alternat

Kepler

Dec 24, 2020

CONTENTS

alternat is a collection of open-source toolsets with the ambition of lowering the barrier of adopting accessibility solutions**. alternat** helps to generate default intelligible alternative text for images in websites.

Based on our experience, just adding alt-text is not a complete solution but collecting images to be annotated is a big part of the task. Keeping in mind above two requirements the current version of alternat offers two features:

- Collect images from a website
- Generate recommended alternative text

CHAPTER

ONE

WHY ALTERNAT

70% of the sites are inaccessible and the inaccessibility is causing a loss of 7 billion every year. In spite of availability of accessibility standards for long and tools to point out where you are falling short, there is a shortage of solutions which allow you to implement them with ease. One of these areas is – **Alternate Text (alt text)**.

The alt-text attribute of image tag in html is supposed to make images in websites accessible. But in practice, one doesn't see it meaningfully implemented. Someone has to go through all the images in question and craft a corresponding alt text based on context. The investment to do it can become high and it could take time to author the content.

What if there is a library, which can be integrated in your projects, that provides a recommended alt text for a given image, which can be either passed on as is or as a recommendation to a reviewer? **alternat** just does that.

1.1 Installing alternat

1.1.1 Installing alternat macOS

Install using pypi (macOS)

1. Install Node (>=v.12)
2. Install Python (>=3.8)
3. Install alternat:

```
pip install alternat
```

4. Install apify

```
mkdir -p ~/.alternat && cd ~/.alternat && npm install apify && cd -
```

Install from source (macOS)

1. Install git
2. Install Node (>=v.12)
3. Install Python (>=3.8)
4. Open terminal and clone the repo:

```
git clone https://github.com/keplerlab/alternat.git
```

5. Change the directory to the directory where you have cloned your repo

```
$cd path_to_the_folder_repo_cloned
```

6. Install apify

```
mkdir -p ~/.alternat && cd ~/.alternat && npm install apify && cd -
```

7. Install alternat using setup.py

```
python setup.py install
```

Installation using Anaconda python (macOS)

1. Install git
2. Install Node (>=v.12)
3. Install Python (>=3.8)
4. Open terminal and clone the repo:

```
git clone https://github.com/keplerlab/alternat.git
```

5. Change the directory to the directory where you have cloned your repo

```
$cd path_to_the_folder_repo_cloned
```

6. Create conda environment and install dependencies using alternat.yml file

```
cd setup_scripts
conda env create --name alternat --file=alternat.yml
```

7. Activate newly created environment:

```
conda activate alternat
```

8. Install apify

```
mkdir -p ~/.alternat && cd ~/.alternat && npm install apify && cd -
```

Installation using Docker (macOS)

1. Download and Install Docker Desktop for Mac using link: https://docs.docker.com/docker-for-mac/install/
2. Clone this repo
3. Change your directory to the cloned repo.
4. Open terminal and run following commands:

```
cd <path-to-repo> //you need to be in your repo folder
docker-compose build
```

5. Start docker container using this command:

```
docker-compose up
```

6. In a new terminal window open terminal and enter into alternat docker container using command:

```
docker-compose exec alternat bash
```

1.1.2 Installing alternat ubuntu

Install using pypi (ubuntu)

1. Install Node (>=v.12)
2. Install Python (>=3.8)
3. Install alternat:

```
pip install alternat
```

4. Install apify by first downloading install_apify_ubuntu.sh located at setup_scripts folder in alternat Repo link and then executing downloaded script

```
sudo sh install_apify_ubuntu.sh
```

Install from source (ubuntu)

1. Install git
2. Install Node (>=v.12)
3. Install Python (>=3.8)
4. Open terminal and clone the repo

   git clone https://github.com/keplerlab/alternat.git

5. Change the directory to the directory where you have cloned your repo

```
$cd path_to_the_folder_repo_cloned
```

6. Install apify by executing given script

```
cd setup_scripts
sudo sh install_apify_ubuntu.sh
```

7. Install alternat using setup.py

```
python setup.py install
```

Installation using Anaconda python (ubuntu)

1. Install git
2. Install Node (>=v.12)
3. Install Python (>=3.8)
4. Open terminal and clone the repo:

```
git clone https://github.com/keplerlab/alternat.git
```

5. Change the directory to the directory where you have cloned your repo

```
$cd path_to_the_folder_repo_cloned
```

6. Create conda environment and install dependencies using alternat.yml file

```
cd setup_scripts
conda env create --name alternat --file=alternat.yml
```

7. Activate newly created environment:

```
conda activate alternat
```

8. Install apify by executing given script

```
cd setup_scripts
sudo sh install_apify_ubuntu.sh
```

Installation using Docker (ubuntu)

1. Download and Install Docker Desktop for Mac using link: https://docs.docker.com/docker-for-mac/install/
2. Clone this repo
3. Change your directory to the cloned repo.
4. Open terminal and run following commands:

```
cd <path-to-repo> //you need to be in your repo folder
docker-compose build
```

5. Start docker container using this command:

```
docker-compose up
```

6. In a new terminal window open terminal and enter into alternat docker container using command:

```
docker-compose exec alternat bash
```

1.1.3 Installing alternat windows

Install using pypi (Windows)

1. Install Node (>=v.12)
2. Install Python (>=3.8)

3. Install apify by first downloading install_from_pypi_windows.bat script located at setup_scripts folder in alternat repo [link](https://github.com/keplerlab/alternat/blob/main/setup_scripts/install_from_pypi_windows.bat) and then executing downloaded script inside new windows powershell prompt:

```
.\install_from_pypi_windows.bat
```

Install from source (Windows)

1. Install git
2. Install Node (>=v.12)
3. Install Python (>=3.8)
4. Open terminal and clone the repo

   git clone https://github.com/keplerlab/alternat.git

5. Change the directory to the directory where you have cloned your repo

```
$cd path_to_the_folder_repo_cloned
```

6. Install apify by executing given script inside windows powershell prompt:

```
cd setup_scripts
.\install_from_source_windows.bat
```

Installation using Anaconda python (Windows)

1. Install git
2. Install Node (>=v.12)
3. Install Python (>=3.8)
4. Open terminal and clone the repo inside windows powershell prompt:

```
git clone https://github.com/keplerlab/alternat.git
```

5. Change the directory to the directory where you have cloned your repo

```
$cd path_to_the_folder_repo_cloned
```

6. Create conda environment and install dependencies using enviorment_windows.yml file

```
cd setup_scripts
conda env create --name alternat --file=enviorment_windows.yml
```

7. Activate newly created environment:

```
conda activate alternat
```

8. Install apify by executing given script inside windows powershell promp:

```
cd setup_scripts
.\install_apify_windows.bat
```

Installation using Docker (Windows)

1. Download and Install Docker Desktop for Mac using link: https://docs.docker.com/docker-for-mac/install/
2. Clone this repo
3. Change your directory to the cloned repo.
4. Open terminal and run following commands:

```
cd <path-to-repo> //you need to be in your repo folder
docker-compose build
```

5. Start docker container using this command inside windows powershell or cmd promp:

```
docker-compose up
```

6. In a new windows powershell or cmd window open terminal and enter into alternat docker container using command:

```
docker-compose exec alternat bash
```

1.2 Running alternat in 5 minutes

Alternat can run in the following mode:

1. Application Mode: In application mode, users use Command Line Interface (CLI) to run the alternat. We have created a sample app.py which will run the application via CLI command.
2. Library Mode: In Library mode, users install alternat from pip (python package installer) and can use in new or existing application via the library.
3. Service Mode: In Service mode, REST API endpoint is exposed where a POST request can be submitted with a JSON request to get the alt-text generated by alternat.

Below is the 5-minute guide to run alternat in the modes described above:

1. Application Mode:

   Collection:

   Use case : collect and store images from a URL and store them in a folder

```
python app.py collect --url="https://page_url" --output-dir-path="sample/
↪images/test"
```

Generation:

Use case: generate alt-text for images in input folder and save result in a directory)

```
python app.py generate --input-dir-path="sample/images_with_text" --
↪output-dir-path="results"
```

Use case: generate alt-text for single image and save result json in directory

```
python app.py generate --input-image-file-path="sample/images_with_text/
↪sample1.png" --output-dir-path="results"
```

2. Library Mode:

Collection

```
# import the alternat library
from alternat.collection import Collector

# instantiate the collector
collector = Collector()

# Download images from url and saves image files in  output_dir_path
# Optional parameters, download_recursive if True crawls whole site␣
↪mentioned in
# url by visiting each link recursively and downloads images
# collect_using_apify in future more crawlers will be supported this␣
↪parameter
# ensures that apify crawler is used.
collector.process(url, output_dir_path, download_recursive, collect_using_
↪apify)
```

Generation:

```
# import the alternat library
from alternat.generation import Generator

# instantiate the generator
generator = Generator()

# generate alt text from file (file at location sample/images_with_text/
↪sample1.png
# and results saved at location folder results)
generator.generate_alt_text_from_file("sample/images_with_text/sample1.png
↪", "results")
```

(continues on next page)

(continued from previous page)

```
# OR

# generate alt text from base64 image
generator.generate_alt_text_from_base64(base64_image_string)
```

3. Service Mode:

   In this mode, alternat exposes web API to generate alt-text for an image. Alternat use python based API framework - fastAPI to create APIs. fastAPI comes with a lightweight python server uvicorn which is used to expose the API. To start the server :

```
# Go to api folder
cd api

# run this command to start the service
uvicorn message_processor:app --port 8080 --host 0.0.0.0 --reload
```

   The following web APIs are available:

```
# send a post request with base64 image to the Web Server
URL: http://localhost:8080/generate_text_base64
body: { base64: "base64_image_str"}

# send a post request with URL of the image to the Web Server
URL: http://localhost:8080/generate_text_url
body: { url: "url_of_the_image"}
```

1.3 Understanding alternat

alternat features are centered around tasks. Following table features break up across each task:

Task	Description	Options	Details
Collection	Scans the website and downloads images	Uses puppeteer to crawl the web page.	We are using apify - A puppeteer scrapper that crawls website using the headless chrome https://apify.com
Generate	Generates alt-text using, image captioning, OCR and images labels	Azure ML API	Use azure CV API for caption & OCR
		Google ML API based.	Use Google vision OCR and Image Labelling
		Open source based.	Use pytorch based model for OCR (EasyOCR) as well as image captioning

Library offers the flexibility of choosing either or both tasks and selecting suitable options from each task. Options are called drivers in alternat lingo. So, if you want to use azure for alt-text generation then you initialize the generator with azure driver. Same goes for google and "opensource" driver. Read the options as drivers.

There are few reasons for providing 3 drivers:

- Azure and google gives ready to use API, essentially lowering the barrier to get started.

- Most of the organizations don't have the data to train their own model for OCR and image captioning.
- Open source is a free alternative but can be little less accurate in few situations.

The tradeoff here is between cost and accuracy.

The OCR function is responsible for reading text from images. However, most of the ML API for OCR would treat single line as one text blob and might lead to unexpected out-of-order OCR text. For this reason, alternat comes with its own clustering implementation for OCR. alternat by default applies a clustering algorithm to create nearby data as a single text blob and combines them into a single line thereby generating more in-order human friendly OCR text.

1.4 Configuring alternat

Alternat can be configured at a global generator level or at the driver level with settings related to individual driver. We discuss configuration for both the generator and the driver for Application as well as Library mode below:

1.4.1 Configure Generator

Following configuration parameters are available for generator:

1. **DEBUG:** Setting the debug value to true will generate confidence level value for each of the OCR line detected by the system. In the debug output, it gives the line height and its ratio compared to the image height. This information is useful if you want to tune confidence level threshold value for drivers and OCR line height to image height ratio for filtering out insignificant text in the image.
2. **ENABLE_OCR_CLUSTERING:** Alternat comes with its own clustering rule which clusters (blobs) OCR data to create final OCR text from it. This is enabled by default and can be disabled by setting this value to false.

Application Mode:

You can find sample configuration for all the three drivers namely: opensource, azure, and google under "path-to-repo/sample/generator_driver_conf/<drivername>.json". Inside the JSON file you will find the following configuration parameter: GENERATOR: {DEBUG: false, ENABLE_OCR_CLUSTERING: true}

Here is an example to use the generator configuration for driver with name <drivername>:

```
python app.py generate --output-dir-path="results" --input-image-file-path="sample/
↪images_with_text/sample1.png" --driver-config-file-path="sample/generator_driver_
↪conf/<drivername>.json"
```

Where <drivername> is the name of the driver (opensource, azure or google)

Based on the setting for DEBUG and ENABLE_OCR_CLUSTERING in the sample/generator_driver_conf/<drivername>.json file the above command will generate the result and dump it inside "results" folder.

Library Mode:

In Library mode, you can directly interact with alternat library API to set the generator level config via a json object. Following is an example that will walk you through the same:

Once you have setup the alternat using "pip install alternat" you can open the python shell and run these commands to set generator config

```
# import the Generator library
from alternat.generation import Generator

# instantiate the Generator for opensource driver (you can pass "azure" or
# "google" when instantiating to the let the library know the driver you want
# to use.
generator = Generator()

# get the current generator settings
# This will return the existing configuration
# {'DEBUG': False, 'ENABLE_OCR_CLUSTERING': True}
generator.get_config()

# set debug to true
generator_config = {"DEBUG": True}
generator.set_config(generator_config)

# or disable OCR clustering
generator_config = {"ENABLE_OCR_CLUSTERING": False}
generator.set_config(generator_config)

# or set values for both the parameters in one go
generator_config = {"DEBUG": True, "ENABLE_OCR_CLUSTERING": False}
generator.set_config(generator_config)

# run generator over an image and dump the output inside "results" folder
# this will run with DEBUG=true.
generator.generate_alt_text_from_file("sample/images_with_text/sample1.png", "results
↪")
```

1.4.2 Configure Driver

Generator comes with 3 drivers:

1. **Opensource:** Currently opensource drivers uses a pytorch based trained model for image captioning based on this repo It also uses EasyOCR for generating OCR text in case image have text in it. This is the default driver for generator, does not require any kind of registration and is free to use.
2. **Azure:** Uses Azure computer vision API to describe an image, generate OCR and also provide labels to the image. To use this driver, you need to register for computer vision API from Microsoft which will give you the SUBSCRIPTION_KEY and ENDPOINT URL to access the API.
3. **Google:** Uses Google computer vision API to generate OCR and provide labels to the image. To use this driver, you need to register for google computer vision API, download the google cloud service credentials file on your system and set the path to it in the driver configuration parameter ABSOLUTE_PATH_TO_CREDENTIAL_FILE (will be discussed below)

The following generator driver settings are available:

1. **CAPTION_CONFIDENCE_THRESHOLD:** Decimal based threshold to filter out caption data. For example, if you only want captions with confidence level above say 70%, then set this value to 0.70. This is most useful when using "azure" as driver as Microsoft compute vision API has support for describing an image. This option is also used in opensource driver.
2. **OCR_CONFIDENCE_THRESHOLD:** Decimal based threshold to filter out OCR data. For example, if you want OCR text with confidence level about say 50%, then set this value to 0.50.
3. **LABEL_CONFIDENCE_THRESHOLD:** Decimal based threshold to filter out label data. For example, if you want labels with confidence level about say 80%, then set this value to 0.80. This is useful when using google and azure driver as both the APIs have support for labelling image.
4. **OCR_HEIGHT_RATIO_TO_IMAGE_THRESHOLD:** Decimal based threshold to filter out OCR text which does not occupy a major portion of image and is practically irrelevant even if detected by the system. This threshold considers the ratio of the height of the text and the image to decide whether the text needs to be filtered out or not. For example, if you want OCR data only when the line height is greater than let's say 1.5% then set this value to 0.015 in the config.
5. **SUBSCRIPTION_KEY:** This is the subscription key for azure computer vision API, and is only required when using **azure** as the driver.
6. **ENDPOINT:** This is the API endpoint URL for azure computer vision API, and is only required when using **azure** as the driver.
7. **AZURE_RATE_LIMIT_ON:** This enables rate limiting when using azure driver in free account. Azure has a limit of 30 requests / minute in free tier account and when running alternat over a set of images this limit can hit very quickly. Alternat avoids this by sleeping for 30 sec by default and trying again. This setting is enabled by default. This setting is only required when using **azure** as the driver.
8. **AZURE_RATE_LIMIT_TIME_IN_SEC:** This is the rate limit time in sec. Alternat will sleep for these many seconds (30 by default) when azure rate limiting is reached in free tier account. To increase the sleep timer from 30 to say 40 seconds, set the value of this parameter to 40. This setting is only required when using **azure** as the driver.
9. **ABSOLUTE_PATH_TO_CREDENTIAL_FILE:** This setting holds the absolute path to the google credentials file (required to access the Google cloud services and computer vision API). This setting is only required when using **google** as the driver.

Let's see how to configure the above parameters in both the application and library mode.

Application Mode:

You can find sample configuration for all the three drivers namely: opensource, azure, and google under "path-to-repo/sample/generator_driver_conf/<drivername>.json". Inside the configuration file, you find all the parameters above with default values already set. To change these values and run generator use the following command:

```
python app.py generate --output-dir-path="results" --input-image-file-path="sample/
↪images_with_text/sample1.png" --driver-config-file-path="sample/generator_driver_
↪conf/<drivername>.json"
```

Where <drivername> is the name of the driver (opensource, azure or google)

Library Mode:

Once you have setup the alternat using "pip install alternat" you can open the python shell and run these commands to set generator config:

```
# import the Generator library
from alternat.generation import Generator

# instantiate the Generator for opensource driver (you can pass "azure" or
# "google" when instantiating to the let the library know the driver you want
# to use.

# for opensource
generator = Generator()

# or for azure
generator = Generator("azure")

# or for google
generator = Generator("google")

# get the current generator driver settings
# This will return the existing configuration based on the driver
generator.get_driver_config()

# set threshold value for caption, OCR and label
generator_driver_config = {"CAPTION_CONFIDENCE_THRESHOLD": 0.2, "OCR_CONFIDENCE_
↪THRESHOLD": 0.3, "LABEL_CONFIDENCE_THRESHOLD":0.75}
generator. generator.set_driver_config (generator_driver_config)

# or set OCR_HEIGHT_RATIO_TO_IMAGE_THRESHOLD
generator_driver_config = {"OCR_HEIGHT_RATIO_TO_IMAGE_THRESHOLD":0.015}
generator. generator.set_driver_config (generator_driver_config)

# or set subscription key and endpoint URL for azure
generator_driver_config = {"SUBSCRIPTION_KEY": "yoursubscriptionkey", "ENDPOINT":
↪"https://<ENTER_PROJECT_NAME>.cognitiveservices.azure.com/"}
generator. generator.set_driver_config (generator_driver_config)

# run generator over an image and dump the output inside "results" folder
# this will run with DEBUG=true.
generator.generate_alt_text_from_file("sample/images_with_text/sample1.png", "results
↪")
```

1.4.3 Configure Web API

Web API use opensource driver by default. Both application mode and Web API internally rely on the alternat library. To configure Web API for different driver and configuration the following changes are required:

1. Navigate to **api** folder.
2. Locate file **message_processor.py**. Here you will see the Generator being instantiated (just like in library mode).
3. Use the samples from **Library Mode** section under *Configure Driver* to configure web API using alternat library.

Here is an example to say change the driver to azure. In **message_processor.py**,

```
# find the following statement
generator = Generator()

# for azure, change the statement to this
generator = Generator("azure")

# following statements change the driver specific configuration
# add this to set subscription key and endpoint URL for azure
generator_driver_config = {"SUBSCRIPTION_KEY": "yoursubscriptionkey", "ENDPOINT":
↪"https://<ENTER_PROJECT_NAME>.cognitiveservices.azure.com/"}

# add this to update the threshold value for caption, OCR and label
generator_driver_config = {"CAPTION_CONFIDENCE_THRESHOLD": 0.2, "OCR_CONFIDENCE_
↪THRESHOLD": 0.3, "LABEL_CONFIDENCE_THRESHOLD":0.75}

#  add this to update OCR_HEIGHT_RATIO_TO_IMAGE_THRESHOLD
generator_driver_config = {"OCR_HEIGHT_RATIO_TO_IMAGE_THRESHOLD":0.015}

# add this to set the configuration
generator.set_driver_config(generator_driver_config)
```

1.5 Using alternat

1.5.1 Application Mode via CLI (Command Line Interface)

Alternat comes with a python-based CLI app **app.py** which provides commands to run collection and generation task. Below we give some example on how to use this app:

Collection:

1. **Collect and store images from a URL and store them in a folder sample/images/test**

```
python app.py collect --url="https://page_url" --output-dir-path="sample/
↪images/test"
```

2. **Collect and store the images from a URL recursively and store them in a folder sample/images/test**

```
python app.py collect --url="https://page_url" --output-dir-path="sample/
↪images/test" --download-recursively=true
```

Generation

1. **Generate alt-text for the images in a directory name sample/images_with_text and save data in directory structure results**

```
python app.py generate --input-dir-path="sample/images_with_text" --output-
↪dir-path="results"
```

2. **Generate alt-text for a single image in a folder sample/images_with_text and save its result in a directory inside results:**

```
python app.py generate --input-image-file-path=./sample/images_with_text/
↪sample1.png --output-dir-path=./results
```

3. **Generate alt-text based on user defined (custom) config for driver azure :**

```
python app.py generate --input-image-file-path=./sample/images/sample1.jpg --
↪output-dir-path=./results --driver-config-file-path=./sample/generator_
↪driver_conf/azure.json
```

The above command can be changed based on the driver by using the driver sample files under sample/generator_driver_conf. For example, to use google driver change the –driver-config-file-path to "sample/generator_driver_conf/google.json".

1.5.2 Library Mode

With library mode, users can integrate alternat in their existing applications as well. In library mode the package is installed via pip and can be import into python applications directly. Below are some examples on using the library mode for collection and generation tasks:

Collection:

Download the image from a site given its URL to specified folder location:

```
# import the alternat library
from alternat.collection import Collector

# instantiate the collector
collector = Collector()

# Download images from url and saves image files in  output_dir_path
# Optional parameters, download_recursive if True crawls whole site␣
↪mentioned in
# url by visiting each link recursively and downloads images
# collect_using_apify in future more crawlers will be supported this␣
↪parameter
# ensures that apify crawler is used.
collector.process(url, output_dir_path, download_recursive, collect_using_
↪apify
```

Generator:

1. **Generate alt-text for a single image in a folder "results" and save its result in a directory inside result/test:**

```
# import the Generator
from alternat.generation import Generator

# instantiate the generator (uses opensource driver by default)
generator = Generator()

# to use a specific driver pass the driver name when instantiating. For e.g,␣
↪to use
```

(continues on next page)

(continued from previous page)

```
# azure driver use
generator = Generator("azure")

# generate the alt text
generator.generate_alt_text_from_file("sample/images_with_text/sample1.png",
↪"results")
```

2. **Generate alt-text for a single image in base64 image:**

```
# import the Generator
from alternat.generation import Generator

# instantiate the generator (uses opensource driver by default)
generator = Generator()

# generate the alt text
base64_image_str = "base64-image-data-here"
generator.generate_alt_text_from_base64(base64_image_str)
```

1.5.3 Service Mode

In this mode, alternat exposes web API to generate alt-text for an image. Alternat use python based API framework - fastAPI to create APIs. fastAPI comes with a lightweight python server uvicorn which is used to expose the API. To start the server :

```
# Go to api folder
cd api

# run this command to start the service
uvicorn message_processor:app --port 8080 --host 0.0.0.0 --reload
```

The following web APIs are available:

```
# send a post request with base64 image to the Web Server
URL: http://localhost:8080/generate_text_base64
body: { base64: "base64_image_str"}

# send a post request with URL of the image to the Web Server
URL: http://localhost:8080/generate_text_url
body: { url: "url_of_the_image"}
```

1.6 Extending alternat

1.6.1 Adding new generator driver

To add a new driver, use the existing driver architecture. Alternat currently supports 3 drivers

1. google
2. azure
3. opensource

Note:

All the drivers need to output JSON data and adhere to the schema here : alternat/generation/data/analyzer_output.json. **Failing to do so would impact proper functioning of the driver**.

Follow the steps to add a new driver:

1. Create a new folder with name custom inside alternat/generation

```
cd alternat/generation
mkdir custom
```

2. Create the same file structure as in the existing drivers

```
# move inside **custom** folder
cd custom

# create the following files inside **custom** folder
# command will differ on windows shell
touch analyzer.py
touch config.py
```

3. Copy paste the contents of config file from opensource/config.py
4. Copy paste the contents of analyzer file from opensource/analyzer.py
5. Edit the following methods in custom/analyzer.py to add your own functionality.
    1. open analyzer.py in custom/analyzer.py
    2. overwrite **describe_image** method to add your custom implementation of image captioning.

```
# overwrite this method to extract caption

  def describe_image(self, image: PIL_Image):
  """Describe image using your custom solution.

  :param image: PIL Image object
  :type image: PIL_Image
  """

      # add the extracted caption data here instead of empty dictionary
      # the data needs to adhere to the sample JSON data at alternat/
↪data/analyzer_output.json
      self.data[self.actions.DESCRIBE] = {}
```

    3. overwrite the **extract_labels** method to add your custom implementation of getting label data.

```
def extract_labels(self, image: PIL_Image):
"""Extract labels of image using open source solution.

:param image: PIL Image object.
:type image: PIL_Image
"""

    # add the extracted label data here instead of empty dictionary
    # the data needs to adhere to the sample JSON data at alternat/
↪data/analyzer_output.json
    self.data[self.actions.LABELS] = {}
```

    4. overwrite the **ocr_analysis** method to add your custom implementation for ocr extraction.

```
def ocr_analysis(self, image: PIL_Image):
"""Does OCR Analysis using EasyOCR.

:param image: PIL Image object.
:type image: PIL_Image
"""

    # add the ocr extracted data here instead of empty dictionary
    # the data needs to adhere to the sample JSON data at alternat/
↪data/analyzer_output.json
    self.data[self.actions.OCR] = {}
```

6. Expose the driver to the generator library so it is available across the application. Following are the steps to the same:

    1. open alternat/generation/generator.py (This is the library for alternat)
    2. Import the Analyzer & Config class of your custom driver.

    ```
    from alternat.generation.custom.config import Config as␣
    ↪CustomAnalyzerConfig
    from alternat.generation.custom.analyze import AnalyzeImage as␣
    ↪CustomAnalyzer
    ```

    2. find the **Drivers** class and add your custom driver there.

    ```
    class Drivers:
    """Driver name for alternat Library.
    """
        OPEN = "opensource"
        MICROSOFT = "azure"
        GOOGLE = "google"

        # custom driver added here
        CUSTOM = "custom"
    ```

    3. modify **_set_current_driver** method and add your custom driver in if-elif-else statements.

    ```
    # TODO: This behavior will be changed later one so no method␣
    ↪modification is required.

    def _set_current_driver(self):
    """Sets the current driver internally within the application.

    :raises InvalidGeneratorDriver: Driver name is invalid or not␣
    ↪implemented.
    """
        if self.CURRENT_DRIVER == Drivers.OPEN:
            setattr(Config, Config.CURRENT_ANALYZER.__name__,␣
    ↪OpenAnalyzer)
        elif self.CURRENT_DRIVER == Drivers.MICROSOFT:
            setattr(Config, Config.CURRENT_ANALYZER.__name__,␣
    ↪MicrosoftAnalyzer)
        elif self.CURRENT_DRIVER == Drivers.GOOGLE:
            setattr(Config, Config.CURRENT_ANALYZER.__name__,␣
    ↪GoogleAnalyzer)

        # custom driver added
    ```

(continues on next page)

(continued from previous page)

```
    elif self.CURRENT_DRIVER == Drivers.CUSTOM:
        setattr(Config, Config.CURRENT_ANALYZER.__name__,␣
↪CustomAnalyzer)
    else:
        raise InvalidGeneratorDriver(self.ALLOWED_DRIVERS)
```

4. modify **_get_current_driver** method and add your custom driver in if-elif-else statements.

```
def _get_current_driver_conf_cls(self):
"""Retreives the driver configuration class based on the␣
↪currently driver

:raises InvalidGeneratorDriver: Driver name is invalid or not␣
↪implemented.
:return: [description]
:rtype: [type]
"""
    current_driver_cls = None
    if self.CURRENT_DRIVER == Drivers.OPEN:
        current_driver_cls = OpenAnalyzerConfig
    elif self.CURRENT_DRIVER == Drivers.MICROSOFT:
        current_driver_cls = MicrosoftAnalyzerConfig
    elif self.CURRENT_DRIVER == Drivers.GOOGLE:
        current_driver_cls = GoogleAnalyzerConfig

    # custom driver added
    elif self.CURRENT_DRIVER == Drivers.CUSTOM:
        current_driver_cls = CustomAnalyzerConfig
    else:
        raise InvalidGeneratorDriver(self.ALLOWED_DRIVERS)

    return current_driver_cls
```

7. The new custom driver will be available for use now.

1.7 Troubleshooting and FAQ

1. If you get error like **Error: spawn wmic.exe ENOENT** while running collect command (using apify) in alternat on **Microsoft Windows** This indicates that the wmic utility's directory is not found on your PATH. Open the advanced System Properties window (you can open the System page with Windows+Pause/Break) and on the Advanced tab, click Environment Variables. In the section for system variables, find PATH (or any capitalization thereof). Add this entry to it:

```
%SystemRoot%\System32\Wbem
```

Note that entries are delimited by semicolons.

2. In some cases with running collect command on windows you might get error: Chrome is downloaded but fails to launch on Node.js 14 If you get an error that looks like this when trying to launch Chromium:

(node:15505) UnhandledPromiseRejectionWarning: Error: Failed to launch the browser process! spawn /Users/. . . /node_modules/puppeteer/.local-chromium/mac-756035/chrome-mac/Chromium.app/Contents/MacOS/Chromium ENOENT This means that the browser was downloaded but failed to be extracted correctly. The most common cause is a bug in Node.js v14.0.0 which broke extract-zip, the module Puppeteer uses to extract browser downloads into the right place. The bug was fixed in Node.js v14.1.0, so

please make sure you're running that version or higher. Alternatively, if you cannot upgrade, you could downgrade to Node.js v12, but we recommend upgrading when possible.

1.8 Alternat Reference

The alternat reference guide will walk through the implementation for both Generation and Collection, the alternat library, and the app.

1.8.1 Generation - Analyzer

Google Analyzer

class alternat.generation.google.analyze.**AnalyzeImage**

Bases: alternat.generation.base.analyzer.AnalyzeImageBase

Google Analyzer driver class.

Parameters **AnalyzeImageBase** (*[type]*) – Driver base class.

describe_image(*image: <module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/alternat/envs/v0.1.4/ packages/PIL/Image.py'>*)

Describe image (used for captioning) - Not availble in Google Computer Vision API

Parameters **image** (*PIL_IMAGE*) – [description]

extract_labels(*image: <module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/alternat/envs/v0.1.4/ packages/PIL/Image.py'>*)

Extract labels of image using Google Computer Vision API.

Parameters **image** (*PIL_IMAGE*) – PIL Image object.

Raises **Exception** – Google Cloud specific error messages based on request.

handle(*image_path: str = None*, *base64_image: str = None*, *actions: list = None*) → dict

Entry point for the driver. Implements all the action and generates data for rule engine.

Parameters

- **image_path** (*str, optional*) – Path to image on disk, defaults to None
- **base64_image** (*str, optional*) – Base64 image string, defaults to None
- **actions** (*list, optional*) – list of actions to run, defaults to None (all actions execute)

Returns [description]

Return type dict

ocr_analysis(*image: <module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/alternat/envs/v0.1.4/lib/ packages/PIL/Image.py'>*)

Does OCR analysis using Google Computer Vision API. Also runs the alternat clustering rule if app is configured for it.

Parameters **image** (*PIL_IMAGE*) – PIL Image object.

set_environment_variables()

Sets environment variable GOOGLE_APPLICATION_CREDENTIALS based on config.

Azure Analyzer

class alternat.generation.microsoft.analyze.**AnalyzeImage**

Bases: alternat.generation.base.analyzer.AnalyzeImageBase

Azure / Microsoft Analyzer driver class.

Parameters **AnalyzeImageBase** (*[type]*) – Driver base class.

describe_image(*image: <module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/alternat/envs/v0.1.4/ packages/PIL/Image.py'>*)

Describe image using Azure Vision API.

Parameters **image** (*PIL_Image*) – PIL Image object

extract_labels(*image: <module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/alternat/envs/v0.1.4/ packages/PIL/Image.py'>*)

Extract labels of image using Azure Vision API.

Parameters **image** (*PIL_Image*) – PIL Image object.

handle(*image_path: str = None*, *base64_image: str = None*, *actions: list = None*) → dict

Entry point for the driver. Implements all the action and generates data for rule engine.

Parameters

- **image_path** (*str, optional*) – Path to image on disk, defaults to None
- **base64_image** (*str, optional*) – Base64 image string, defaults to None
- **actions** (*list, optional*) – list of actions to run, defaults to None (all actions execute)

Returns [description]

Return type dict

is_clean(*image: <module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/alternat/envs/v0.1.4/lib/pytho packages/PIL/Image.py'>*) → bool

Check if the image has proper resolution, and is clean.

:param image:PIL Image object. :type image: PIL_Image :return: [description] :rtype: bool

modifyBoundingBoxData(*bounding_box: list*)

Transform bounding box data as per the convention. Azure API return bounding box info in the format [left, top, right, top, right, bottom, left, bottom] which is transformed to format [{x: left, y: top}, {x: right, y: top}, {x: right, y: bottom}, {x: left, y: bototm}].

Parameters **bounding_box** (*list*) – Bounding box data form Azure API.

Returns [description]

Return type [type]

ocr_analysis(*image: <module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/alternat/envs/v0.1.4/lib/ packages/PIL/Image.py'>*)

Does OCR Analysis using Azure Vision API.

Parameters **image** (*PIL_Image*) – PIL Image object.

resize_image(*image: <module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/alternat/envs/v0.1.4/lib/ packages/PIL/Image.py'>*)

Resize image (maintaining aspect ratio) if width / height > 5000 pixels (API constrain from Azure)

Parameters **image** (*PIL_Image*) – [description]

Opensource Analyzer

class alternat.generation.opensource.analyze.**AnalyzeImage**

Bases: alternat.generation.base.analyzer.AnalyzeImageBase

Opensource driver class.

Parameters **AnalyzeImageBase** (*[type]*) – Driver base class.

describe_image(*image: <module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/alternat/envs/v0.1.4/ packages/PIL/Image.py'>*)

Describe image using open source solution. Not implemented right now.

Parameters **image** (*PIL_Image*) – PIL Image object

extract_labels(*image: <module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/alternat/envs/v0.1.4/ packages/PIL/Image.py'>*)

Extract labels of image using open source solution. Not implemented righ now.

Parameters **image** (*PIL_Image*) – PIL Image object.

handle(*image_path: str = None*, *base64_image: str = None*, *actions: list = None*) → dict

Entry point for the driver. Implements all the action and generates data for rule engine.

Parameters

- **image_path** (*str, optional*) – Path to image on disk, defaults to None
- **base64_image** (*str, optional*) – Base64 image string, defaults to None
- **actions** (*list, optional*) – list of actions to run, defaults to None (all actions execute)

Returns [description]

Return type dict

modifyBoundingBoxData(*bounding_box: list*)

Transform bounding box data as per the convention. EasyOCR return bounding box info in the format [left, top, right, top, right, bottom, left, bottom] which is transformed to format [{x: left, y: top}, {x: right, y: top}, {x: right, y: bottom}, {x: left, y: bototm}].

Parameters **bounding_box** (*list*) – Bounding box data form EasyOCR.

Returns [description]

Return type [type]

ocr_analysis(*image: <module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/alternat/envs/v0.1.4/lib/ packages/PIL/Image.py'>*)

Does OCR Analysis using EasyOCR.

Parameters **image** (*PIL_Image*) – PIL Image object.

1.8.2 Generation - Rule Engine

OCR Rule

class alternat.generation.rules.ocr_handler.**OCRDataHandler**(*input_data*, *confidence_threshold: float = None*, *ocr_filter_threshold: float = None*)

Bases: alternat.generation.base.action_data_handler.ActionDataHandler

Rule for processing OCR data from driver.

Parameters **ActionDataHandler** (*[type]*) – Base class for rule.

Initialize the handler with input data and confidence threshold (if available)

Parameters

- **input_data** (*[type]*) – Data from driver.
- **confidence_threshold**(*float, optional*) – Confidence threshold to filter OCR with low threshold, defaults to None (Driver config default)
- **ocr_filter_threshold**(*float, optional*) – Confidence threshold to filter OCR data based on line height ratio to image height.

apply(*interim_result: dict*) → dict

Process intermin result from previous rules in the chain and run OCR rule.

Parameters **interim_result** (*dict*) – Intermediate results from previous rules in the chain.

Returns [description]

Return type dict

has_data() → bool

Checks whethere OCR data is avalible in the input data.

Returns [description]

Return type bool

process_ocr() → dict

Process the OCR data from the driver and filter it on the basis of line confidence threshold value and the ratio of line height to image height. Based on the configuration also invokes alternat clustering implementation (default to True)

Returns [description]

Return type dict

Caption Rule

class alternat.generation.rules.caption_handler.**CaptionDataHandler**(*input_data: dict*, *confidence_threshold: float = None*)

Bases: alternat.generation.base.action_data_handler.ActionDataHandler

Rule for processing caption data from driver.

Parameters **ActionDataHandler** (*[type]*) – Base class for rule.

Initialize the handler with input data and confidence threshold (if available)

Parameters

- **input_data** (*dict*) – Data from driver.
- **confidence_threshold** (*float, optional*) – Confidence threshold to filter captions with low threshold, defaults to None (Driver config default)

apply(*interim_result: dict*) → dict

Process intermin result from previous rules in the chain and run caption rule.

Parameters **interim_result** (*dict*) – Intermediate results from previous rules in the chain.

Returns [description]

Return type dict

has_data() → bool

Checks whethere caption data is avalible in the input data.

Returns [description]

Return type bool

Label Rule

class alternat.generation.rules.label_handler.**LabelDataHandler**(*input_data*, *confidence_threshold: float = None*)

Bases: alternat.generation.base.action_data_handler.ActionDataHandler

Rule for processing label data from driver.

Parameters **ActionDataHandler** (*[type]*) – Base class for rule.

Initialize the handler with input data and confidence threshold (if available)

Parameters

- **input_data** (*[type]*) – Data from driver.
- **confidence_threshold** (*float, optional*) – Confidence threshold to filter labels with low threshold, defaults to None (Driver config default)

apply(*interim_result: dict*) → dict

Process intermin result from previous rules in the chain and run label rule.

Parameters **interim_result** (*dict*) – Intermediate results from previous rules in the chain.

Returns [description]

Return type dict

has_data() → bool
Checks whethere label data is avalible in the input data.

Returns [description]

Return type bool

1.8.3 Library

class alternat.generation.generator.**Drivers**
Bases: object

Driver name for alternat Library.

GOOGLE = 'google'

MICROSOFT = 'azure'

OPEN = 'opensource'

class alternat.generation.generator.**Generator**(*driver_name: str = None*)
Bases: object

Generator class to implement alternat Library.

Raises

- **InvalidGeneratorDriver** – Driver Invalid
- **InvalidGeneratorDriver** – Driver Invalid
- **InvalidGeneratorDriver** – Driver Invalid
- **OutputDirPathNotGiven** – Output director path is not given.

Returns [description]

Return type [type]

Initializes generator with driver to use.

Parameters **driver_name** (*str, optional*) – Name of the driver], defaults to None (open-source)

Raises **InvalidGeneratorDriver** – Driver name is invalid or not implemented.

ALLOWED_DRIVERS = ['opensource', 'azure', 'google']

DEFAULT_DRIVER = 'opensource'

generate_alt_text_from_base64(*base64_image: str*)
Generates alt-text from base64 image string.

Parameters **base64_image** (*str*) – base64 image string

Returns [description]

Return type [type]

generate_alt_text_from_file(*input_image_path: str*, *output_dir_path: str*)
Generates alt-text from file on disk.

Parameters

- **input_image_path** (*str*) – Path to image to be processed.
- **output_dir_path** (*str*) – Path to directory where the results needs to be saved.

Returns [description]

Return type [type]

get_config()
Get the generator level config in the form of JSON.

Returns [description]

Return type [type]

get_current_driver()
Get the current driver.

Returns [description]

Return type [type]

get_driver_config()
Get the driver config in the form of JSON. Retreives public members [name: value] pair from the driver config class.

Returns [description]

Return type [type]

set_config(*conf*)
Sets the generator level configuration parameters passed via JSON.

Parameters **conf** (*[type]*) – Generator configuration parameters with values.

set_driver_config(*conf: dict*)
Sets the driver config parameters using the JSON passed. There is one-to-one mapping between key in json and driver class public members.

Parameters **conf** (*dict*) – Configuration JSON to set the driver configuration.

class alternat.collection.collector.**Collector**
Bases: object

process(*url: str*, *output_dir_path: str*, *download_recursive: bool = False*, *collect_using_apify: bool = False*)
Collects image from the url into the output directory

Parameters

- **url** (*str*) – [description]
- **output_dir_path** (*str*) – [description]
- **download_recursive** (*bool, optional*) – [description], defaults to False
- **collect_using_apify** (*bool, optional*) – [description], defaults to False