alexlib

Release 0.0.6

Alex Al-Saffar

Nov 23, 2020

CONTENTS:

CHAPTER

ONE

WHAT DOES IT OFFER?

- Extended *Path* class to make life easier.
- Extended *dict* class.
- Extended *list* class.

1.1 alexlib

1.1.1 alexlib package

1.1.1.1 Submodules

1.1.1.2 alexlib.toolbox module

A collection of classes extending the functionality of Python's builtins. email programmer@usa.com

class alexlib.toolbox.**Artist**(**args*, *ax=None*, *figname='Graph'*, *title=''*, *label='curve'*, *style='seaborn'*, *create_new_axes=False*, *figure_policy=<FigurePolicy.add_new: 'Create a new figure with same name but with added suffix'>*, *figsize=(7, 4)*, ***kwargs*)

Bases: *alexlib.toolbox.FigureManager*

accessorize(**args*, *legends=None*, *title=None*, ***kwargs*)

get_axes()

plot(**args*, ***kwargs*)

static styler(*plot_gen*)

suptitle(*title*)

visibility()

class alexlib.toolbox.**Base**

Bases: object

evalstr(*string_*, *expected='self'*)

classmethod from_saved(*path*)

get_attributes()

save(*path*)

this method goes with default *.npy* format (because it is generic). # P(path).parent.create()

save_json(*path*)
Use case: json is good for simple dicts, e.g. settings. Advantage: human-readable from file explorer.

save_mat(*path*)
for Matlab compatibility.

class alexlib.toolbox.**Browse**(*path*, *directory=True*)
Bases: object

class alexlib.toolbox.**Compression**
Bases: object

Provides consistent behaviour across all methods ... Both files and folders when compressed, default is being under the root of archive.

static compress_folder(*ip_path*, *op_path*, *arcname*, *format_='zip'*, ***kwargs*)
base_dir has to be relevant to op_path. If you want to compress a folder in Downloads/myfolder/compress_this Then, say that your rootdir is where you want the archive structure to include, then mention the folder you want to actually archive relatively to that root. **fromat_**: zip, tar, gztar, bztar, xztar

static gz(*file*)

static tar()

static ungz(*self*, *op_path=None*)

static untar(*self*, *fname=None*, *mode='r'*, ***kwargs*)

static unzip(*ip_path*, *op_path*, *fname=None*, ***kwargs*)

static zip_file(*ip_path*, *op_path*, *arcname*, ***kwargs*)
arcname determines the directory of the file being archived inside the archive. Defaults to same as original directory except for drive. When changed, it should still include the file name in its end. If arcname = filename without any path, then, it will be in the root of the archive.

class alexlib.toolbox.**Cycle**(*c*, *name=''*)
Bases: object

get()

get_index()

next()

previous()

sample(*size=1*)

set(*value*)

set_index(*index*)

class alexlib.toolbox.**DisplayData**(*x*)
Bases: object

static eng()

static set_display()

class alexlib.toolbox.**FigureManager**(*info_loc=None*, *figure_policy=<FigurePolicy.same: 'Grab the figure of the same name'>*)
Bases: object

Handles figures of matplotlib.

static activate_latex(*size=20*)
Setting up matplotlib

adjust_brightness(*event*)

animate()

annotate(*event*, *axis=None*, *data=None*)

change_cmap(*event*)

change_facecolor(*event*)

clear_axes()

static findobj(*fig_name*, *obj_name*)

static get_ax_size(*ax*)

get_fig(*figname=''*, *suffix=None*, ***kwargs*)

static get_fig_static(*figure_policy*, *figname=''*, *suffix=None*, ***kwargs*)

Parameters

- **figure_policy** –
- **figname** –
- **suffix** – only relevant if figure_policy is add_new
- **kwargs** –

Returns

static get_nrows_ncols(*num_plots*, *nrows=None*, *ncols=None*)

static grid(*ax*, *factor=5*, *x_or_y='both'*, *color='gray'*, *alpha1=0.5*, *alpha2=0.25*)

maximize_fig()

next(*event*)

pause_func(*event*)

previous(*event*)

process_key(*event*)

replay(*event*)

save(*event*)

static set_ax_size(*ax*, *w*, *h*)
w, h: width, height in inches

static set_ax_to_real_life_size(*ax*, *inch_per_unit=0.03937007874015748*)

static set_linestyles_and_markers_and_colors(*test=False*)

show_cursor(*event*)

show_help(*event*)

show_pix_val(*event*)

static show_pixels_values(*ax*)

show_ticks(*event*)

text_info(*event*)

toggle_annotate(*event*)

static toggle_ticks(*an_ax*, *state=None*)

transperent_fig()

static try_figure_size()

static update(*fig_name*, *obj_name*, *data=None*)
Fastest update ever. But, you need access to label name. Using this function external to the plotter. But inside the plotter you need to define labels to objects The other alternative is to do the update inside the plotter, but it will become very verbose.

Parameters

- **fig_name** –
- **obj_name** –
- **data** –

Returns

update_info_text(*message*)

static write(*txt*, *name='text'*, *size=8*, ***kwargs*)

class alexlib.toolbox.**FigurePolicy**(*value*)
Bases: enum.Enum

An enumeration.

add_new = 'Create a new figure with same name but with added suffix'

close_create_new = 'Close the previous figure that has the same figname and create a n

same = 'Grab the figure of the same name'

class alexlib.toolbox.**ImShow**(**images_list: Union[list, numpy.ndarray], sup_titles=None, sub_labels=None, labels=None, save_type=<class 'alexlib.toolbox.SaveType.Null'>, save_name=None, save_dir=None, save_kwargs=None, subplots_adjust=None, gridspec=None, tight=True, info_loc=None, nrows=None, ncols=None, ax=None, figsize=None, figname='im_show', figure_policy=<FigurePolicy.add_new: 'Create a new figure with same name but with added suffix'>, auto_brightness=True, delay=200, pause=False, **kwargs*)
Bases: *alexlib.toolbox.FigureManager*

animate()

annotate(*event*, *axis=None*, *data=None*)

artist = 'internal'

static cm(*im*, *nrows=3*, *ncols=7*, ***kwargs*)
Useful for looking at one image in multiple cmaps

Parameters

- **im** –
- **nrows** –
- **ncols** –
- **kwargs** –

Returns

classmethod complex(*data*, *pause=True*, ***kwargs*)

classmethod from_directories(**directories*, *extension='png'*, ***kwargs*)

classmethod from_saved(**things*, ***kwargs*)

classmethod from_saved_images_path_lists(**image_list*, ***kwargs*)

parser = 'internal'

static resize(*path*, *m*, *n*)

stream = 'update'

static test()

alexlib.toolbox.**L**
alias of *alexlib.toolbox.List*

class alexlib.toolbox.**List**(*obj_list=None*)
Bases: list, *alexlib.toolbox.Base*

Use this class to keep items of the same type.

append(*obj*)
Append object to the end of the list.

apply(*func*, **args*, *lest=None*, *jobs=None*, *depth=1*, ***kwargs*)

Parameters

- **jobs** –
- **func** – func has to be a function, possibly a lambda function. At any rate, it should return something.
- **args** –
- **lest** –
- **depth** – apply the function to inner Lists
- **kwargs** – a list of outputs each time the function is called on elements of the list.

Returns

attr(*name*)

combine()

df(*names=None*)

filter(*func*)

find(*patt*, *match='fnmatch'*)
Looks up the string representation of all items in the list and finds the one that partially matches the argument passed. This method is a short for self.filter(lambda x: **string_** in str(x)) If you need more complicated logic in the search, revert to filter method.

find_index(*string_*)

classmethod from_replication(*obj*, *count*)

index_entries(*start*, *end=None*, *step=None*)
Used to access entries of items

property len

method(*name*, **args*, ***kwargs*)

modify(*func*, *lest=None*)
Modifies objects rather than returning new list of objects, hence the name of the method. :param func: a string that will be executed, assuming idx, x and y are given. :param lest: :return:

property np

print(*nl=1*, *sep=False*, *char='-'*, *style=<class 'str'>*)

sample(*size=1*)

save_items(*directory*, *names=None*, *saver=None*)

sort(**args*, ***kwargs*)
Stable sort *IN PLACE*.

sorted(**args*, ***kwargs*)

to_struct(*keys=None*)
it has to be a property so that the struct is updated when list is updated.

class alexlib.toolbox.**Log**(*path=None*)
Bases: object

finish()

class alexlib.toolbox.**Manipulator**
Bases: object

static expand_axis(*array*, *ax_idx*, *factor*)

static indexer(*axis*, *myslice*, *rank=None*)
Returns a tuple of slicers.

static merge_adjacent_axes(*array*, *ax1*, *ax2*)
Multiplies out two axes to generate reduced order array. :param array: :param ax1: :param ax2: :return:

static merge_axes(*array*, *ax1*, *ax2*)
Brings ax2 next to ax1 first, then combine the two axes into one. :param array: :param ax1: :param ax2: :return:

static slicer(*array*, *a_slice: slice*, *axis=0*)

class alexlib.toolbox.**P**(**args*, ***kwargs*)
Bases: pathlib.PosixPath, pathlib.Path, *alexlib.toolbox.Base*

Path Class: Designed with one goal in mind: any operation on paths MUST NOT take more than one line of code.

append(*name=''*, *suffix=None*)
Add extra text after file name, and optionally add extra suffix. e.g: blahlah.extenion ==> becomes ==> blah/blah_name.extension

property browse

property browse2

clean()
removes contents on a folder, rather than deleting the folder.

compress(*op_path=None*, *base_dir=None*, *format_='zip'*, ***kwargs*)

copy(*target=None*, *contents=False*, *verbose=False*)
contents: copy the parent directory or its contents.

create(*parents=True*, *exist_ok=True*, *parent_only=False*)
Creates directory while returning the same object

decompress()

delete(*are_you_sure=False*)

explore()

find(**args*, *r=True*, ***kwargs*)
short for globbing then using next method to get the first result

get_num(*astring=None*)

property len

listdir()

make_python_name(*astring=None*)

move(*new_path*)

myglob(*pattern='*'*, *r=False*, *list_=True*, *files=True*, *folders=True*, *dotfiles=False*, *return_type=None*, *absolute=True*, *filters=None*, *win_order=False*)

Parameters

- **win_order** –
- **self** –
- **filters** –
- **dotfiles** –
- **pattern** – regex expression.
- **r** – recursive search
- **list** – output format, list or generator.
- **files** – include files in search.
- **folders** – include directories in search.
- **return_type** – output type, Pathlib objects or strings.
- **absolute** – return relative paths or abosolute ones.

Returns search results.

:param visible: exclude hidden files and folders (Windows)

prepend(*name*, *stem=False*)
Add extra text before file name e.g: blahlah.extenion ==> becomes ==> blah/name_blah.extension

readit(*reader=None*, ***kwargs*)

relativity_transform(*reference='deephead'*, *abs_reference=None*)
Takes in a path defined relative to reference, transform it to a path relative to execution directory, then makes it absolute path.

> **Warning:** reference must be included in the execution directory. Otherwise, absolute path of reference should be provided.

renameit(*new_name*)

send2trash()

setitem(*key*, *val*)

size(*units='mb'*)

split(*at=None*, *index=None*)
Splits a path at a given string or index :param self: :param at: :param index: :return: two paths

property string

static tmp(*folder=None*, *fn=None*, *path='home'*)
folder is created. file name is not created, only appended.

property trunk
useful if you have multiple dots in file name where .stem fails.

unzip(*op_path=None*, *fname=None*, ***kwargs*)

zip(*op_path=None*, *arcname=None*, ***kwargs*)

class alexlib.toolbox.**Read**
Bases: object

static csv(*path*, **args*, ***kwargs*)

static json(*path*)
Returns a Structure

static mat(*path*, *correct_dims=True*)

Parameters

- **path** –
- **correct_dims** –

Returns Structure object

static nii(*path*)

static npy(*path*)
returns Structure if the object loaded is a dictionary

static pickle(*path*)

static read(*path*, ***kwargs*)

class alexlib.toolbox.**Save**
Bases: object

static json(*path*, *obj*)
This format is compatible with simple dictionaries that hold strings or numbers but nothing more than that. E.g. arrays or any other structure. An example of that is settings dictionary. It is useful because it can be inspected using any text editor.

static mat(*path=AlexPath(/home/docs/tmp_results)*, *mdict=None*)
Avoid using mat for saving results because of incompatiblity. * Nones are not accepted. * Scalars are conveteed to [1 x 1] arrays. * etc. Unless you want to pass the results to Matlab animals, avoid this format.

static pickle(*path*, *obj*)

class alexlib.toolbox.**SaveType**
Bases: object

Programming philosophy: this class only worries about saving, and saving only. In other words, the figure must be fully prepared beforehand. Names here are only used for the purpose of saving, never putting titles on figures.

class GIF(*interval=100*, ***kwargs*)
Bases: *alexlib.toolbox.SaveType.GenericSave*

Requirements: same axis must persist, only new objects are drawn inside it. This is not harsh as no one wants to add multiple axes on top of each other. Next, the objects drawn must not be removed, or updated, instead they should pile up in axis.

do not pass names in the add method. names will be extracted from figures. # usually it is smoother when adding animate=True to plot or imshow commands for GIF purpose

Works for images only. Add more .imshow to the same axis, and that's it. imshow will conver up previous images. For lines, it will superimpose it and will look ugly.

If you clear the axis, nothing will be saved. This should not happend. The class will automatically detect new lines by their "neo" labels. and add them then hide them for the next round. Limitation of ArtistAnimation: works on lines and images list attached to figure axes. Doesn't work on axes, unless you add large number of them. As such, titles are not incorporated etc.

finish()

class GIFAuto(*plotter_class*, *data*, *interval=500*, *extension='gif'*, *fps=4*, ***kwargs*)
Bases: *alexlib.toolbox.SaveType.GenericAuto*

class GIFFileBased(*fps=4*, *dpi=100*, *bitrate=1800*, *_type='GIFFileBased'*, ***kwargs*)
Bases: *alexlib.toolbox.SaveType.GenericSave*

finish()

class GIFFileBasedAuto(*plotter_class*, *data*, *fps=4*, *dpi=150*, *bitrate=2500*, *_type='GIFFileBasedAuto'*, ***kwargs*)
Bases: *alexlib.toolbox.SaveType.GenericAuto*

class GIFPipeBased(**args*, ***kwargs*)
Bases: *alexlib.toolbox.SaveType.GIFFileBased*

class GIFPipeBasedAuto(**args*, ***kwargs*)
Bases: *alexlib.toolbox.SaveType.GIFFileBasedAuto*

class GenericAuto(*plotter_class*, *data*, *names_list=None*, ***kwargs*)
Bases: *alexlib.toolbox.SaveType.GenericSave*

animate()

save_type = 'auto'

class GenericSave(*save_dir=None*, *save_name=None*, *watch_figs=None*, *max_calls=2000*, *delay=100*, ***kwargs*)
Bases: `object`

You can either pass the figures to be tracked or, pass them dynamically at add method, or, add method will capture every figure and axis

add(*fig_names=None*, *names=None*, ***kwargs*)

stream = 'clear'

class MPEGFileBased(**args*, ***kwargs*)
Bases: *alexlib.toolbox.SaveType.GIFFileBased*

class MPEGFileBasedAuto(**args*, ***kwargs*)
Bases: *alexlib.toolbox.SaveType.GIFFileBasedAuto*

class MPEGPipeBased(**args*, ***kwargs*)
Bases: *alexlib.toolbox.SaveType.GIFFileBased*

class MPEGPipeBasedAuto(**args*, ***kwargs*)
Bases: *alexlib.toolbox.SaveType.GIFFileBasedAuto*

class Null(**args*, ***kwargs*)
Bases: *alexlib.toolbox.SaveType.GenericSave*

Use this when you do not want to save anything. This class will help plot to work faster by removing lines of previous plot, so you get live animation cheaply.

finish()

class NullAuto(***kwargs*)
Bases: *alexlib.toolbox.SaveType.GenericAuto*

class PDF(**args*, ***kwargs*)
Bases: *alexlib.toolbox.SaveType.GenericSave*

For pdf, you just need any figure manager, [update, clear, accumalate], preferabbly fastest.

finish(*open_result=True*)

class PDFAuto(***kwargs*)
Bases: *alexlib.toolbox.SaveType.GenericAuto*

class PNG(**args*, ***kwargs*)
Bases: *alexlib.toolbox.SaveType.GenericSave*

finish()

class PNGAuto(***kwargs*)
Bases: *alexlib.toolbox.SaveType.GenericAuto*

class alexlib.toolbox.**Struct**(*dictionary=None*, ***kwargs*)
Bases: *alexlib.toolbox.Base*

Use this class to keep bits and sundry items.

append(**others*, ***kwargs*)

static concat_dicts_(**dicts*, *method=None*, *lenient=True*, *collect_items=False*, *copyit=True*)

copy()

classmethod defaultdict(**args*, ***kwargs*)

property df

property dict

empty_class()

classmethod from_keys_values(*names*, *values*)

classmethod from_names(**names*, *default=None*)

index(*idx*)

inverse()

items()

keys()

map(*keys*)

plot(*artist=None*, *xdata=None*)

spawn_from_values(*values*)

update(**args*, ***kwargs*)
Accepts dicts and keyworded args

values()

class alexlib.toolbox.**VisibilityViewer**(*artist=None*, *hide_artist_axes=True*)
Bases: *alexlib.toolbox.FigureManager*

add(*artist=None*, *increment_index=True*, *hide_artist_axes=True*)

animate()

artist = 'external'

finish()

hide_artist_axes()

parser = 'external'

stream = 'accumulate'
Viewer Building Philosophy:

Viewer should act as Saver and Browser:

- How is the data viewed:
  - Can either be an artist himself, as in ImShow.
  - external artist is required to view the data (especially non image data)
- Data parsing:
  - **internal for loop to go through all the dataset passed.** # Allows manual control over parsing.
  - **external for loop. It should have add method.** # Manual control only takes place after the external loop is over. #TODO parallelize this.
- Refresh mechanism.
  - Clear the axis.
  - accumulate, using visibility to hide previous axes.
  - The artist has an update method.

The artist has to have:

- fig, ax, txt attributes. ax and txt should be lists.
- the ax and txt attributes should always belong to the same figure.

Here and in all Visibility classes, the artist is assumed to be always creating new axes along the way.

class alexlib.toolbox.**VisibilityViewerAuto**(*data=None*, *artist=None*, *memorize=False*, *transpose=True*, *save_type=<class 'alexlib.toolbox.SaveType.Null'>*, *save_dir=None*, *save_name=None*, *delay=1*, *titles=None*, *legends=None*, *x_labels=None*, *pause=True*, ***kwargs*)
Bases: *alexlib.toolbox.VisibilityViewer*

animate()

static test()

alexlib.toolbox.**accelerate**(*func*, *ip*)
Conditions for this to work: * Must run under __main__ context * func must be defined outside that context.

To accelerate IO-bound process, use multithreading. An example of that is somthing very cheap to process, but takes a long time to be obtained like a request from server. For this, multithreading launches all threads together, then process them in an interleaved fashion as they arrive, all will line-up for same processor, if it happens that they arrived quickly.

To accelerate processing-bound process use multiprocessing, even better, use Numba. Method1 use: multiprocessing / multithreading. Method2: using joblib (still based on multiprocessing) from joblib import Parallel, delayed Fast method using Concurrent module

alexlib.toolbox.**assert_package_installed**(*package*)

alexlib.toolbox.**batcher**(*func_type='function'*)

alexlib.toolbox.**batcherv2**(*func_type='function'*, *order=1*)

alexlib.toolbox.**browse**(*path*, *depth=2*, *width=20*)

Parameters

- **width** – if there are more than this items in a directory, dont' parse the rest.
- **depth** – to prevent crash, limit how deep recursive call can happen.
- **path** – absolute path

Returns constructs a class dynamically by using object method.

alexlib.toolbox.**get_time_stamp**(*name=None*)

alexlib.toolbox.**run_globally**(*name*, *asis=False*)
Takes in a function name, reads it source code and returns a new version of it that can be run in the main. This is useful to debug functions and class methods alike.

alexlib.toolbox.**tmp**(*folder=None*, *fn=None*, *path='home'*)
folder is created. file name is not created, only appended.

1.1.1.3 alexlib.deeplearning module

class alexlib.deeplearning.**BaseModel**(*hp=None*, *data=None*, *model=None*, *compiler=None*, *history=None*)

Bases: abc.ABC

build(*shape=None*, *dtype=<class 'numpy.float32'>*)
Building has two main uses.

- Useful to baptize the model, especially when its layers are built lazily. Although this will eventually happen as the first batch goes in. This is a must before showing the summary of the model.
- Doing sanity check about shapes when designing model.
- Sanity check about values and ranges when random normal input is fed.

Parameters

- **dtype** –
- **shape** –

Returns

compile(*loss=None*, *optimizer=None*, *metrics=None*, *compile_model=True*)
Updates compiler attributes. This acts like a setter.

- Must be run prior to fit method.
- Can be run only after defining model attribute.

config()

evaluate(*x_test=None*, *y_test=None*, *names_test=None*, *idx=None*, *viz=True*, *return_loss=False*, ***kwargs*)

fit(*viz=False*, *update_default=False*, *fit_kwargs=None*, *epochs=None*, ***kwargs*)

classmethod from_class_model(*path*)

classmethod from_class_weights(*path*, *hp=None*)

infer(*x*)
This method assumes numpy input, datatype-wise and is also preprocessed. NN is put in eval mode.

Parameters **x** –

Returns prediction as numpy

static load_model(*directory*)

load_weights(*directory*)

plot_loss()

plot_model()

postprocess(*x*, **args*, ***kwargs*)

predict(*x*, ***kwargs*)

predict_from_position(*position*, *viz=True*, ***kwargs*)

predict_from_s_obj(*s_obj*, *cal_obj=None*, *viz=True*, *names=None*, ***kwargs*)

preprocess(**args*, ***kwargs*)

save_class(*weights_only=True*, *version='0'*)
Simply saves everything:

1. Hparams
2. Data specs
3. Model architecture or weights depending on the following argument.

Parameters

- **version** –
- **weights_only** – self-explanatory

Returns

save_model(*directory*)

save_weights(*directory*)

summary()

switch_to_l1(*epochs=10*)

switch_to_sgd(*epochs=10*)

viz(*pred*, *gt=None*, *names=None*, ***kwargs*)
Assumes numpy inputs

class alexlib.deeplearning.**Compiler**(*loss=None*, *optimizer=None*, *metrics=None*)
Bases: object

class alexlib.deeplearning.**DataReader**(*hp=None*, *data_specs=None*, *split=None*)
Bases: object

data_split(**args*, *strings=None*, ***kwargs*)

Parameters

- **args** – whatever to be sent to train_test_split
- **kwargs** – whatever to be sent to train_test_split
- **strings** –

Returns

static from_saved(*path*)
This method offers an alternative constructer for DataReader class. Use this when loading training data is not required. It requires saved essential parameters to be stored. Those parameters are required by models to work.

Parameters **path** – full path to the saved .npy file containing a dictionary of attributes names and values.

Returns An object with attributes similar to keys and values as in dictionary loaded.

save()

class alexlib.deeplearning.**Device**(*value*)
Bases: enum.Enum

An enumeration.

auto = 'auto'

cpu = 'cpu'

gpu0 = 'gpu0'

gpu1 = 'gpu1'

two_gpus = '2gpus'

class alexlib.deeplearning.**Ensemble**(*hp_class=None*, *data_class=None*, *model_class=None*, *n=15*, *_from_saved=False*)
Bases: object

clear_memory()

fit(*shuffle_train_test=True*, *save=True*, ***kwargs*)

classmethod from_saved_models(*parent_dir*, *wrapper_class*)

classmethod from_saved_weights(*parent_dir*, *wrapper_class*)

get_model(*n*)

infer(*s*)

predict_from_position(*pos*, *central_tendency='mean'*, *bins=None*, *verbose=True*)

read_fit_results()

class alexlib.deeplearning.**HPTuning**
Bases: object

gen_writer()

static help()
Steps of use: subclass this and do the following: * Set directory attribute. * set params * set accuracy metric * generate writer. * implement run method. * run loop method. * in the command line, run *tensorboard –logdir <self.dir>*

loop()

optimize()

run(*param_dict*)

class alexlib.deeplearning.**HyperParam**
Bases: object

Benefits of this way of organizing the hyperparameters:

- one place to control everything.
- When doing multiple experiments, one command in console reminds you of settings used in that run (hp.__dict__).
- Ease of saving settings of experiments! and also replicating it later.

static from_saved(*path*)

save()

save_code()

property save_dir
Ensures that the folder created is directly under deephead root, no matter what directory the run was done from. This is especially useful during imports, resulting in predicted behaviour.

class alexlib.deeplearning.**KerasOptimizer**(*d*)
Bases: object

tune()

alexlib.deeplearning.**config_device**(*handle*, *device: alexlib.deeplearning.Device = <Device.gpu0: 'gpu0'>*)

Parameters

- **handle** – package handle
- **device** – device

Returns possibly a handle to device (in case of Pytorch)

alexlib.deeplearning.**get_mean_max_error**(*tf*)
For Tensorflow

1.1.1.4 alexlib.deeplearning_torch module

1.1.1.5 alexlib.miscellaneous module

alexlib.miscellaneous.**compute_num_of_lines_of_code_in_repo**(*path=AlexPath(/home/docs/checkouts/readthed*
extension='.py',
r=True, ***kwargs*)

alexlib.miscellaneous.**polygon_area**(*points*)
Return the area of the polygon whose vertices are given by the sequence points.

1.1.1.6 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z