Stylish

Release 0.4.0

Jun 29, 2020

Contents

Style transfer using *Deep Neural Network*.

CHAPTER 1

Introduction

A brief introduction to Stylish.

CHAPTER 2

Installing

Note: Using *Virtualenv* is recommended when evaluating or running locally.

Installation is simple with pip:

```
pip install stylish
```

2.1 Installing from source

You can also install manually from the source for more control. First obtain a copy of the source by either downloading the zipball or cloning the public repository:

```
git clone github.com:buddly27/stylish.git
```

Then you can build and install the package into your current Python environment:

```
pip install .
```

If actively developing, you can perform an editable install that will link to the project source and reflect any local changes made instantly:

```
pip install -e .
```

Note: If you plan on building documentation and running tests, run the following command instead to install required extra packages for development:

```
pip install -e .[dev]
```

Alternatively, just build locally and manage yourself:

```
python setup.py build
```

2.1.1 Building documentation from source

Ensure you have installed the 'extra' packages required for building the documentation:

```
pip install -e .[doc]
```

Then you can build the documentation with the command:

```
python setup.py build_sphinx
```

View the result in your browser at:

```
file:///path/to/stylish/build/doc/html/index.html
```

2.1.2 Running tests against the source

Ensure you have installed the 'extra' packages required for running the tests:

```
pip install -e .[test]
```

Then run the tests as follows:

```
python setup.py -q test
```

You can also generate a coverage report when running tests:

```
python setup.py -q test --addopts "--cov --cov-report=html"
```

View the generated report at:

```
file:///path/to/stylish/htmlcov/index.html
```

CHAPTER 3

Tutorial

A quick dive into using Stylish.

CHAPTER 4

Command line

4.1 stylish

Style transfer using deep neural network

```
stylish [OPTIONS] COMMAND [ARGS]...
```

Options

--version
Show the version and exit.

-v, --verbosity <verbosity>
Set the logging output verbosity. [default: info]

Options debuglinfolwarninglerror

4.1.1 apply

Apply a style generator model to an image.

```
stylish apply [OPTIONS]
```

Options

--model <model>
Path to trained style generator model which will be used to apply the style. [required]

-i, --input <input>
Path to image to transform. [required]

-o, --output <output>
Path to folder in which the transformed image will be saved. Current directory is used by default. [required]

4.1.2 download

Download necessary elements to train a style generator model

Example:

stylish download vgg19 stylish download coco2014 -o /tmp

```
stylish download [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...
```

coco2014

Download COCO 2014 Training dataset (13GB).

```
stylish download coco2014 [OPTIONS]
```

Options

-o, --output <output>
Output path to save the element. Current directory is used by default.

vgg19

Download pre-trained Vgg19 model (549MB).

```
stylish download vgg19 [OPTIONS]
```

Options

-o, --output <output>
Output path to save the element (Current directory is used by default)

4.1.3 extract

Extract the style to an image.

```
stylish extract [OPTIONS]
```

Options

--vgg <vgg>
Path to Vgg19 pre-trained model in the MatConvNet data format.

-s, --style <style>
Path to image from which the style features will be extracted.

-o, --output <output>
Path to folder in which the style pattern image will be saved. Current directory is used by default.

4.1.4 train

Train a style generator model.

```
stylish train [OPTIONS]
```

Options

--vgg <vgg>
Path to Vgg19 pre-trained model in the MatConvNet data format.

-s, --style <style>
Path to image from which the style features will be extracted.

-t, --training <training>
Path to a training dataset folder (e.g. COCO 2014).

--limit <limit>
Maximum number of files to use from the training dataset folder.

-l, --learning-rate <learning_rate>
Learning rate for optimizer. [default: 0.001]

-b, --batch-size <batch_size>
Batch size for training. [default: 4]

-e, --epochs <epochs>
Epochs to train for. [default: 2]

-C, --content-weight <content_weight>
Weight of content in loss function. [default: 7.5]

-S, --style-weight <style_weight>
Weight of style in loss function. [default: 100.0]

-T, --tv-weight <tv_weight>
Weight of total variation term in loss function. [default: 200.0]

-o, --output <output>
Path to folder in which the trained model will be saved. Current directory is used by default.

--log-path <log_path>
Path to extract the log information. Default is the same path as the output path.

4.1.5 transfer

Transfer a style to an image.

```
stylish transfer [OPTIONS]
```

Options

--vgg <vgg>
Path to Vgg19 pre-trained model in the MatConvNet data format.

-i, --input <input>
Path to image to transform. [required]

-s, --style <style>
Path to image from which the style features will be extracted.

-l, --learning-rate <learning_rate>
Learning rate for optimizer. [default: 0.001]

-I, --iterations <iterations>
Iterations to train for. [default: 100]

-C, --content-weight <content_weight>
Weight of content in loss function. [default: 7.5]

-S, --style-weight <style_weight>
Weight of style in loss function. [default: 100.0]

-T, --tv-weight <tv_weight>
Weight of total variation term in loss function. [default: 200.0]

-o, --output <output>
Path to folder in which the transformed image will be saved. Current directory is used by default.

--log-path <log_path>
Path to extract the log information. Default is the same path as the output path.

CHAPTER 5

API Reference

5.1 stylish

stylish.**transform_image**(*path*, *style_path*, *output_path*, *vgg_path*, *iterations=None*, *learning_rate=None*, *content_weight=None*, *style_weight=None*, *tv_weight=None*, *content_layer=None*, *style_layers=None*, *log_path=None*)

Generate new image from *path* with style from another image.

Usage example:

```
>>> transform_image(
...     "/path/to/image.jpg",
...     "/path/to/style_image.jpg",
...     "/path/to/output_image/",
...     "/path/to/vgg_model.mat"
... )
```

Parameters

- **path** – path to the image to transform.
- **style_path** – path to an image from which the style features will be extracted.
- **output_path** – path where the transformed image will be generated.
- **vgg_path** – path to the *Vgg19* pre-trained model in the *MatConvNet* data format.
- **iterations** – number of time that image should be trained against *style_path*. Default is *stylish.core.ITERATIONS_NUMBER*.
- **learning_rate** – *Learning Rate* value to train the model. Default is *stylish.core.LEARNING_RATE*.
- **content_weight** – weight of the content feature cost. Default is *stylish.core.CONTENT_WEIGHT*.

- **style_weight** – weight of the style feature cost. Default is *stylish.core.STYLE_WEIGHT*.
- **tv_weight** – weight of the total variation cost. Default is *stylish.core.TV_WEIGHT*.
- **content_layer** – Layer name from pre-trained *Vgg19* model used to extract the content information. Default is *stylish.vgg.CONTENT_LAYER*.
- **style_layers** – Layer names from pre-trained *Vgg19* model used to extract the style information with corresponding weights. Default is *stylish.vgg.STYLE_LAYERS*.
- **log_path** – path to extract the log information. Default is the same path as the output path.

Returns path to transformed image.

Note: Lists of all image formats currently supported

---

stylish.**create_model**(*training_path*, *style_path*, *output_path*, *vgg_path*, *learning_rate=None*, *batch_size=None*, *batch_shape=None*, *epoch_number=None*, *content_weight=None*, *style_weight=None*, *tv_weight=None*, *content_layer=None*, *style_layers=None*, *limit_training=None*, *log_path=None*)

Train a style generator model based on an image and a dataset folder

Usage example:

```
>>> create_model(
...     "/path/to/training_data/",
...     "/path/to/style_image.jpg",
...     "/path/to/output_model/",
...     "/path/to/vgg_model.mat"
... )
```

Parameters

- **training_path** – training dataset folder.
- **style_path** – path to an image from which the style features will be extracted.
- **output_path** – path where the trained model and logs should be saved
- **vgg_path** – path to the *Vgg19* pre-trained model in the *MatConvNet* data format.
- **learning_rate** – *Learning Rate* value to train the model. Default is *stylish.core.LEARNING_RATE*.
- **batch_size** – number of images to use in one training iteration. Default is *stylish.core.BATCH_SIZE*.
- **batch_shape** – shape used for each images within training dataset. Default is *stylish.core.BATCH_SHAPE*.
- **epoch_number** – number of time that model should be trained against *training_images*. Default is *stylish.core.EPOCHS_NUMBER*.
- **content_weight** – weight of the content feature cost. Default is *stylish.core.CONTENT_WEIGHT*.
- **style_weight** – weight of the style feature cost. Default is *stylish.core.STYLE_WEIGHT*.

- **tv_weight** – weight of the total variation cost. Default is *stylish.core.TV_WEIGHT*.
- **content_layer** – Layer name from pre-trained *Vgg19* model used to extract the content information. Default is *stylish.vgg.CONTENT_LAYER*.
- **style_layers** – Layer names from pre-trained *Vgg19* model used to extract the style information with corresponding weights. Default is *stylish.vgg.STYLE_LAYERS*.
- **limit_training** – maximum number of files to use from the training dataset folder. By default, all files from the training dataset folder are used.
- **log_path** – path to extract the log information. Default is the same path as the output path.

Returns None

Note: Lists of all image formats currently supported

---

stylish.**apply_model**(*model_path*, *input_path*, *output_path*)

Apply style generator model to a new image.

Usage example:

```
>>> apply_model(
...     "/path/to/saved_model/",
...     "/path/to/image.jpg",
...     "/path/to/output_image/"
... )

/path/to/output/image.jpg
```

Parameters

- **model_path** – path to trained model saved.
- **input_path** – path to image to inferred model to.
- **output_path** – path folder to save image output.

Returns path to transformed image.

Note: Lists of all image formats currently supported

---

stylish.**extract_style_pattern**(*path*, *output_path*, *vgg_path*, *style_layers=None*)

Generate style pattern images from image *path*.

Usage example:

```
>>> apply_model(
...     "/path/to/style_image.jpg",
...     "/path/to/output/"
...     "/path/to/vgg_model.mat"
... )

/path/to/output/image.jpg
```

Parameters

- **path** – path to the image to extract style pattern images from.
- **output_path** – path where the images will be generated.
- **vgg_path** – path to the *Vgg19* pre-trained model in the *MatConvNet* data format.
- **style_layers** – Layer names from pre-trained *Vgg19* model used to extract the style information with corresponding weights. Default is *stylish.vgg.STYLE_LAYERS*.

Returns list of image paths generated.

5.1.1 stylish.command_line

stylish.command_line.**CONTEXT_SETTINGS = {'help_option_names': ['-h', '--help'], 'max_conte**
Click default context for all commands.

5.1.2 stylish.core

stylish.core.**BATCH_SIZE = 4**
Default batch size used for training.

stylish.core.**BATCH_SHAPE = (256, 256, 3)**
Default shape used for each images within training dataset.

stylish.core.**EPOCHS_NUMBER = 2**
Default epoch number used for training a model.

stylish.core.**ITERATIONS_NUMBER = 100**
Default iteration number used for transferring a style to an image.

stylish.core.**CONTENT_WEIGHT = 7.5**
Default weight of the content for the loss computation.

stylish.core.**STYLE_WEIGHT = 100.0**
Default weight of the style for the loss computation.

stylish.core.**TV_WEIGHT = 200.0**
Default weight of the total variation term for the loss computation.

stylish.core.**LEARNING_RATE = 0.001**
Default *Learning Rate*.

stylish.core.**create_session**()
Create a *Tensorflow* session and reset the default graph.

Should be used as follows:

```
>>> with create_session() as session:
    ...
```

Returns *Tensorflow* session

stylish.core.**extract_style_from_path**(*path*, *vgg_mapping*, *style_layers*, *image_size=None*)
Extract style feature mapping from image *path*.

This mapping will be used to train a model which should learn to apply those features on any images.

Parameters

- **path** – path to image from which style features will be extracted.
- **vgg_mapping** – mapping gathering all weight and bias matrices extracted from a pre-trained *Vgg19* model (typically retrieved by *stylish.vgg.extract_mapping()*).
- **style_layers** – Layer names from pre-trained *Vgg19* model used to extract the style information with corresponding weights. Default is *stylish.vgg.STYLE_LAYERS*.
- **image_size** – optional shape to resize the style image.

list of 5 values for each layer used for style features extraction. Default is LAYER_WEIGHTS.

Returns

mapping in the form of:

```
{
    "conv1_1/Relu": numpy.array([...]),
    "conv2_1/Relu": numpy.array([...]),
    "conv3_1/Relu": numpy.array([...]),
    "conv4_1/Relu": numpy.array([...]),
    "conv5_1/Relu": numpy.array([...])
}
```

stylish.core.**optimize_image**(*image*, *style_mapping*, *vgg_mapping*, *log_path*, *iterations=None*, *learning_rate=None*, *content_weight=None*, *style_weight=None*, *tv_weight=None*, *content_layer=None*, *style_layer_names=None*)

Transfer style mapping features to *image* and return result.

The training duration can vary depending on the *Hyperparameters* specified (iterations number) and the power of your workstation.

Parameters

- **image** – 3-D Numpy array representing the image loaded.
- **style_mapping** – mapping of pre-computed style features extracted from selected layers from a pre-trained *Vgg19* model (typically retrieved by *extract_style_from_path()*)
- **vgg_mapping** – mapping gathering all weight and bias matrices extracted from a pre-trained *Vgg19* model (typically retrieved by *stylish.vgg.extract_mapping()*).
- **log_path** – path to save the log information into, so it can be used with *Tensorboard* to analyze the training.
- **iterations** – number of time that image should be trained against the style mapping. Default is *ITERATIONS_NUMBER*.
- **learning_rate** – *Learning Rate* value to train the model. Default is *LEARNING_RATE*.
- **content_weight** – weight of the content feature cost. Default is *CONTENT_WEIGHT*.
- **style_weight** – weight of the style feature cost. Default is *STYLE_WEIGHT*.
- **tv_weight** – weight of the total variation cost. Default is *TV_WEIGHT*.
- **content_layer** – Layer name from pre-trained *Vgg19* model used to extract the content information. Default is *stylish.vgg.CONTENT_LAYER*.
- **style_layer_names** – Layer names from pre-trained *Vgg19* model used to extract the style information. Default are layer names extracted from *stylish.vgg.STYLE_LAYERS* tuples.

Returns Path to output image generated.

stylish.core.**optimize_model**(*training_images*, *style_mapping*, *vgg_mapping*, *model_path*, *log_path*, *learning_rate=None*, *batch_size=None*, *batch_shape=None*, *epoch_number=None*, *content_weight=None*, *style_weight=None*, *tv_weight=None*, *content_layer=None*, *style_layer_names=None*)

Create style generator model from a style mapping and a training dataset.

The training duration can vary depending on the *Hyperparameters* specified (epoch number, batch size, etc.), the power of your workstation and the number of images in the training data.

The model trained will be saved in *model_path*.

Parameters

- **training_images** – list of images to train the model with.
- **style_mapping** – mapping of pre-computed style features extracted from selected layers from a pre-trained *Vgg19* model (typically retrieved by `extract_style_from_path()`)
- **vgg_mapping** – mapping gathering all weight and bias matrices extracted from a pre-trained *Vgg19* model (typically retrieved by `stylish.vgg.extract_mapping()`).
- **model_path** – path to save the trained model into.
- **log_path** – path to save the log information into, so it can be used with *Tensorboard* to analyze the training.
- **learning_rate** – *Learning Rate* value to train the model. Default is `LEARNING_RATE`.
- **batch_size** – number of images to use in one training iteration. Default is `BATCH_SIZE`.
- **batch_shape** – shape used for each images within training dataset. Default is `BATCH_SHAPE`.
- **epoch_number** – number of time that model should be trained against *training_images*. Default is `EPOCHS_NUMBER`.
- **content_weight** – weight of the content feature cost. Default is `CONTENT_WEIGHT`.
- **style_weight** – weight of the style feature cost. Default is `STYLE_WEIGHT`.
- **tv_weight** – weight of the total variation cost. Default is `TV_WEIGHT`.
- **content_layer** – Layer name from pre-trained *Vgg19* model used to extract the content information. Default is `stylish.vgg.CONTENT_LAYER`.
- **style_layer_names** – Layer names from pre-trained *Vgg19* model used to extract the style information. Default are layer names extracted from `stylish.vgg.STYLE_LAYERS` tuples.

Returns None

stylish.core.**compute_cost**(*session*, *style_mapping*, *output_node*, *batch_size=None*, *content_weight=None*, *style_weight=None*, *tv_weight=None*, *content_layer=None*, *style_layer_names=None*, *input_namespace='vgg1'*, *output_namespace='vgg2'*)

Compute total cost.

Parameters

- **session** – *Tensorflow* session.

- **style_mapping** – mapping of pre-computed style features extracted from selected layers from a pre-trained *Vgg19* model (typically retrieved by *extract_style_from_path()*)
- **output_node** – output node of the model to train.
- **batch_size** – number of images to use in one training iteration. Default is *BATCH_SIZE*.
- **content_weight** – weight of the content feature cost. Default is *CONTENT_WEIGHT*.
- **style_weight** – weight of the style feature cost. Default is *STYLE_WEIGHT*.
- **tv_weight** – weight of the total variation cost. Default is *TV_WEIGHT*.
- **content_layer** – Layer name from pre-trained *Vgg19* model used to extract the content information. Default is *stylish.vgg.CONTENT_LAYER*.
- **style_layer_names** – Layer names from pre-trained *Vgg19* model used to extract the style information. Default are layer names extracted from *stylish.vgg.STYLE_LAYERS* tuples.
- **input_namespace** – Namespace used for the pre-trained *Vgg19* model added after the input node. Default is "vgg1".
- **output_namespace** – Namespace used for the pre-trained *Vgg19* model added after *output_node*. Default is "vgg2".

Returns Tensor computing the total cost.

stylish.core.**compute_content_cost**(*session*, *layer_name1*, *layer_name2*, *batch_size=4*, *content_weight=7.5*)

Compute content cost.

Parameters

- **session** – *Tensorflow* session.
- **layer_name1** – Layer name from pre-trained *Vgg19* model used to extract the content information of input node.
- **layer_name2** – Layer name from pre-trained *Vgg19* model used to extract the content information of output node.
- **batch_size** – number of images to use in one training iteration. Default is *BATCH_SIZE*.
- **content_weight** – weight of the content feature cost. Default is *CONTENT_WEIGHT*.

Returns Tensor computing the content cost.

stylish.core.**compute_style_cost**(*session*, *style_mapping*, *layer_names1*, *layer_names2*, *batch_size=4*, *style_weight=100.0*)

Compute style cost.

Parameters

- **session** – *Tensorflow* session.
- **style_mapping** – mapping of pre-computed style features extracted from selected layers from a pre-trained *Vgg19* model (typically retrieved by *extract_style_from_path()*)
- **layer_names1** – Sorted layer names used in *style_mapping*.

- **layer_names2** – Layer name from pre-trained *Vgg19* model used to extract the style information of output node.
- **batch_size** – number of images to use in one training iteration. Default is *BATCH_SIZE*.
- **style_weight** – weight of the style feature cost. Default is *STYLE_WEIGHT*.

Returns Tensor computing the style cost.

stylish.core.**compute_total_variation_cost**(*output_node*, *batch_size*, *tv_weight=200.0*)
Compute total variation cost.

Parameters

- **output_node** – output node of the model to train.
- **batch_size** – number of images to use in one training iteration.
- **tv_weight** – weight of the total variation cost. Default is *TV_WEIGHT*.

Returns Tensor computing the total variation cost.

stylish.core.**load_dataset_batch**(*index*, *training_images*, *batch_size=None*, *batch_shape=None*)
Return list of images for current batch *index*.

Usage example:

```
>>> for index in range(len(training_images) // batch_size)):
...     images = load_dataset_batch(
...         index, training_images,
...         batch_size=batch_size
...     )
```

Parameters

- **index** – index number of the current batch to load.
- **training_images** – complete list of images to train the model with.
- **batch_size** – number of images to use in one training iteration. Default is *BATCH_SIZE*.
- **batch_shape** – shape used for each images within training dataset. Default is *BATCH_SHAPE*.

Returns 4-dimensional matrix storing images in batch.

stylish.core.**save_model**(*session*, *input_node*, *output_node*, *path*)
Save trained model from *session*.

Parameters

- **session** – *Tensorflow* session.
- **input_node** – input placeholder node of the model trained.
- **output_node** – output node of the model trained.
- **path** – Path to save the model into.

Returns None

stylish.core.**infer_model**(*model_path*, *input_path*)
Inferred trained model to convert input image.

Parameters

- **model_path** – path to trained model saved.
- **input_path** – path to image to inferred model to.

Returns Path to output image generated.

5.1.3 stylish.filesystem

stylish.filesystem.**create_log_path**(*style_path*, *relative_path=None*)

Return *Tensorflow* log path.

Example:

```
>>> create_log_path("/path/to/Foo.jpg")
"/tmp/stylish/log/foo-2019-09-14_15:12:11/"

>>> create_log_path("/path/to/Foo.jpg", relative_path="/path")
"/path/stylish/log/foo-2019-09-14_15:12:11/"
```

Parameters

- **style_path** – path to an image from which the style features will be extracted.
- **relative_path** – Path to a folder to generate logs into. The temporary folder is used otherwise.

Returns Path to save *Tensorflow* logs.

stylish.filesystem.**load_image**(*path*, *image_size=None*)

Return 3-D Numpy array from image *path*.

Parameters

- **path** – path to image file to load.
- **image_size** – targeted size of the image matrix loaded. By default, the image will not be resized.

Returns 3-D Numpy array representing the image loaded.

Note: Lists of all formats currently supported

---

stylish.filesystem.**save_image**(*image*, *path*)

Save *image_matrix* to *path*.

Parameters

- **image** – 3-D Numpy array representing the image to save.
- **path** – path to image file to save *image* into.

Returns None

Note: Lists of all formats currently supported

---

stylish.filesystem.**fetch_images**(*path*, *limit=None*)

Return list of image paths from *path*.

Parameters

- **path** – path to the directory containing all images to fetch.
- **limit** – maximum number of files to fetch from *path*. By default, all files are fetched.

Returns list of image file path.

stylish.filesystem.**ensure_directory**(*path*)
Ensure directory exists at *path*.

Parameters **path** – directory path.

Returns None

stylish.filesystem.**sanitise_value**(*value*, *substitution_character='_'*, *case_sensitive=True*)
Return *value* suitable for use with filesystem.

Parameters

- **value** – string value to sanitise.
- **substitution_character** – string character to replace awkward characters with. Default is "_".
- **case_sensitive** – indicate whether sanitised value should be kept with original case. Otherwise, the sanitised value will be return in lowercase. By default, the original case is preserved.

Returns sanitised value.

5.1.4 stylish.logging

stylish.logging.**root = <sawmill.handler.distribute.Distribute object>**
Top level handler responsible for relaying all logs to other handlers.

stylish.logging.**configure**(*stderr_level='info'*)
Configure logging handlers.

A standard error handler is created to output any message with a level greater than *stderr_level*.

A file handler is created to log warnings and greater to `stylish/logs` under system temporary directory.

Parameters **stderr_level** – minimum level to record.

Note: Standard Python logging are redirected to `sawmill` to unify the logging systems.

---

class stylish.logging.**Formatter**(*template*, *with_color=True*)
Mustache template to format `logs`.

__init__(*template*, *with_color=True*)
Initialize with *Mustache* template.

format(*logs*)
Format *logs* for display.

class stylish.logging.**Logger**(*name*, ***kw*)
Extended logger with timestamp and username information.

prepare(**args*, ***kw*)
Prepare and return a log for emission.

__init__(*name*, ***kw*)
Initialise logger with identifying *name*.

clear() → None. Remove all items from D.

clone()
Return a clone of this log.

This is a mixture of shallow and deep copies where the log instance and its attributes are shallow copied, but the actual mapping (items) are deepcopied.

copy() → a shallow copy of D

debug(*message*, ***kw*)
Log a debug level *message*.

error(*message*, ***kw*)
Log an error level *message*.

fromkeys()
Create a new dictionary with keys from iterable and values set to value.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

info(*message*, ***kw*)
Log an info level *message*.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

log(*message*, ***kw*)
Log a *message* with additional *kw* arguments.

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

warning(*message*, ***kw*)
Log a warning level *message*.

5.1.5 stylish.transform

The image transformation network is a deep residual convolutional neural network parameterized by weights.

The network body consists of five residual blocks. All non-residual convolutional layers are followed by an instance normalization and ReLU non-linearities with the exception of the output layer, which instead uses a scaled "tanh" to ensure that the output image has pixels in the range [0, 255]. Other than the first and last layers which use 9 × 9 kernels, all convolutional layers use 3 × 3 kernels.

See also:

Johnson et al. (2016). Perceptual losses for real-time style transfer and superresolution. CoRR, abs/1603.08155.

See also:

Ulyanov et al. (2017). Instance Normalization: The Missing Ingredient for Fast Stylization. CoRR, abs/1607.08022.

stylish.transform.**network**(*input_node*)

Apply the image transformation network.

The last node of the graph will be returned. The network will be applied to the current *Tensorflow* graph.

Example:

```
>>> g = tf.Graph()
>>> with g.as_default(), tf.Session() as session:
...     ...
...     network(input_node)
```

Parameters **input_node** – 4-D Tensor representing a list of images. It will be the input of the network.

stylish.transform.**residual_block**(*input_node*, *operation_name*, *in_channels*, *out_channels*, *kernel_size*, *strides*)

Apply a residual block to the network.

Parameters

- **input_node** – Input tensor of the block
- **operation_name** – Name of the operation which will be set as a scope.
- **in_channels** – Number of channels at the input of the block.
- **out_channels** – Number of channels at the output of the block.
- **kernel_size** – width and height of the convolution matrix used within the block.
- **strides** – stride of the sliding window for each dimension of *input_node*.

stylish.transform.**conv2d_layer**(*input_node*, *operation_name*, *in_channels*, *out_channels*, *kernel_size*, *strides*, *activation=False*)

Apply a 2-D convolution layer to the network.

Parameters

- **input_node** – Input tensor of the block
- **operation_name** – Name of the operation which will be set as a scope.
- **in_channels** – Number of channels at the input of the block.
- **out_channels** – Number of channels at the output of the block.
- **kernel_size** – width and height of the convolution matrix used within the block.
- **strides** – stride of the sliding window for each dimension of *input_node*.
- **activation** – indicate whether a 'relu' node should be added after the convolution layer. Default is False.

stylish.transform.**conv2d_transpose_layer**(*input_node*, *operation_name*, *in_channels*, *out_channels*, *kernel_size*, *strides*, *activation=False*)

Apply a transposed 2-D convolution layer to the network.

Parameters

- **input_node** – Input tensor of the block

- **operation_name** – Name of the operation which will be set as a scope.
- **in_channels** – Number of channels at the input of the block.
- **out_channels** – Number of channels at the output of the block.
- **kernel_size** – width and height of the convolution matrix used within the block.
- **strides** – stride of the sliding window for each dimension of *input_node*.
- **activation** – indicate whether a 'relu' node should be added after the convolution layer. Default is False.

stylish.transform.**instance_normalization**(*input_node*, *channels*)

Apply an instance normalization to the network.

Parameters

- **input_node** – Input tensor of the block
- **channels** – Number of channels at the input of the block.

See also:

Ulyanov et al. (2017). Instance Normalization: The Missing Ingredient for Fast Stylization. CoRR, abs/1607.08022.

5.1.6 stylish.vgg

Training model computation module from a *Vgg19* model.

The *Vgg19* model pre-trained for image classification is used as a loss network in order to define perceptual loss functions that measure perceptual differences in content and style between images.

The loss network remains fixed during the training process.

See also:

Johnson et al. (2016). Perceptual losses for real-time style transfer and superresolution. CoRR, abs/1603.08155.

See also:

Simonyan et al. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR, abs/1409.1556.

And the corresponding Vgg19 pre-trained model in the *MatConvNet* data format.

stylish.vgg.**STYLE_LAYERS = (('conv1_1/Relu', 1.0), ('conv2_1/Relu', 1.0), ('conv3_1/Relu',**

Default layers used to extract style features with coefficient weights.

stylish.vgg.**CONTENT_LAYER = 'conv4_2/Relu'**

Default layer used to extract the content features.

stylish.vgg.**extract_mapping**(*path*)

Compute and return weights and biases mapping from *Vgg19* model *path*.

The mapping should be returned in the form of:

```
{
    "conv1_1": {
        "weight": numpy.ndarray([...]),
        "bias": numpy.ndarray([...])
    },
    "conv1_2": {
```

(continues on next page)

(continued from previous page)

```
        "weight": numpy.ndarray([...]),
        "bias": numpy.ndarray([...])
    },
    ...
}
```

path should be the path to the *Vgg19* pre-trained model in the *MatConvNet* data format.

See also:

http://www.vlfeat.org/matconvnet/pretrained/

Raise `RuntimeError` if the model loaded is incorrect.

`stylish.vgg.`**`network`**(*vgg_mapping*, *input_node*)
Compute and return network from *mapping* with an *input_node*.

vgg_mapping should gather all weight and bias matrices extracted from a pre-trained *Vgg19* model (e.g. *`extract_mapping()`*).

input_node should be a 3-D Tensor representing an image of undefined size with 3 channels (Red, Green and Blue). It will be the input of the graph model.

`stylish.vgg.`**`conv2d_layer`**(*name*, *vgg_mapping*, *input_node*)
Add 2D convolution layer named *name* to *mapping*.

The layer returned should contain:

- A 2D convolution node
- A ReLU activation node

name should be the name of the convolution layer.

vgg_mapping should gather all weight and bias matrices extracted from a pre-trained *Vgg19* model (e.g. *`extract_mapping()`*).

input_node should be a Tensor that will be set as the input of the convolution layer.

Raise `KeyError` if the weight and bias matrices cannot be extracted from *vgg_mapping*.

`stylish.vgg.`**`pool_layer`**(*name*, *input_node*)
Return max pooling layer named *name*.

The layer returned should contain:

- An max pooling node

name should be the name of the max layer.

input_node should be a Tensor that will be set as the input of the max layer.

CHAPTER 6

Release and migration notes

Find out what has changed between versions and see important migration notes to be aware of when switching to a new version.

6.1 Release Notes

6.1.1 0.4.0

Released: 2019-07-07

- Added `stylish train --layer-weights` option to initialize weights for each layer from *`STYLE_LAYERS`*. The default value was initially hard-coded to 0.2, but has now be changed to 1.0 as it produces better results. ¶
- Updated *`stylish`*, *`stylish.transform`* and *`stylish.vgg`* to uses name scopes as much as possible in order to improve the graph visibility within *Tensorboard*. ¶
- Improved time display during training. ¶

6.1.2 0.3.0

Released: 2019-07-05

- Added `stylish train --limit` option to set a maximum number of files to use from the training dataset folder. ¶
- Record style loss, content loss, total variation loss and the sum of all losses to generate scalar curves within Tensorboard. ¶

6.1.3 0.2.0

Released: 2019-05-27

- Added `stylish download` command line option to download elements necessary for the training (*Vgg19* model and training data). ¶
- Added *stylish.logging* to manage logger using sawmill for convenience. ¶
- Removed `stylish.train` and moved logic within *stylish* to increase code readability. ¶
- **[command line]** Updated *stylish.command_line* to use click instead of argparse to manage the command line interface for convenience. ¶
- Fixed *stylish.transform.instance_normalization()* logic. ¶

6.1.4 0.1.4

Released: 2018-05-19

- Always use GPU for the training when available. ¶

6.1.5 0.1.3

Released: 2018-05-19

- Updated `stylish.train` module to prevent fixing the shape of the input placeholder. ¶

6.1.6 0.1.2

Released: 2018-05-18

- Updated *stylish.transform* module to let the size of the images unknown when processing the checkpoint. ¶
- Updated `stylish.train.extract_model()` to increase verbosity. ¶

6.1.7 0.1.1

Released: 2018-05-09

- Fixed `--content-target` command line option as it should take a single value, not a list of values. ¶
- Fixed `stylish.train.extract_model()` to pass the correct placeholder identifier to the session. ¶

6.1.8 0.1.0

Released: 2018-05-08

- Initial release. ¶

6.2 Migration notes

This section will show more detailed information when relevant for switching to a new version, such as when upgrading involves backwards incompatibilities.

CHAPTER 7

Glossary

Convolutional Neural Network Convolutional Neural Network (CNN) is a class of *Deep Neural Networks* most commonly applied to analyzing visual imagery.

See also:

https://en.wikipedia.org/wiki/Convolutional_neural_network

Deep Neural Network Deep Neural Networks (DNN) are *Neural Networks* with more than 2 layers between the input and output layers.

See also:

https://en.wikipedia.org/wiki/Deep_learning

Hyperparameter Parameter whose value is set before the learning process begins. By contrast, the values of other parameters are derived via training.

See also:

https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)

Learning Rate The learning rate or step size in machine learning is a *hyperparameter* which determines to what extent newly acquired information overrides old information

See also:

https://en.wikipedia.org/wiki/Learning_rate

Machine Learning Scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead.

See also:

https://en.wikipedia.org/wiki/Machine_learning

MatConvNet MatConvNet is a *MATLAB* toolbox implementing *Convolutional Neural Networks* for computer vision applications. It can store trained model in a ".mat" file.

See also:

http://www.vlfeat.org/matconvnet/

MATLAB MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks.

See also:

https://www.mathworks.com/help/matlab/

Mustache Simple web template system with implementations available for multiple languages

See also:

https://mustache.github.io

Neural Network Set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns.

See also:

https://en.wikipedia.org/wiki/Artificial_neural_network

TensorFlow An open source *Machine Learning* library for research and production

See also:

https://www.tensorflow.org/

Tensorboard TensorBoard provides the visualization and tooling needed for machine learning experimentation with *TensorFlow*

See also:

https://www.tensorflow.org/tensorboard

Vgg19

VGG-19 is a *Convolutional Neural Network* that is trained on more than a million images from the ImageNet database.

See also:

https://www.mathworks.com/help/deeplearning/ref/vgg19.html

Virtualenv A tool to create isolated Python environments.

See also:

https://virtualenv.pypa.io/en/latest/

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

Symbols

T

U

V

W