Stylish

Release 0.4.0

Jul 08, 2019

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 debug|info|warning|error

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. [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 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]

-L, --layer-weights <layer_weights>
Weights of layers used for style features extraction. [default: 1.0, 1.0, 1.0, 1.0, 1.0]

-o, --output <output>
Path to folder in which the trained model will be saved.

CHAPTER 5

API Reference

5.1 stylish

`stylish.BATCH_SIZE = 4`
Default batch size used for training.

`stylish.EPOCHS_NUMBER = 2`
Default epoch number used for training.

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

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

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

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

`stylish.LAYER_WEIGHTS = (1.0, 1.0, 1.0, 1.0, 1.0)`
Default weights for each layer used for style features extraction.

`stylish.train_model`(*style_path*, *training_path*, *output_path*, *vgg_path*, *learning_rate=0.001*, *batch_size=4*, *epoch_number=2*, *content_weight=7.5*, *style_weight=100.0*, *tv_weight=200.0*, *layer_weights=(1.0, 1.0, 1.0, 1.0, 1.0)*, *limit_training=None*)
Train a style generator model for *style_path* on *training_path*.

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.

Usage example:

```
>>> train_model(
...     "/path/to/style_image.jpg",
...     "/path/to/training_data/",
```

(continues on next page)

(continued from previous page)

```
...     "/path/to/output_model/",
...     "/path/to/vgg_model.mat"
... )
```

style_path should be the path to an image from which the style features will be extracted.

training_path should be the training dataset folder.

output_path should be the path where the trained model should be saved.

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

learning_rate should indicate the *Learning Rate* to minimize the loss. Default is `LEARNING_RATE`.

batch_size should indicate the number of training examples utilized in one iteration. Default is `BATCH_SIZE`.

epoch_number should indicate the number of time that the *training data* should be trained. Default is `EPOCHS_NUMBER`.

content_weight should indicate the weight of the content for the loss computation. Default is `CONTENT_WEIGHT`.

style_weight should indicate the weight of the style for the loss computation. Default is `STYLE_WEIGHT`.

tv_weight should indicate the weight of the total variation term for the loss computation. Default is `TV_WEIGHT`.

layer_weights should indicate a list of 5 values for each layer used for style features extraction. Default is `LAYER_WEIGHTS`.

limit_training should be the maximum number of files to use from the training dataset folder. By default, all files from the training dataset folder are used.

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

Apply style generator *model_path* for input image.

Return path to image generated.

Usage example:

```
>>> apply_model(
...     "/path/to/saved_model/",
...     "/path/to/input_image.jpg",
...     "/path/to/output/"
... )
```

model_path should be the path to a *Tensorflow* model path that has been `trained` on an other image to extract its style.

input_path should be the path to an image to apply the *model_path* to.

output_path should be the folder where the output image should be saved.

`stylish.`**`create_session`**()

Create a *Tensorflow* session and reset the default graph.

Should be used as follows:

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

`stylish.`**`compute_style_feature`**(*session*, *path*, *vgg_mapping*, *layer_weights=(1.0*, *1.0*, *1.0*, *1.0*, *1.0)*)

Return computed style features mapping from image *path*.

The style feature map will be used to penalize the predicted image when it deviates from the style (colors, textures, common patterns, etc.).

Usage example:

```
>>> compute_style_feature(session, path, vgg_mapping)

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

session should be a *Tensorflow* session.

path should be the path to an image from which the style features will be extracted.

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

layer_weights should indicate a list of 5 values for each layer used for style features extraction. Default is *`LAYER_WEIGHTS`*.

`stylish.`**`compute_loss`**(*session*, *input_node*, *style_features*, *vgg_mapping*, *batch_size=4*, *content_weight=7.5*, *style_weight=100.0*, *tv_weight=200.0*)

Create loss network from *input_node*.

Return a mapping with the content loss, the style loss, the total variation loss and the total loss nodes.

Usage example:

```
>>> compute_loss(session, input_node, style_features, vgg_mapping)

{
    "total": tf.Tensor(...),
    "content": tf.Tensor(...),
    "style": tf.Tensor(...),
    "total_variation": tf.Tensor(...)
}
```

session should be a *Tensorflow* session.

input_node should be the output tensor of the main graph.

style_features should be the style features map `extracted`.

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

batch_size should indicate the number of training examples utilized in one iteration. Default is *`BATCH_SIZE`*.

content_weight should indicate the weight of the content. Default is *`CONTENT_WEIGHT`*.

style_weight should indicate the weight of the style. Default is *`STYLE_WEIGHT`*.

tv_weight should indicate the weight of the total variation term. Default is *`TV_WEIGHT`*.

`stylish.optimize(session, training_node, training_data, input_node, loss_mapping, output_checkpoint, writer, batch_size=4, epoch_number=2)`

Optimize the loss for *training_node*.

session should be a Tensorflow session.

training_node should be the optimizer node that should be executed.

training_data should be a list containing all training images to feed to the *input_node*.

input_node should be the placeholder node in which should be feed each image from *training_data* to train the model.

loss_mapping should be a mapping of all loss nodes as returned by `compute_loss()`.

output_checkpoint should be the path to export each checkpoints to resume the training at any time. A checkpoint will be saved after each epoch and at each 500 batches.

writer is a FileWriter instance to record training data.

batch_size should indicate the number of training examples utilized in one iteration. Default is `BATCH_SIZE`.

epoch_number should indicate the number of time that the *training data* should be trained. Default is `EPOCHS_NUMBER`.

`stylish.get_next_batch(iteration, content_targets, batch_size, batch_shape)`

Return array with image matrices according to *iteration* index.

iteration should be an integer specifying the current portion of the images to return.

content_targets should be the list of image paths from which the content features should be extracted.

batch_size should be the size of the image list to return.

batch_shape should be indicate the dimensions in which each image should be resized to.

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.filesystem

`stylish.filesystem.load_image(image_path, image_size=None)`

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

image_size can be specified to resize the image.

`stylish.filesystem.save_image(image_matrix, path)`

Save *image_matrix* to *path*.

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

Return list of image paths from *path*.

limit should be the maximum number of files to fetch from *path*. By default, all files are fetched.

`stylish.filesystem.ensure_directory(path)`

Ensure directory exists at *path*.

`stylish.filesystem.sanitise_value(value, substitution_character='_', case_sensitive=True)`

Return *value* suitable for use with filesystem.

Replace awkward characters with *substitution_character*. Where possible, convert unicode characters to their closest "normal" form.

If not *case_sensitive*, then also lowercase value.

5.1.3 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.

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*.

5.1.4 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)
```

input_node should be a 4-D Tensor representing a batch 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.

input_node will be the input of the block.

in_channels should be the number of channels at the input of the block.

out_channels should be the number of channels at the output of the block.

kernel_size should be the width and height of the convolution matrix used within the block.

strides should indicate the 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.

input_node will be the input of the layer.

in_channels should be the number of channels at the input of the layer.

out_channels should be the number of channels at the output of the layer.

kernel_size should be the width and height of the convolution matrix used within the block.

strides should indicate the stride of the sliding window for each dimension of *input_node*.

activation should indicate whether a 'relu' node should be added after the convolution layer.

`stylish.transform.`**`conv2d_transpose_layer`**(*input_node*, *operation_name*, *in_channels*, *out_channels*, *kernel_size*, *strides*, *activation=None*)

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

input_node will be the input of the layer.

in_channels should be the number of channels at the input of the layer.

out_channels should be the number of channels at the output of the layer.

kernel_size should be the width and height of the convolution matrix used within the block.

strides should indicate the stride of the sliding window for each dimension of *input_node*.

activation should indicate whether a 'relu' node should be added after the convolution layer.

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

Apply an instance normalization to the network.

input_node will be the input of the layer.

See also:

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

5.1.5 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', 'conv2_1/Relu', 'conv3_1/Relu', 'conv4_1/Relu',`**

List of layers used to extract style features.

`stylish.vgg.`**`CONTENT_LAYER = 'conv4_2/Relu'`**
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": {
        "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

A

B

C

D

U

V

W