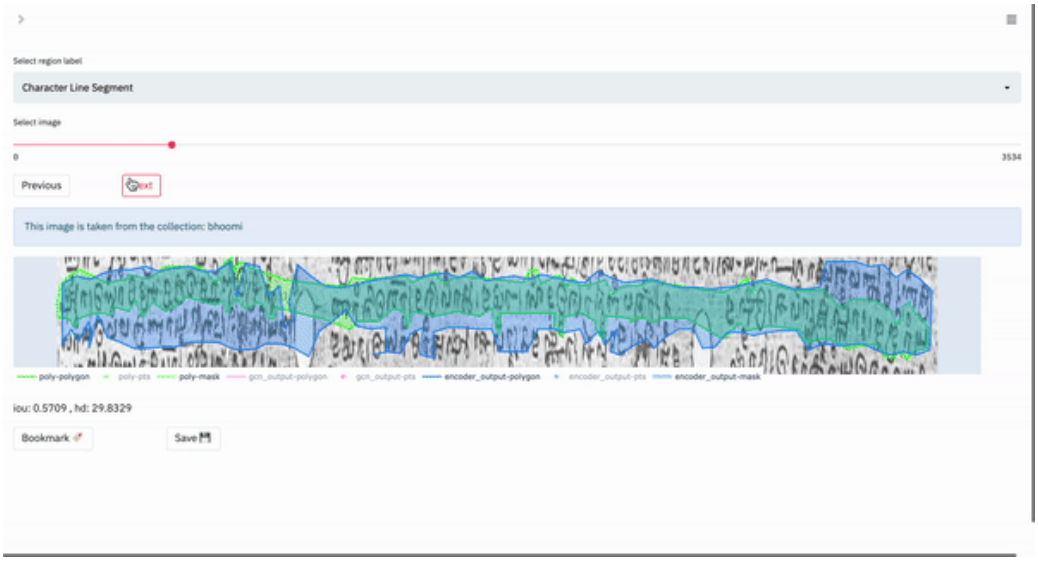
For Feedback or queries, you can either visit the github repo and create issues or use the discussion format. For more details, you can mail, `docvisor.iiith@gmail.com.`

Navigation

Summary: Description of Navigation process in Box Supervised Layout.

The Navigation Process

The navigation system of the Box Supervised layout allows the user to browse through the various documents in their dataset in an intuitive manner.



The user is served document images from the region they’ve selected to browse through in the `Select Region Label` dropdown at the top of the page.

Once they’ve done so, the user can use the `previous` and `next` buttons to move back and forth between the images. Additionally, a slider is available for moving across large numbers of images. The number shown above the slider position (the circle that moves) is the current position of the document being served in the image carousel.

Initially, the order of the documents served is random. The user can select metrics to sort the data by, so that the navigation experience would be more logical. Refer to Metrics (page 93) for more information.

Visualization

Summary: Description of Visualization in Box Supervised Region Parsing layout.

Overview

The main feature of the Box Supervision layout is the visualization of the model's outputs it enables the user to have. The visualization plot offers a visual comparison to the ground-truth data for the same region. Refer to Navigation (page 86) for more information on navigating between images.

The plot is interactive, and the user can select or de-select the outputs (polygons, points and masks). These outputs can be toggled from the legend of the plot, which has the color-coded labels for each output. Double-clicking a single output in the legend would hide all other outputs, and show only the clicked output. Double-clicking on an output twice would display all outputs simultaneously in the plot.

The plot also has features such as zooming in and out, panning and cropping. These options appear in the top-right of the plot, when the user hovers their mouse over it.

Shown below is a gif of how a user would visualize outputs for a region:

Settings

Summary: Description of Settings in Box Supervised Region Parsing layout.

Overview

Some settings of the current layout instance (BoundaryNet in this case) can be configured on the left-navbar of the app. The navbar can be hidden by clicking the `x` on the top-right of the navbar, and can be expanded by clicking the `>` in the top-left of the page when it is hidden.

Shown below is a gif of how a user would go about changing the settings:

1. The user can select the main page

layout type they would like to choose (OCR, Fully Automatic Region Parsing and Box-Supervised Region Parsing) in the `Go To` dropdown.

2. In the selected layout, the user can select the instance of the layout they would like to view under the `Choose Page` dropdown.
3. If the user has different data splits in their dataset (train/test/val), the user can select the desired split under the `Select dataset` dropdown. If there are none, the app creates a default `data` split.
4. If the data has metrics computed for the regions, the user can sort the data being served in ascending or descending order according to that metric. Once selected, the regions served during navigation will be in that order. These can be selected under the `Sort By` and `Sort Order` dropdowns.

Display Options

The outputs selected on the plot legend are temporary selections, i.e. they do not persist between images. Hence, the user can select those outputs they wish to be displayed constantly under the `Display Options` setting. Every output will have a checkbox, and those outputs selected will be continuously displayed between images served during navigation.

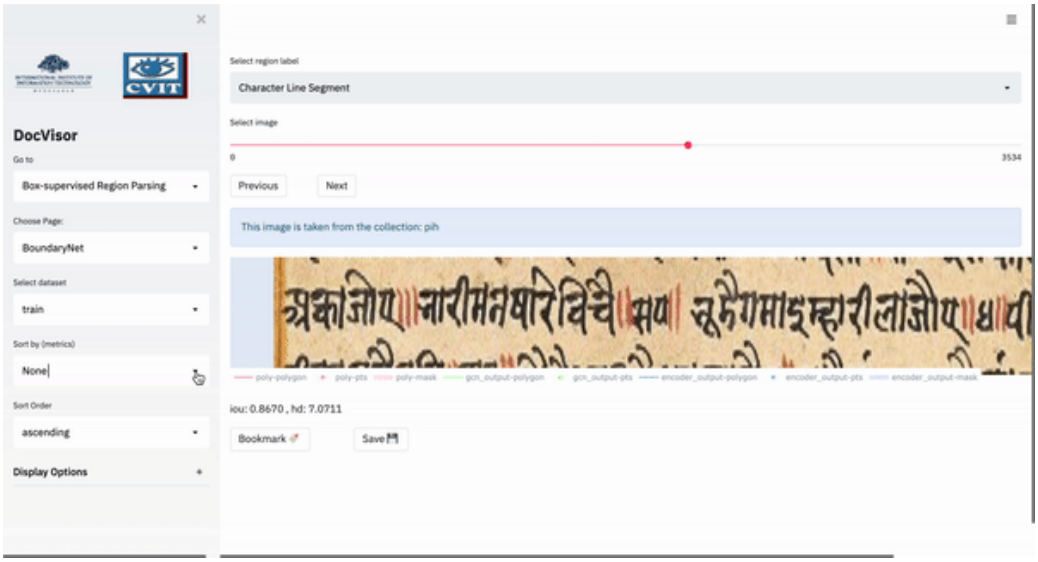
Metrics

Summary: Description of Metrics in Box Supervised Region Parsing layout.

Sorting by metrics during navigation

If the data has metrics computed for the region data, the user can sort the data being served in ascending or descending order according to that metric. However, if there is any metric that is present for one region, that metric must be present in all regions. Refer to Layout Setup (page 0) for more information on the format of the data.

Once selected, the region images served during navigation will be in the order of that metric. These can be selected under the `Sort By` and `Sort Order` dropdowns. In addition, There is a default `None` option in the cases where the user may not have metrics to sort the data by.



For those images which have metrics values present in the data, the values are displayed as text below the visualization plot.

Bookmark and Save

Summary: Description of Bookmarking and Saving in Box Supervised Region Parsing layout.

Bookmark

During analysis, the user may find an image they would want to come back to later. It would be tedious for the user to have to remember the index of the image, and this is not feasible for a large number of images. Hence, DocVisor offers a bookmarking feature, where the user can bookmark a particular region. The steps would be as follows:

1. Click on the `Bookmark 🔖` button at the bottom of the page.
2. Once you have bookmarked atleast one image, the `bookmarks` option is available under the `Select dataset` dropdown.
3. Users can browse through all the bookmarked images once the `bookmarks` option is selected.

Shown below is a gif of the process. In the gif below, there is an existing bookmark in `bookmarks`, and the user bookmarks another image.

Save

If the user wants to reference the information of a region they are interested in later, bookmarking is only a temporary solution: it only persists for a single session. For that reason, the user can save data of the region. The steps are as follows:

1. Click the `Save` 💾 button at the bottom of the page.
2. Once saved, a folder is created for the

current date, as well as a number denoting the session of the user. Every new session would create a new folder in the date folder.

3. Within the number folder, three folders `OCR`, `Fully Automatic Region Parsing` and `Box-supervised Region Parsing` can be seen. The desired file will be present in the folder with the name of the instance of the `Box Supervised Region Parsing` layout.

In bookmarks, there is an additional option of being able to `Save All` 💾 the bookmarked images.

Shown below is an example of how the directory structure would look for the saved files:

```
31_05_2021/
└── 5
    ├── Box-supervised Region Parsing
    │   └── BoundaryNet
    │       └── save.txt
    ├── Fully Automatic Region Parsing
    │   ├── DocBank
    │   ├── FullyAutomatic
    │   └── PubTab
    └── OCR
        ├── OCR-Handwritten
        └── OCR-Printed
```

About the Tool's author

Summary: All about the contributors to this tool. Feel free to contact them over their mail id's provided.

Contributors

Box-Supervised Region Parsing Layout

- Pranav Tadimeti (pranav.tadimeti@students.iiit.ac.in)

OCR Layout

- Khadiravana Belagavi (khadiravana.belagavi@research.iiit.ac.in)
- Suman Michael (sumanmichael01@gmail.com)

Fully-Automatic Region Parsing Layout

- Pranav Tadimeti (pranav.tadimeti@students.iiit.ac.in)

Supervision

- Ravi Kiran Sarvadevabhatla (ravi.kiran@iiit.ac.in)