keras$_{o}crDocumentation$

Fausto Morales

Nov 26, 2020

CONTENTS:

`keras-ocr` provides out-of-the-box OCR models and an end-to-end training pipeline to build new OCR models. Please see the *examples* for more information.

CHAPTER

ONE

INSTALLATION

`keras-ocr` supports Python >= 3.6 and TensorFlow >= 2.0.0.

```
# To install from master
pip install git+https://github.com/faustomorales/keras-ocr.git#egg=keras-ocr

# To install from PyPi
pip install keras-ocr
```

CHAPTER

TWO

TROUBLESHOOTING

- *This package is installing* `opencv-python-headless` *but I would prefer a different* `opencv` *flavor.* This is due to aleju/imgaug#473. You can uninstall the unwanted OpenCV flavor after installing `keras-ocr`. We apologize for the inconvenience.

2.1 Examples

2.1.1 Using pretrained models

The below example shows how to use the pretrained models.

```
import matplotlib.pyplot as plt

import keras_ocr

# keras-ocr will automatically download pretrained
# weights for the detector and recognizer.
pipeline = keras_ocr.pipeline.Pipeline()

# Get a set of three example images
images = [
    keras_ocr.tools.read(url) for url in [
        'https://upload.wikimedia.org/wikipedia/commons/b/bd/Army_Reserves_
↪Recruitment_Banner_MOD_45156284.jpg',
        'https://upload.wikimedia.org/wikipedia/commons/e/e8/FseeG2QeLXo.jpg',
        'https://upload.wikimedia.org/wikipedia/commons/b/b4/EUBanana-500x112.jpg'
    ]
]

# Each list of predictions in prediction_groups is a list of
# (word, box) tuples.
prediction_groups = pipeline.recognize(images)

# Plot the predictions
fig, axs = plt.subplots(nrows=len(images), figsize=(20, 20))
for ax, image, predictions in zip(axs, images, prediction_groups):
    keras_ocr.tools.drawAnnotations(image=image, predictions=predictions, ax=ax)
```

2.1.2 Complete end-to-end training

You may wish to train your own end-to-end OCR pipeline. Here's an example for how you might do it. Note that the image generator has many options not documented here (such as adding backgrounds and image augmentation). Check the documentation for the *keras_ocr.tools.get_image_generator* function for more details.

Please note that, right now, we use a very simple training mechanism for the text detector which seems to work but does not match the method used in the original implementation.

An interactive version of this example on Google Colab is provided here.

Generating synthetic data

First, we define the alphabet that encompasses all characters we want our model to be able to detect and recognize. Below we designate our alphabet as the numbers 0-9, upper- and lower-case letters, and a few puncuation marks. For the recognizer, we will actually only predict lowercase letters because we know some fonts print lower- and upper-case characters with the same glyph.

In order to train on synthetic data, we require a set of fonts and backgrounds. `keras-ocr` includes a set of both of these which have been downloaded from Google Fonts and Wikimedia. The code to generate both of these sets is

available in the repository under `scripts/create_fonts_and_backgrounds.py`.

The fonts cover different languages which may have non-overlapping characters. `keras-ocr` supplies a function (`font_supports_alphabet`) to verify that a font includes the characters in an alphabet. We filter to only these fonts. We also exclude any fonts that are marked as *thin* in the filename because those tend to be difficult to render in a legible manner.

The backgrounds folder contains about just over 1,000 image backgrounds.

```
import zipfile
import datetime
import string
import math
import os

import tqdm
import matplotlib.pyplot as plt
import tensorflow as tf
import sklearn.model_selection

import keras_ocr

assert tf.test.is_gpu_available(), 'No GPU is available.'

data_dir = '.'
alphabet = string.digits + string.ascii_letters + '!?. '
recognizer_alphabet = ''.join(sorted(set(alphabet.lower())))
fonts = keras_ocr.data_generation.get_fonts(
    alphabet=alphabet,
    cache_dir=data_dir
)
backgrounds = keras_ocr.data_generation.get_backgrounds(cache_dir=data_dir)
```

With a set of fonts, backgrounds, and alphabet, we now build our data generators.

In order to create images, we need random strings. `keras-ocr` has a simple method for this for English, but anything that generates strings of characters in your selected alphabet will do!

The image generator generates *(image, lines)* tuples where *image* is a HxWx3 image and *lines* is a list of lines of text in the image where each line is itself a list of tuples of the form `((x1, y1), (x2, y2), (x3, y3), (x4, y4), c)`. *c* is the character in the line and `(x1, y1), (x2, y2), (x3, y3), (x4, y4)` define the bounding coordinates in clockwise order starting from the top left. You can replace this with your own generator, just be sure to match that function signature.

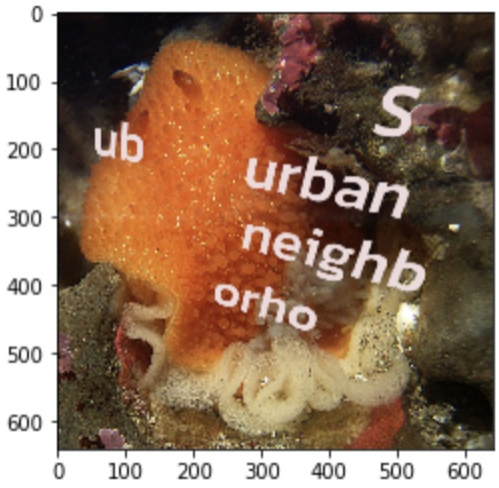
We split our generators into train, validation, and test by separating the fonts and backgrounds used in each.

```
text_generator = keras_ocr.data_generation.get_text_generator(alphabet=alphabet)
print('The first generated text is:', next(text_generator))

def get_train_val_test_split(arr):
    train, valtest = sklearn.model_selection.train_test_split(arr, train_size=0.8,␣
↪random_state=42)
    val, test = sklearn.model_selection.train_test_split(valtest, train_size=0.5,␣
↪random_state=42)
    return train, val, test

background_splits = get_train_val_test_split(backgrounds)
font_splits = get_train_val_test_split(fonts)
```

(continues on next page)

(continued from previous page)

```
image_generators = [
    keras_ocr.data_generation.get_image_generator(
        height=640,
        width=640,
        text_generator=text_generator,
        font_groups={
            alphabet: current_fonts
        },
        backgrounds=current_backgrounds,
        font_size=(60, 120),
        margin=50,
        rotationX=(-0.05, 0.05),
        rotationY=(-0.05, 0.05),
        rotationZ=(-15, 15)
    )  for current_fonts, current_backgrounds in zip(
        font_splits,
        background_splits
    )
]

# See what the first validation image looks like.
image, lines = next(image_generators[1])
text = keras_ocr.data_generation.convert_lines_to_paragraph(lines)
print('The first generated validation image (below) contains:', text)
plt.imshow(image)
```



Build base detector and recognizer models

Here we build our detector and recognizer models. For both, we'll start with pretrained models. Note that for the recognizer, we freeze the weights in the backbone (all the layers except for the final classification layer).

```
detector = keras_ocr.detection.Detector(weights='clovaai_general')
recognizer = keras_ocr.recognition.Recognizer(
    alphabet=recognizer_alphabet,
    weights='kurapan'
)
recognizer.compile()
```

(continues on next page)

(continued from previous page)

```
for layer in recognizer.backbone.layers:
    layer.trainable = False
```

Train the detector

We are now ready to train our text detector. Below we use some simple defaults.

- Run training until we have no improvement on the validation set for 5 epochs.
- Save the best weights.
- For each epoch, iterate over all backgrounds one time.

The `detector` object has a `get_batch_generator` method which converts the `image_generator` (which returns images and associated annotations) into a `batch_generator` that returns `X, y` pairs for training with `fit_generator`.

If training on Colab and it assigns you a K80, you can only use batch size 1. But if you get a T4 or P100, you can use larger batch sizes.

```
detector_batch_size = 1
detector_basepath = os.path.join(data_dir, f'detector_{datetime.datetime.now().
↪isoformat()}')
detection_train_generator, detection_val_generator, detection_test_generator = [
    detector.get_batch_generator(
        image_generator=image_generator,
        batch_size=detector_batch_size
    ) for image_generator in image_generators
]
detector.model.fit_generator(
    generator=detection_train_generator,
    steps_per_epoch=math.ceil(len(background_splits[0]) / detector_batch_size),
    epochs=1000,
    workers=0,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(restore_best_weights=True, patience=5),
        tf.keras.callbacks.CSVLogger(f'{detector_basepath}.csv'),
        tf.keras.callbacks.ModelCheckpoint(filepath=f'{detector_basepath}.h5')
    ],
    validation_data=detection_val_generator,
    validation_steps=math.ceil(len(background_splits[1]) / detector_batch_size)
)
```

Train the recognizer

After training the text detector, we train the recognizer. Note that the recognizer expects images to already be cropped to single lines of text. `keras-ocr` provides a convenience method for converting our existing generator into a single-line generator. So we perform that conversion.

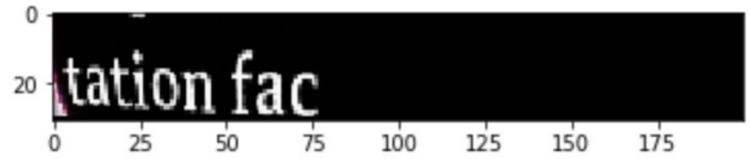
```
max_length = 10
recognition_image_generators = [
    keras_ocr.data_generation.convert_image_generator_to_recognizer_input(
        image_generator=image_generator,
        max_string_length=min(recognizer.training_model.input_shape[1][1], max_
↪length),
```

(continues on next page)

(continued from previous page)

```
        target_width=recognizer.model.input_shape[2],
        target_height=recognizer.model.input_shape[1],
        margin=1
    ) for image_generator in image_generators
]

# See what the first validation image for recognition training looks like.
image, text = next(recognition_image_generators[1])
print('This image contains:', text)
plt.imshow(image)
```

```
This image contains: tation fac
```



Just like the `detector`, the `recognizer` has a method for converting the image generator into a `batch_generator` that Keras' `fit_generator` can use.

We use the same callbacks for early stopping and logging as before.

```
recognition_batch_size = 8
recognizer_basepath = os.path.join(data_dir, f'recognizer_{datetime.datetime.now().
↪isoformat()}')
recognition_train_generator, recognition_val_generator, recogntion_test_generator = [
    recognizer.get_batch_generator(
    image_generator=image_generator,
    batch_size=recognition_batch_size,
    lowercase=True
    ) for image_generator in recognition_image_generators
]
recognizer.training_model.fit_generator(
    generator=recognition_train_generator,
    epochs=1000,
    steps_per_epoch=math.ceil(len(background_splits[0]) / recognition_batch_size),
    callbacks=[
        tf.keras.callbacks.EarlyStopping(restore_best_weights=True, patience=25),
        tf.keras.callbacks.CSVLogger(f'{recognizer_basepath}.csv', append=True),
        tf.keras.callbacks.ModelCheckpoint(filepath=f'{recognizer_basepath}.h5')
    ],
    validation_data=recognition_val_generator,
    validation_steps=math.ceil(len(background_splits[1]) / recognition_batch_size),
    workers=0
)
```

Use the models for inference

Once training is done, you can use `recognize` to extract text.

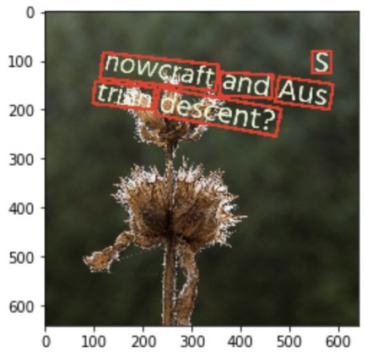
```
pipeline = keras_ocr.pipeline.Pipeline(detector=detector, recognizer=recognizer)
image, lines = next(image_generators[0])
predictions = pipeline.recognize(images=[image])[0]
```

(continues on next page)

(continued from previous page)

```
drawn = keras_ocr.tools.drawBoxes(
    image=image, boxes=predictions, boxes_format='predictions'
)
print(
    'Actual:', '\n'.join([' '.join([character for _, character in line]) for line in␣
↪lines]),
    'Predicted:', [text for text, box in predictions]
)
plt.imshow(drawn)
```

```
Snowcraft and Austrian descent? ['s', 'nowcraft', 'and', 'aus', 'trian', 'descent']
```



2.1.3 Fine-tuning the detector

This example shows how to fine-tune the recognizer using an existing dataset. In this case, we will use the text segmentation dataset from ICDAR 2013, available from https://rrc.cvc.uab.es/?ch=1&com=downloads.

First, we download our dataset. `keras-ocr` provides a convenience function for this, which you are welcome to examine to understand how the dataset is downloaded and parsed.

An interactive version of this example on Google Colab is provided here.

```
data_dir = '.'

import os
import math
import imgaug
import numpy as np
import matplotlib.pyplot as plt
import sklearn.model_selection
import tensorflow as tf

import keras_ocr

dataset = keras_ocr.datasets.get_icdar_2013_detector_dataset(
    cache_dir='.',
    skip_illegible=False
)
```

Now we split the dataset into training and validation.

```
train, validation = sklearn.model_selection.train_test_split(
    dataset, train_size=0.8, random_state=42
```

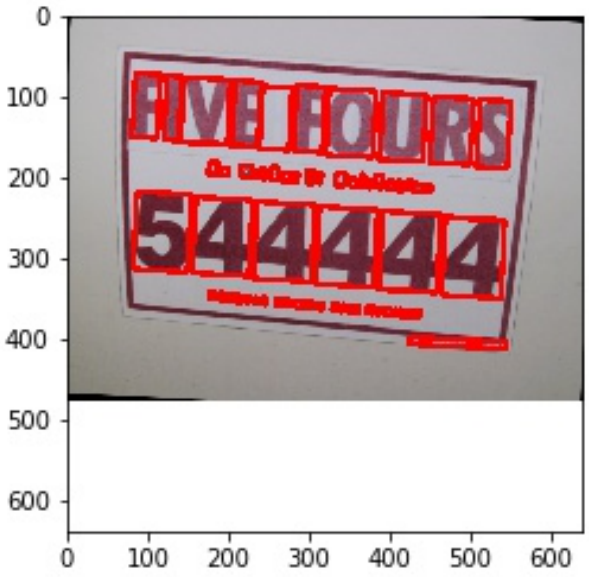
(continues on next page)

(continued from previous page)

```
)
augmenter = imgaug.augmenters.Sequential([
    imgaug.augmenters.Affine(
    scale=(1.0, 1.2),
    rotate=(-5, 5)
    ),
    imgaug.augmenters.GaussianBlur(sigma=(0, 0.5)),
    imgaug.augmenters.Multiply((0.8, 1.2), per_channel=0.2)
])
generator_kwargs = {'width': 640, 'height': 640}
training_image_generator = keras_ocr.datasets.get_detector_image_generator(
    labels=train,
    augmenter=augmenter,
    **generator_kwargs
)
validation_image_generator = keras_ocr.datasets.get_detector_image_generator(
    labels=validation,
    **generator_kwargs
)
```

We can visualize what the samples look like pretty easily.

```
image, lines, confidence = next(training_image_generator)
canvas = keras_ocr.tools.drawBoxes(image=image, boxes=lines, boxes_format='lines')
plt.imshow(canvas)
```



Now we can build the detector and train it.

```
detector = keras_ocr.detection.Detector()

batch_size = 1
training_generator, validation_generator = [
    detector.get_batch_generator(
```

(continues on next page)

(continued from previous page)

```
        image_generator=image_generator, batch_size=batch_size
    ) for image_generator in
    [training_image_generator, validation_image_generator]
]
detector.model.fit_generator(
    generator=training_generator,
    steps_per_epoch=math.ceil(len(train) / batch_size),
    epochs=1000,
    workers=0,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(restore_best_weights=True, patience=5),
        tf.keras.callbacks.CSVLogger(os.path.join(data_dir, 'detector_icdar2013.csv
↪')),
        tf.keras.callbacks.ModelCheckpoint(filepath=os.path.join(data_dir, 'detector_
↪icdar2013.h5'))
    ],
    validation_data=validation_generator,
    validation_steps=math.ceil(len(validation) / batch_size)
)
```

Weights can be loaded into the model attribute of the detector. This is how you can reuse the weights later.

```
detector.model.load_weights(os.path.join(data_dir, 'detector_icdar2013.h5'))
```

2.1.4 Fine-tuning the recognizer

This example shows how to fine-tune the recognizer using an existing dataset. In this case, we will use the "Born Digital" dataset from https://rrc.cvc.uab.es/?ch=1&com=downloads

First, we download our dataset. Below we get both the training and test datasets, but we only use the training dataset. The training dataset consists of a single folder containing images, each of which has a single word in it.

An interactive version of this example on Google Colab is provided here.

```
import random
import string
import math
import itertools
import os

import numpy as np
import imgaug
import matplotlib.pyplot as plt
import tensorflow as tf
import sklearn.model_selection

import keras_ocr

assert tf.test.is_gpu_available()

train_labels = keras_ocr.datasets.get_born_digital_recognizer_dataset(
    split='train',
    cache_dir='.'
)
test_labels = keras_ocr.datasets.get_born_digital_recognizer_dataset(
```

(continues on next page)

(continued from previous page)

```
    split='test',
    cache_dir='.'
)
train_labels = [(filepath, box, word.lower()) for filepath, box, word in train_labels]
test_labels = [(filepath, box, word.lower()) for filepath, box, word in train_labels]
```

We next build our recognizer, using the default options to get a pretrained model.

```
recognizer = keras_ocr.recognition.Recognizer()
recognizer.compile()
```

We need to convert our dataset into the format that `keras-ocr` requires. To do that, we have the following, which includes support for an augmenter to generate synthetically altered samples. Note that this code is set up to skip any characters that are not in the recognizer alphabet and that all labels are first converted to lowercase.

```
batch_size = 8
augmenter = imgaug.augmenters.Sequential([
    imgaug.augmenters.GammaContrast(gamma=(0.25, 3.0)),
])

train_labels, validation_labels = sklearn.model_selection.train_test_split(train_
↪labels, test_size=0.2, random_state=42)
(training_image_gen, training_steps), (validation_image_gen, validation_steps) = [
    (
        keras_ocr.datasets.get_recognizer_image_generator(
            labels=labels,
            height=recognizer.model.input_shape[1],
            width=recognizer.model.input_shape[2],
            alphabet=recognizer.alphabet,
            augmenter=augmenter
        ),
        len(labels) // batch_size
    ) for labels, augmenter in [(train_labels, augmenter), (validation_labels, None)]
]
training_gen, validation_gen = [
    recognizer.get_batch_generator(
        image_generator=image_generator,
        batch_size=batch_size
    )
    for image_generator in [training_image_gen, validation_image_gen]
]
```

As a sanity check, we show one of the samples.

```
image, text = next(training_image_gen)
print('text:', text)
plt.imshow(image)
```

```
text: collaboration
```

Now we can run training.

```
callbacks = [
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=10,␣
↪restore_best_weights=False),
    tf.keras.callbacks.ModelCheckpoint('recognizer_borndigital.h5', monitor='val_loss
↪', save_best_only=True),
    tf.keras.callbacks.CSVLogger('recognizer_borndigital.csv')
]
recognizer.training_model.fit_generator(
    generator=training_gen,
    steps_per_epoch=training_steps,
    validation_steps=validation_steps,
    validation_data=validation_gen,
    callbacks=callbacks,
    epochs=1000,
)
```

Finally, run inference on a test sample.

```
image_filepath, _, actual = test_labels[1]
predicted = recognizer.recognize(image_filepath)
print(f'Predicted: {predicted}, Actual: {actual}')
_ = plt.imshow(keras_ocr.tools.read(image_filepath))
```

```
Predicted: developer, Actual: Developer
```

You can load weights back into a model using *recognizer.model.load_weights()*.

```
recognizer.model.load_weights('recognizer_borndigital.h5')
```

2.2 API

2.2.1 Core Detector and Recognizer

The detector and recognizer classes are the core of the package. They provide wrappers for the underlying Keras models.

class keras_ocr.detection.**Detector**(*weights='clovaai_general'*, *load_from_torch=False*, *optimizer='adam'*, *backbone_name='vgg'*)

A text detector using the CRAFT architecture.

Parameters

- **weights** – The weights to use for the model. Currently, only *clovaai_general* is supported.
- **load_from_torch** – Whether to load the weights from the original PyTorch weights.
- **optimizer** – The optimizer to use for training the model.
- **backbone_name** – The backbone to use. Currently, only 'vgg' is supported.

detect(*images*, *detection_threshold=0.7*, *text_threshold=0.4*, *link_threshold=0.4*, *size_threshold=10*, ***kwargs*)

Recognize the text in a set of images.

Parameters

- **images** – Can be a list of numpy arrays of shape HxWx3 or a list of filepaths.
- **link_threshold** – This is the same as *text_threshold*, but is applied to the link map instead of the text map.
- **detection_threshold** – We want to avoid including boxes that may have represented large regions of low confidence text predictions. To do this, we do a final check for each word box to make sure the maximum confidence value exceeds some detection threshold. This is the threshold used for this check.
- **text_threshold** – When the text map is processed, it is converted from confidence (float from zero to one) values to classification (0 for not text, 1 for text) using binary thresholding. The threshold value determines the breakpoint at which a value is converted to a 1 or a 0. For example, if the threshold is 0.4 and a value for particular point on the text map is 0.5, that value gets converted to a 1. The higher this value is, the less likely it is that characters will be merged together into a single word. The lower this value is, the more likely it is that non-text will be detected. Therein lies the balance.
- **size_threshold** – The minimum area for a word.

get_batch_generator(*image_generator*, *batch_size=8*, *heatmap_size=512*, *heatmap_distance_ratio=1.5*)

Get a generator of X, y batches to train the detector.

Parameters

- **image_generator** – A generator with the same signature as keras_ocr.tools.get_image_generator. Optionally, a third entry in the tuple (beyond image and lines) can be provided which will be interpreted as the sample weight.
- **batch_size** – The size of batches to generate.
- **heatmap_size** – The size of the heatmap to pass to get_gaussian_heatmap
- **heatmap_distance_ratio** – The distance ratio to pass to get_gaussian_heatmap. The larger the value, the more tightly concentrated the heatmap becomes.

class keras_ocr.recognition.**Recognizer**(*alphabet=None*, *weights='kurapan'*, *build_params=None*)

A text detector using the CRNN architecture.

Parameters

- **alphabet** – The alphabet the model should recognize.
- **build_params** – A dictionary of build parameters for the model. See *keras_ocr.recognition.build_model* for details.
- **weights** – The starting weight configuration for the model.
- **include_top** – Whether to include the final classification layer in the model (set to False to use a custom alphabet).

compile(**args*, ***kwargs*)

Compile the training model.

get_batch_generator(*image_generator*, *batch_size=8*, *lowercase=False*)

Generate batches of training data from an image generator. The generator should yield tuples of (image, sentence) where image contains a single line of text and sentence is a string representing the contents of the image. If a sample weight is desired, it can be provided as a third entry in the tuple, making each tuple an (image, sentence, weight) tuple.

Parameters

- **image_generator** – An image / sentence tuple generator. The images should be in color even if the OCR is setup to handle grayscale as they will be converted here.
- **batch_size** – How many images to generate at a time.
- **lowercase** – Whether to convert all characters to lowercase before encoding.

recognize(*image*)
Recognize text from a single image.

Parameters **image** – A pre-cropped image containing characters

recognize_from_boxes(*images*, *box_groups*, ***kwargs*)
Recognize text from images using lists of bounding boxes.

Parameters

- **images** – A list of input images, supplied as numpy arrays with shape (H, W, 3).
- **boxes** – A list of groups of boxes, one for each image

Return type List[str]

2.2.2 Data Generation

The data_generation module contains the functions for generating synthetic data.

keras_ocr.data_generation.**compute_transformed_contour**(*width*, *height*, *fontsize*, *M*, *contour*, *minarea=0.5*)
Compute the permitted drawing contour on a padded canvas for an image of a given size. We assume the canvas is padded with one full image width and height on left and right, top and bottom respectively.

Parameters

- **width** – Width of image
- **height** – Height of image
- **fontsize** – Size of characters
- **M** – The transformation matrix
- **contour** – The contour to which we are limited inside the rectangle of size width / height
- **minarea** – The minimum area required for a character slot to qualify as being visible, expressed as a fraction of the untransformed fontsize x fontsize slot.

keras_ocr.data_generation.**convert_image_generator_to_recognizer_input**(*image_generator*, *max_string_length*, *target_width*, *target_height*, *margin=0*)
Convert an image generator created by get_image_generator to (image, sentence) tuples for training a recognizer.

Parameters

- **image_generator** – An image generator created by get_image_generator
- **max_string_length** – The maximum string length to allow

- **target_width** – The width to warp lines into
- **target_height** – The height to warp lines into
- **margin** – The margin to apply around a single line.

keras_ocr.data_generation.**convert_lines_to_paragraph**(*lines*)
Convert a series of lines, each consisting of (box, character) tuples, into a multi-line string.

keras_ocr.data_generation.**draw_text_image**(*text*, *fontsize*, *height*, *width*, *fonts*, *use_ligatures=False*, *thetaX=0*, *thetaY=0*, *thetaZ=0*, *color=(0, 0, 0)*, *permitted_contour=None*, *draw_contour=False*)
Get a transparent image containing text.

Parameters

- **text** – The text to draw on the image
- **fontsize** – The size of text to show.
- **height** – The height of the output image
- **width** – The width of the output image
- **fonts** – A dictionary of {subalphabet: paths_to_font}
- **thetaX** – Rotation about the X axis
- **thetaY** – Rotation about the Y axis
- **thetaZ** – Rotation about the Z axis
- **color** – The color of drawn text
- **permitted_contour** – A contour defining which part of the image we can put text. If None, the entire canvas is permitted for text.
- **use_ligatures** – Whether to render ligatures. If True, ligatures are always used (with an initial check for support which sometimes yields false positives). If False, ligatures are never used.

Returns An (image, lines) tuple where image is the transparent text image and lines is a list of lines where each line itself is a list of (box, character) tuples and box is an array of points with shape (4, 2) providing the coordinates of the character box in clockwise order starting from the top left.

keras_ocr.data_generation.**font_supports_alphabet**(*filepath*, *alphabet*)
Verify that a font contains a specific set of characters.

Parameters

- **filepath** – Path to fsontfile
- **alphabet** – A string of characters to check for.

keras_ocr.data_generation.**get_backgrounds**(*cache_dir=None*)
Download a set of pre-reviewed backgrounds.

Parameters **cache_dir** – Where to save the dataset. By default, data will be saved to ~/.keras-ocr.

Returns A list of background filepaths.

keras_ocr.data_generation.**get_fonts**(*cache_dir=None*, *alphabet='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234*
exclude_smallcaps=False)
Download a set of pre-reviewed fonts.

Parameters

- **cache_dir** – Where to save the dataset. By default, data will be saved to ~/.keras-ocr.
- **alphabet** – An alphabet which we will use to exclude fonts that are missing relevant characters. By default, this is set to *string.ascii_letters* + *string.digits*.
- **exclude_smallcaps** – If True, fonts that are known to use the same glyph for lowercase and uppercase characters are excluded.

Returns A list of font filepaths.

keras_ocr.data_generation.**get_image_generator**(*height*, *width*, *font_groups*, *text_generator*, *font_size=18*, *backgrounds=None*, *background_crop_mode='crop'*, *rotationX=0*, *rotationY=0*, *rotationZ=0*, *margin=0*, *use_ligatures=False*, *augmenter=None*, *draw_contour=False*, *draw_contour_text=False*)

Create a generator for images containing text.

Parameters

- **height** – The height of the generated image
- **width** – The width of the generated image.
- **font_groups** – A dict mapping of { subalphabet: [path_to_font1, path_to_font2] }.
- **text_generator** – See get_text_generator
- **font_size** – The font size to use. Alternative, supply a tuple and the font size will be randomly selected between the two values.
- **backgrounds** – A list of paths to image backgrounds or actual images as numpy arrays with channels in RGB order.
- **background_crop_mode** – One of letterbox or crop, indicates how backgrounds will be resized to fit on the canvas.
- **rotationX** – The X-axis text rotation to use. Alternative, supply a tuple and the rotation will be randomly selected between the two values.
- **rotationY** – The Y-axis text rotation to use. Alternative, supply a tuple and the rotation will be randomly selected between the two values.
- **rotationZ** – The Z-axis text rotation to use. Alternative, supply a tuple and the rotation will be randomly selected between the two values.
- **margin** – The minimum margin around the edge of the image.
- **use_ligatures** – Whether to render ligatures (see *draw_text_image*)
- **augmenter** – An image augmenter to be applied to backgrounds
- **draw_contour** – Draw the permitted contour onto images (debugging only)
- **draw_contour_text** – Draw the permitted contour inside the text drawing function.

Yields Tuples of (image, lines) where image is the transparent text image and lines is a list of lines where each line itself is a list of (box, character) tuples and box is an array of points with shape (4, 2) providing the coordinates of the character box in clockwise order starting from the top left.

keras_ocr.data_generation.**get_maximum_uniform_contour**(*image*, *fontsize*, *margin=0*)
Get the largest possible contour of light or dark area in an image.

Parameters

- **image** – The image in which to find a contiguous area.
- **fontsize** – The fontsize for text. Will be used for blurring and for determining useful areas.
- **margin** – The minimum margin required around the image.

Returns A (contour, isDark) tuple. If no contour is found, both entries will be None.

keras_ocr.data_generation.**get_rotation_matrix**(*width*, *height*, *thetaX=0*, *thetaY=0*, *thetaZ=0*)
Provide a rotation matrix about the center of a rectangle with a given width and height.

Parameters

- **width** – The width of the rectangle
- **height** – The height of the rectangle
- **thetaX** – Rotation about the X axis
- **thetaY** – Rotation about the Y axis
- **thetaZ** – Rotation about the Z axis

Returns A 3x3 transformation matrix

keras_ocr.data_generation.**get_text_generator**(*alphabet=None*, *lowercase=False*, *max_string_length=None*)
Generates strings of sentences using only the letters in alphabet.

Parameters

- **alphabet** – The alphabet of permitted characters
- **lowercase** – Whether to convert all strings to lowercase.
- **max_string_length** – The maximum length of the string

2.2.3 Tools

The tools module primarily contains convenience functions for reading images and downloading data.

keras_ocr.tools.**adjust_boxes**(*boxes*, *boxes_format='boxes'*, *scale=1*)
Adjust boxes using a given scale and offset.

Parameters

- **boxes** – The boxes to adjust
- **boxes_format** – The format for the boxes. See the *drawBoxes* function for an explanation on the options.
- **scale** – The scale to apply

keras_ocr.tools.**augment**(*boxes*, *augmenter*, *image=None*, *boxes_format='boxes'*, *image_shape=None*, *area_threshold=0.5*, *min_area=None*)
Augment an image and associated boxes together.

Parameters

- **image** – The image which we wish to apply the augmentation.

- **boxes** – The boxes that will be augmented together with the image
- **boxes_format** – The format for the boxes. See the *drawBoxes* function for an explanation on the options.
- **image_shape** – The shape of the input image if no image will be provided.
- **area_threshold** – Fraction of bounding box that we require to be in augmented image to include it.
- **min_area** – The minimum area for a character to be included.

keras_ocr.tools.**combine_line**(*line*)

Combine a set of boxes in a line into a single bounding box.

Parameters **line** – A list of (box, character) entries

Returns A (box, text) tuple

keras_ocr.tools.**download_and_verify**(*url*, *sha256=None*, *cache_dir=None*, *verbose=True*, *filename=None*)

Download a file to a cache directory and verify it with a sha256 hash.

Parameters

- **url** – The file to download
- **sha256** – The sha256 hash to check. If the file already exists and the hash matches, we don't download it again.
- **cache_dir** – The directory in which to cache the file. The default is *~/.keras-ocr*.
- **verbose** – Whether to log progress
- **filename** – The filename to use for the file. By default, the filename is derived from the URL.

keras_ocr.tools.**drawAnnotations**(*image*, *predictions*, *ax=None*)

Draw text annotations onto image.

Parameters

- **image** – The image on which to draw
- **predictions** – The predictions as provided by *pipeline.recognize*.
- **ax** – A matplotlib axis on which to draw.

keras_ocr.tools.**drawBoxes**(*image*, *boxes*, *color=(255, 0, 0)*, *thickness=5*, *boxes_format='boxes'*)

Draw boxes onto an image.

Parameters

- **image** – The image on which to draw the boxes.
- **boxes** – The boxes to draw.
- **color** – The color for each box.
- **thickness** – The thickness for each box.
- **boxes_format** – The format used for providing the boxes. Options are "boxes" which indicates an array with shape(N, 4, 2) where N is the number of boxes and each box is a list of four points) as provided by *keras_ocr.detection.Detector.detect*, "lines" (a list of lines where each line itself is a list of (box, character) tuples) as provided by

keras_ocr.data_generation.get_image_generator, or "predictions" where boxes is by itself a list of (word, box) tuples as provided by *keras_ocr.pipeline.Pipeline.recognize* or *keras_ocr.recognition.Recognizer.recognize_from_boxes*.

keras_ocr.tools.**fit**(*image*, *width*, *height*, *cval=255*, *mode='letterbox'*, *return_scale=False*)

Obtain a new image, fit to the specified size.

Parameters

- **image** – The input image
- **width** – The new width
- **height** – The new height
- **cval** – The constant value to use to fill the remaining areas of the image
- **return_scale** – Whether to return the scale used for the image

Returns The new image

keras_ocr.tools.**fix_line**(*line*)

Given a list of (box, character) tuples, return a revised line with a consistent ordering of left-to-right or top-to-bottom, with each box provided with (top-left, top-right, bottom-right, bottom-left) ordering.

Returns A tuple that is the fixed line as well as a string indicating whether the line is horizontal or vertical.

keras_ocr.tools.**get_rotated_box**(*points*)

Obtain the parameters of a rotated box.

Returns The vertices of the rotated box in top-left, top-right, bottom-right, bottom-left order along with the angle of rotation about the bottom left corner.

Return type Tuple[Tuple[float, float], Tuple[float, float], Tuple[float, float], Tuple[float, float], float]

keras_ocr.tools.**get_rotated_width_height**(*box*)

Returns the width and height of a rotated rectangle

Parameters

- **box** – A list of four points starting in the top left
- **and moving clockwise.** (*corner*) –

keras_ocr.tools.**pad**(*image*, *width*, *height*, *cval=255*)

Pad an image to a desired size. Raises an exception if image is larger than desired size.

Parameters

- **image** – The input image
- **width** – The output width
- **height** – The output height
- **cval** – The value to use for filling the image.

keras_ocr.tools.**read**(*filepath_or_buffer*)

Read a file into an image object

Parameters **filepath_or_buffer** – The path to the file, a URL, or any object with a *read* method (such as *io.BytesIO*)

keras_ocr.tools.**read_and_fit**(*filepath_or_array*, *width*, *height*, *cval=255*, *mode='letterbox'*)

Read an image from disk and fit to the specified size.

Parameters

- **filepath** – The path to the image or numpy array of shape HxWx3
- **width** – The new width
- **height** – The new height
- **cval** – The constant value to use to fill the remaining areas of the image
- **mode** – The mode to pass to "fit" (crop or letterbox)

Returns The new image

keras_ocr.tools.**resize_image**(*image*, *max_scale*, *max_size*)

Obtain the optimal resized image subject to a maximum scale and maximum size.

Parameters

- **image** – The input image
- **max_scale** – The maximum scale to apply
- **max_size** – The maximum size to return

keras_ocr.tools.**sha256sum**(*filename*)

Compute the sha256 hash for a file.

keras_ocr.tools.**warpBox**(*image*, *box*, *target_height=None*, *target_width=None*, *margin=0*, *cval=None*, *return_transform=False*, *skip_rotate=False*)

Warp a boxed region in an image given by a set of four points into a rectangle with a specified width and height. Useful for taking crops of distorted or rotated text.

Parameters

- **image** – The image from which to take the box
- **box** – A list of four points starting in the top left corner and moving clockwise.
- **target_height** – The height of the output rectangle
- **target_width** – The width of the output rectangle
- **return_transform** – Whether to return the transformation matrix with the image.

2.2.4 Datasets

The datasets module contains functions for using data from public datasets. See the *fine-tuning detector* and *fine-tuning recognizer* examples.

keras_ocr.datasets.**get_born_digital_recognizer_dataset**(*split='train'*, *cache_dir=None*)

Get a list of (filepath, box, word) tuples from the BornDigital dataset. This dataset comes pre-cropped so *box* is always *None*.

Parameters

- **split** – Which split to get (train, test, or traintest)
- **cache_dir** – The directory in which to cache the file. The default is *~/.keras-ocr*.

Returns A recognition dataset as a list of (filepath, box, word) tuples

keras_ocr.datasets.**get_cocotext_recognizer_dataset**(*split='train'*, *cache_dir=None*, *limit=None*, *legible_only=False*, *english_only=False*, *return_raw_labels=False*)

Get a list of (filepath, box, word) tuples from the COCO-Text dataset.

Parameters

- **split** – Which split to get (train, val, or trainval)
- **limit** – Limit the number of files included in the download
- **cache_dir** – The directory in which to cache the file. The default is *~/.keras-ocr*.
- **return_raw_labels** – Whether to return the raw labels object

Returns A recognition dataset as a list of (filepath, box, word) tuples. If return_raw_labels is True, you will also get a (labels, images_dir) tuple containing the raw COCO data and the directory in which you can find the images.

keras_ocr.datasets.**get_detector_image_generator**(*labels*, *width*, *height*, *augmenter=None*, *area_threshold=0.5*, *focused=False*, *min_area=None*, *shuffle=True*)

Generated augmented (image, lines) tuples from a list of (filepath, lines, confidence) tuples. Confidence is not used right now but is included for a future release that uses semi-supervised data.

Parameters

- **labels** – A list of (image, lines, confience) tuples.
- **augmenter** – An augmenter to apply to the images.
- **width** – The width to use for output images
- **height** – The height to use for output images
- **area_threshold** – The area threshold to use to keep characters in augmented images.
- **min_area** – The minimum area for a character to be included.
- **focused** – Whether to pre-crop images to width/height containing a region containing text.
- **shuffle** – Whether to shuffle the data on each iteration.

keras_ocr.datasets.**get_icdar_2013_detector_dataset**(*cache_dir=None*, *skip_illegible=False*)

Get the ICDAR 2013 text segmentation dataset for detector training. Only the training set has the necessary annotations. For the test set, only segmentation maps are provided, which do not provide the necessary information for affinity scores.

Parameters

- **cache_dir** – The directory in which to store the data.
- **skip_illegible** – Whether to skip illegible characters.

Returns Lists of (image_path, lines, confidence) tuples. Confidence is always 1 for this dataset. We record confidence to allow for future support for weakly supervised cases.

keras_ocr.datasets.**get_icdar_2013_recognizer_dataset**(*cache_dir=None*)

Get a list of (filepath, box, word) tuples from the ICDAR 2013 dataset.

Parameters **cache_dir** – The directory in which to cache the file. The default is *~/.keras-ocr*.

Returns A recognition dataset as a list of (filepath, box, word) tuples

keras_ocr.datasets.**get_icdar_2019_semisupervised_dataset**(*cache_dir=None*)

EXPERIMENTAL. Get a semisupervised labeled version of the ICDAR 2019 dataset. Only images with Latin-only scripts are available at this time.

Parameters **cache_dir** – The cache directory to use.

keras_ocr.datasets.**get_recognizer_image_generator**(*labels*, *height*, *width*, *alphabet*, *augmenter=None*, *shuffle=True*)

Generate augmented (image, text) tuples from a list of (filepath, box, label) tuples.

Parameters

- **labels** – A list of (filepath, box, label) tuples
- **height** – The height of the images to return
- **width** – The width of the images to return
- **alphabet** – The alphabet which limits the characters returned
- **augmenter** – The augmenter to apply to images
- **shuffle** – Whether to shuffle the dataset on each iteration

2.2.5 Custom Objects

These are experimental objects to be used for custom operations with Keras.

PYTHON MODULE INDEX

k

INDEX

S

W