TextRecognitionDataGenerator Documentation

Release latest

Edouard Belval

Aug 04, 2022

Contents

Since the name is quite long, all subsequent refrences will be under the acronym TRDG.

If you are new to the project, start with the tutorial section!

CHAPTER 1

Installation

1.1 Official package

TRDG has a pip package with a matching name.

```
pip install trdg
```

Once that is installed, the `trdg` binary should be in your PATH.

1.2 From source

If you want to add a new language The easiest way to use the tool is by cloning the official repo.

```
git clone https://github.com/Belval/TextRecognitionDataGenerator
```

Then you need to install the dependencies. It is recommended to use a virtual environment for those.

```
pip3 install -r requirements.txt
```

If you want to use the handwritten text generation feature, you need to install the `-hw` dependencies.

```
pip3 install -r requirements-hw.txt
```

Once that is done, you can move to the tutorial for tips and tricks on how to use TRDG!

CHAPTER 2

Overview

2.1 Most useful arguments

1. `-i`, `--input_file`

   Use it when the provided dictionaries do not fit your usecase. Each line will become an image, if your `-c` parameter is high enough.

2. `-c`, `--count`

   Self-explanatory parameter, but one you will probably want to change. The default value is 1000.

3. `-l`, `--language`

   This argument is especially important if you want to generate data using a specific script. It changes the dictionary to be used (`-l fr` is equivalent to `-i dicts/fr.txt`), but most importantly it changes the default fonts to take one that supports the language's script. Passing a chinese dictionary without changing the language will cause invalid images to be generated.

4. `-t`, `--thread_count`

   Another self-explanatory parameter, yet very important as most computers these days ship with a multi-core CPU. Setting this to `-t 8` makes TRDG create 8 processes to generate the data.

5. `-f`, `--format`

   By default, all generated images will be 32 pixels high (or wide if you use `-or 1`). Now that might be too small for you. `-f` allows you to make bigger images.

2.2 Getting help

As with most CLI tools, TRDG's help is accessible through the `-h` argument.

If you need more information on a specific argument, find its definition in the reference. If even that does not do, feel free to open an issue on the official repository.

```
usage: trdg [-h] [--output_dir [OUTPUT_DIR]] [-i [INPUT_FILE]] [-l [LANGUAGE]]
            -c [COUNT] [-rs] [-let] [-num] [-sym] [-w [LENGTH]] [-r]
            [-f [FORMAT]] [-t [THREAD_COUNT]] [-e [EXTENSION]]
            [-k [SKEW_ANGLE]] [-rk] [-wk] [-bl [BLUR]] [-rbl]
            [-b [BACKGROUND]] [-hw] [-na NAME_FORMAT] [-d [DISTORSION]]
            [-do [DISTORSION_ORIENTATION]] [-wd [WIDTH]] [-al [ALIGNMENT]]
            [-or [ORIENTATION]] [-tc [TEXT_COLOR]] [-sw [SPACE_WIDTH]]
            [-cs [CHARACTER_SPACING]] [-m [MARGINS]] [-fi] [-ft [FONT]]
            [-ca [CASE]]

Generate synthetic text data for text recognition.

optional arguments:
  -h, --help            show this help message and exit
  --output_dir [OUTPUT_DIR]
                        The output directory
  -i [INPUT_FILE], --input_file [INPUT_FILE]
                        When set, this argument uses a specified text file as
                        source for the text
  -l [LANGUAGE], --language [LANGUAGE]
                        The language to use, should be fr (French), en
                        (English), es (Spanish), de (German), or cn (Chinese).
  -c [COUNT], --count [COUNT]
                        The number of images to be created.
  -rs, --random_sequences
                        Use random sequences as the source text for the
                        generation. Set '-let','-num','-sym' to use
                        letters/numbers/symbols. If none specified, using all
                        three.
  -let, --include_letters
                        Define if random sequences should contain letters.
                        Only works with -rs
  -num, --include_numbers
                        Define if random sequences should contain numbers.
                        Only works with -rs
  -sym, --include_symbols
                        Define if random sequences should contain symbols.
                        Only works with -rs
  -w [LENGTH], --length [LENGTH]
                        Define how many words should be included in each
                        generated sample. If the text source is Wikipedia,
                        this is the MINIMUM length
  -r, --random          Define if the produced string will have variable word
                        count (with --length being the maximum)
  -f [FORMAT], --format [FORMAT]
                        Define the height of the produced images if
                        horizontal, else the width
  -t [THREAD_COUNT], --thread_count [THREAD_COUNT]
                        Define the number of thread to use for image
                        generation
  -e [EXTENSION], --extension [EXTENSION]
                        Define the extension to save the image with
  -k [SKEW_ANGLE], --skew_angle [SKEW_ANGLE]
                        Define skewing angle of the generated text. In
                        positive degrees
  -rk, --random_skew    When set, the skew angle will be randomized between
                        the value set with -k and it's opposite
```

(continues on next page)

(continued from previous page)

```
  -wk, --use_wikipedia  Use Wikipedia as the source text for the generation,
                        using this paremeter ignores -r, -n, -s
  -bl [BLUR], --blur [BLUR]
                        Apply gaussian blur to the resulting sample. Should be
                        an integer defining the blur radius
  -rbl, --random_blur   When set, the blur radius will be randomized between 0
                        and -bl.
  -b [BACKGROUND], --background [BACKGROUND]
                        Define what kind of background to use. 0: Gaussian
                        Noise, 1: Plain white, 2: Quasicrystal, 3: Pictures
  -hw, --handwritten    Define if the data will be "handwritten" by an RNN
  -na NAME_FORMAT, --name_format NAME_FORMAT
                        Define how the produced files will be named. 0:
                        [TEXT]_[ID].[EXT], 1: [ID]_[TEXT].[EXT] 2: [ID].[EXT]
                        + one file labels.txt containing id-to-label mappings
  -d [DISTORSION], --distorsion [DISTORSION]
                        Define a distorsion applied to the resulting image. 0:
                        None (Default), 1: Sine wave, 2: Cosine wave, 3:
                        Random
  -do [DISTORSION_ORIENTATION], --distorsion_orientation [DISTORSION_ORIENTATION]
                        Define the distorsion's orientation. Only used if -d
                        is specified. 0: Vertical (Up and down), 1: Horizontal
                        (Left and Right), 2: Both
  -wd [WIDTH], --width [WIDTH]
                        Define the width of the resulting image. If not set it
                        will be the width of the text + 10. If the width of
                        the generated text is bigger that number will be used
  -al [ALIGNMENT], --alignment [ALIGNMENT]
                        Define the alignment of the text in the image. Only
                        used if the width parameter is set. 0: left, 1:
                        center, 2: right
  -or [ORIENTATION], --orientation [ORIENTATION]
                        Define the orientation of the text. 0: Horizontal, 1:
                        Vertical
  -tc [TEXT_COLOR], --text_color [TEXT_COLOR]
                        Define the text's color, should be either a single hex
                        color or a range in the ?,? format.
  -sw [SPACE_WIDTH], --space_width [SPACE_WIDTH]
                        Define the width of the spaces between words. 2.0
                        means twice the normal space width
  -cs [CHARACTER_SPACING], --character_spacing [CHARACTER_SPACING]
                        Define the width of the spaces between characters. 2
                        means two pixels
  -m [MARGINS], --margins [MARGINS]
                        Define the margins around the text when rendered. In
                        pixels
  -fi, --fit            Apply a tight crop around the rendered text
  -ft [FONT], --font [FONT]
                        Define font to be used
  -ca [CASE], --case [CASE]
                        Generate upper or lowercase only. arguments: upper or
                        lower. Example: --case upper
```

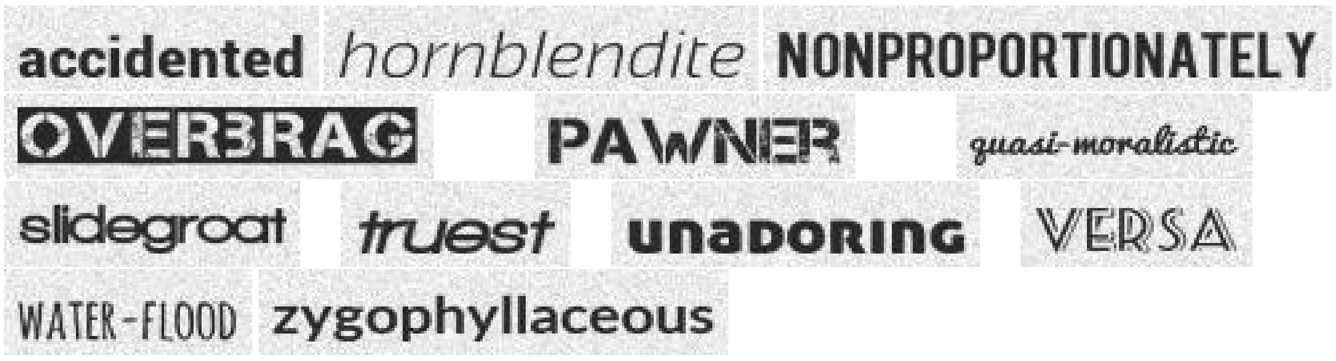
CHAPTER 3

Tutorial

TextRecognitionDataGenerator comes with an (hopefully) easy to use CLI. The tutorial is actually multiple tutorials, combined in a single page. Feel free to skip sections that are not relevant to your use case.

3.1 Just generating data

Fun fact, you don't need to use any command line arguments if you want English data generated using multiple fonts. Indeed, simply running `python3 run.py` will create 1000 English, single word images in the `out/` directory such as these:



Now maybe 1000 is too many or too few for your usecase. You can add the `-c` argument to set how many examples will be generated.

```
python3 run.py -c 10
```

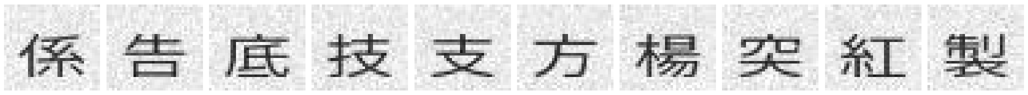
As expected, you will find 10 examples in the `out/` directory.

3.2 Generating Chinese data

This is a common usecase, and one that is easy with TRDG.

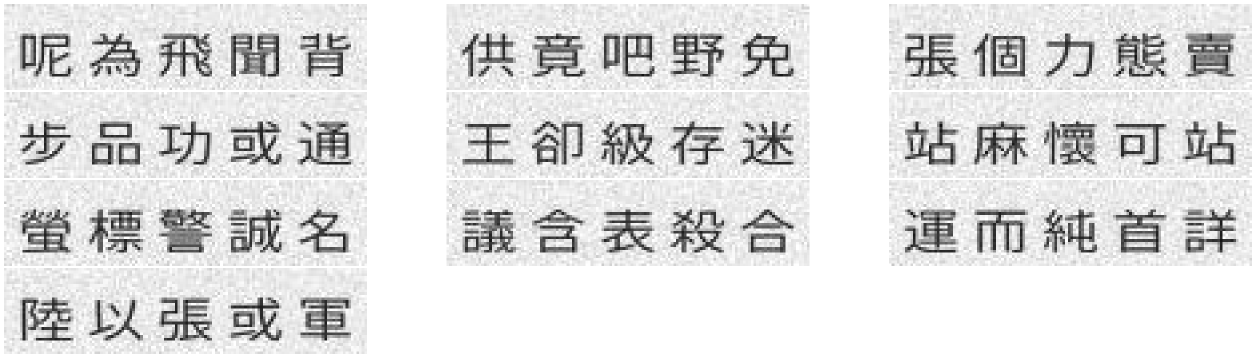
```
python3 run.py -c 10 -l cn
```

This will generate 10 samples using the Chinese dictionary that can be found in in `dicts/cn.txt`:



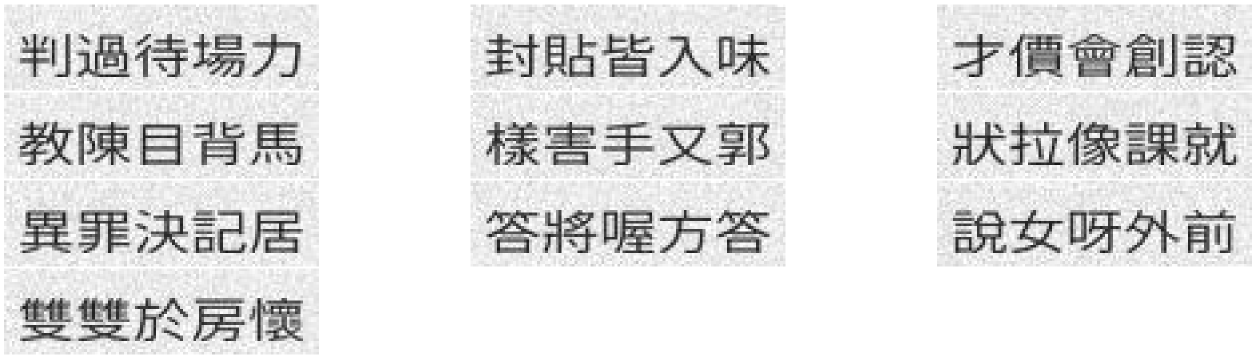
Since the concept of word in Chinese is a bit trickier, the dictionary is made of single characters (make your own!). Let's do this again with `-w 5` to get something prettier.

```
python3 run.py -c 10 -l cn -w 5
```



Now that looks better, but what's up with the spacing between the characters? We would rather have no spaces. Add `-sw 0`.

```
python3 run.py -c 10 -l cn -w 5 -sw 0
```



Asian scripts can be written top to bottom, you might want to add the `-or 1` argument to get vertical text.

```
python3 run.py -c 10 -l cn -w 5 -sw 0 -or 1
```

主啊習構灣
口斯預與石
同當思週著
喔司形化列
基迷懷定碟
密五畫月策
治卻致北測
素亞結已輯
著組文玩員
間越待壞疑

You can do much and more with TRDG, if you run into a missing feature, simply open an issue.

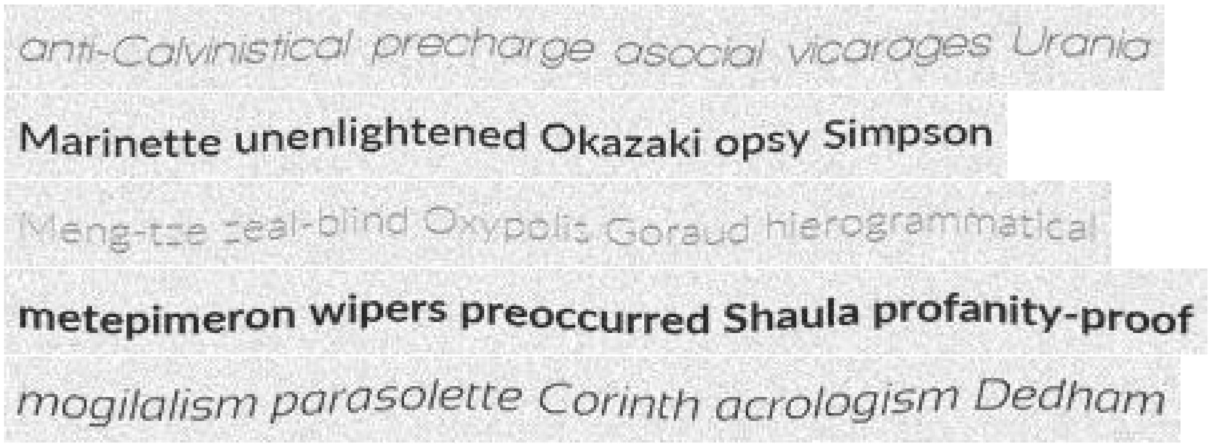
3.3 Text distorsions

For those familiar with the process of training a machine learning model, you often have to deal with overfitting, which is when the model gets too good at predicting the samples in the training data and stops generalizing to unseen examples. One trick to prevent this is by adding the distorsion to the data.

While TRDG does not dwelve too deeply in augmentations, as many better and more complete libraries already take care of it, some operations are available for convenience through the `-d` argument which as 3 possible values:

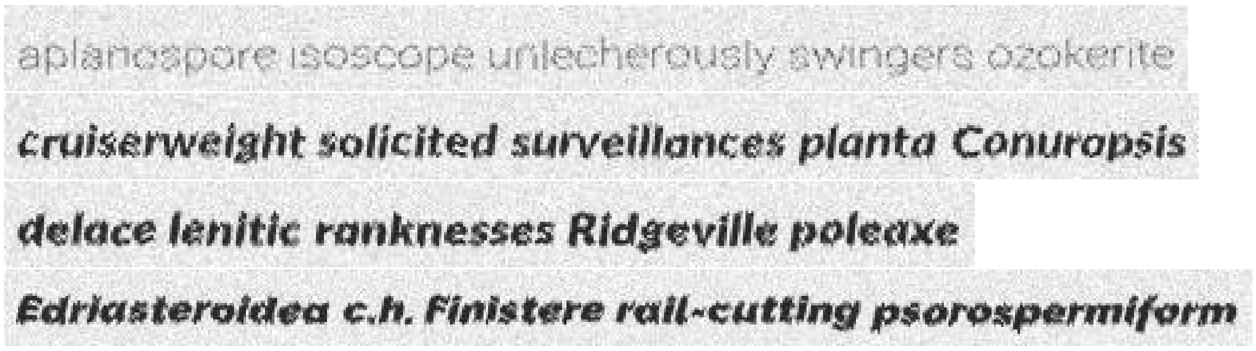
- 0: None
- 1: Sine wave
- 2: Cosine wave
- 3: Random

```
python3 run.py -c 5 -w 5 -d 1
```


anti-Calvinistical precharge asocial vicarages Urania
Marinette unenlightened Okazaki opsy Simpson
Meng-tze zeal-blind Oxypolis Goraud hierogrammatical
metepimeron wipers preoccurred Shaula profanity-proof
mogilalism parasolette Corinth acrologism Dedham

```
python3 run.py -c 5 -w 5 -d 3
```

addible bull-grip Ardsley after-grass persistive


aplanospore isoscope unlecherously swingers ozokerite
cruiserweight solicited surveillances planta Conuropsis
delace lenitic ranknesses Ridgeville poleaxe
Edriasteroidea c.h. Finistere rail-cutting psorospermiform

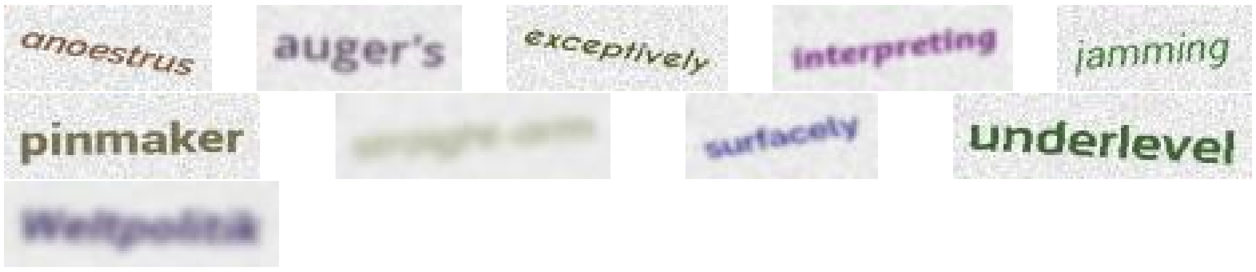
3.4 A more advanced use case

Text in the real world is not always black, and most importantly, text in the real world is almost never straight. What if we want to emulate that?

```
python3 run.py -c 10 -k 15 -rk -bl 0.5 -rbl -tc '#000000,#888888'
```

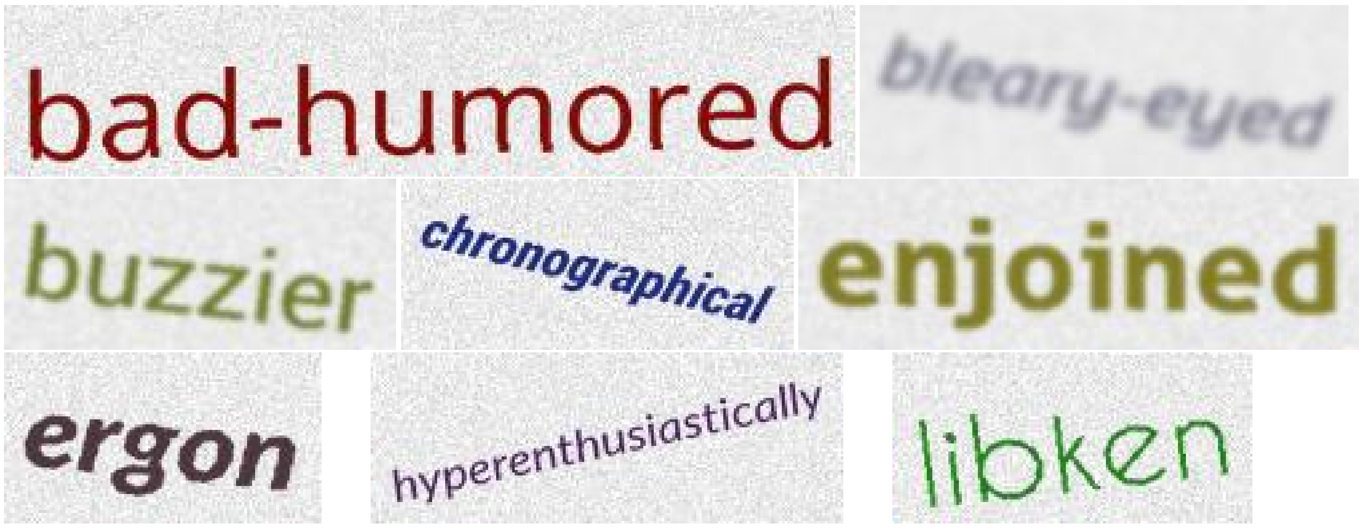
Which can be translated to: generate 10 examples with a skewing angle between -15 and 15 with an added gaussian blur between 0 and 0.1. Finally, the text color should be picked randomly between black and gray (including all the colors inbetween).

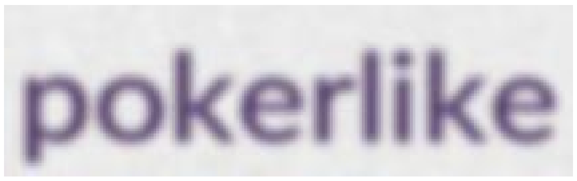
Sure enough, the output is much more colourful!



The default resolution might be too small to your taste (and I agree). By default the output is 32 pixels high because it's the height used by most text recognition papers. Now you can change that with `-f 64`.

```
python3 run.py -c 10 -k 15 -rk -bl 0.5 -rbl -tc '#000000,#888888' -f 64
```

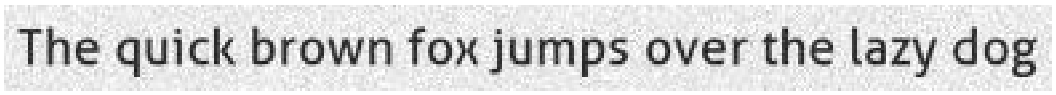




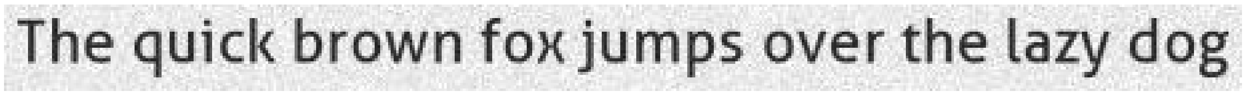
3.5 Manipulating margins

TRDG allows you to control margins around the text using two parameters, `--margins`, `--fit`. The first one controls margins, in pretty much the same way the CSS property `margin` does.

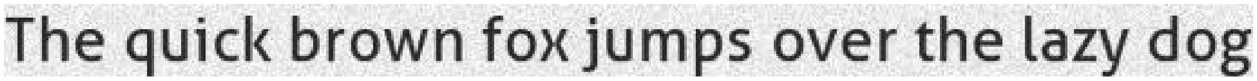
This is the result with no fit and the default (5, 5, 5, 5) margins: `python3 run.py -c 1 -i texts/test.txt`



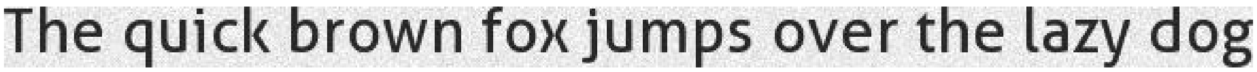
Now we can add `--fit` to apply a tight crop around the rendered text. This changes the size by removing the added space for accents: `python3 run.py -c 1 -i texts/test.txt --fit`



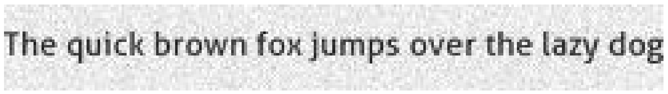
Margins are applied the generated text, so even with `0,0,0,0`, if you don't use `--fit` you will get an apparence of margins: `python3 run.py -c 1 -i texts/test.txt --margins 0,0,0,0`



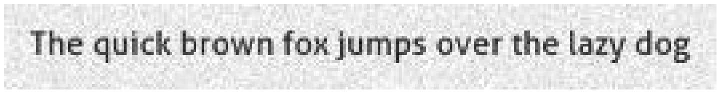
Now if you add `--fit`, you get an absolutely no margins: `python3 run.py -c 1 -i texts/test.txt --margins 0,0,0,0 --fit`



Margin values are comma separated `top,left,bottom,right`, so `--margins 10,0,10,0` will return vertical margins with tight cropping vertically.



And finally, with all margins: `python3 run.py -c 1 -i texts/test.txt --margins 10,10,10,10 --fit`



CHAPTER 4

Module

TRDG is also a module that can be included in your favorite training pipeline. The easiest way to use it, is to import a generator.

```
from trdg.generators import GeneratorFromStrings

generator = GeneratorFromStrings(['Test1', 'Test2', 'Test3'])

for img in generator:
    # Do something with the pillow image here.
```

The basic one is `GeneratorFromStrings` which, as its name indicates, will take a list of strings, and generate an image and label pair.

If you want to avoid having to maintain dictionaries, you can use `GeneratorFromDicts` which will use the bundled ones, `GeneratorFromRandom` which generates random strings, and `GeneratorFromWikipedia` which picks random article from Wikipedia as its source for strings.

Here are examples for each of those, respectively:

```
from trdg.generators import (
    GeneratorFromDicts,
    GeneratorFromRandom,
    GeneratorFromWikipedia,
)

generator_from_dicts = GeneratorFromDicts()
generator_from_random = GeneratorFromRandom()
generator_from_wikipedia = GeneratorFromWikipedia()

for img, lbl in generator_from_dicts:
    # Do something with the pillow image here.
```

The generators will not raise StopIteration, they will keep generating images until you break out of the loop. Set a non-negative value for `count` if that's an issue

CHAPTER 5

Reference

Coming soon

5.1 DataGenerator

5.2 BackgroundGenerator

5.3 ComputerTextGenerator

5.4 DistorsionGenerator

5.5 HandwrittenTextGenerator

5.6 StringGenerator