CLIP tags exploration

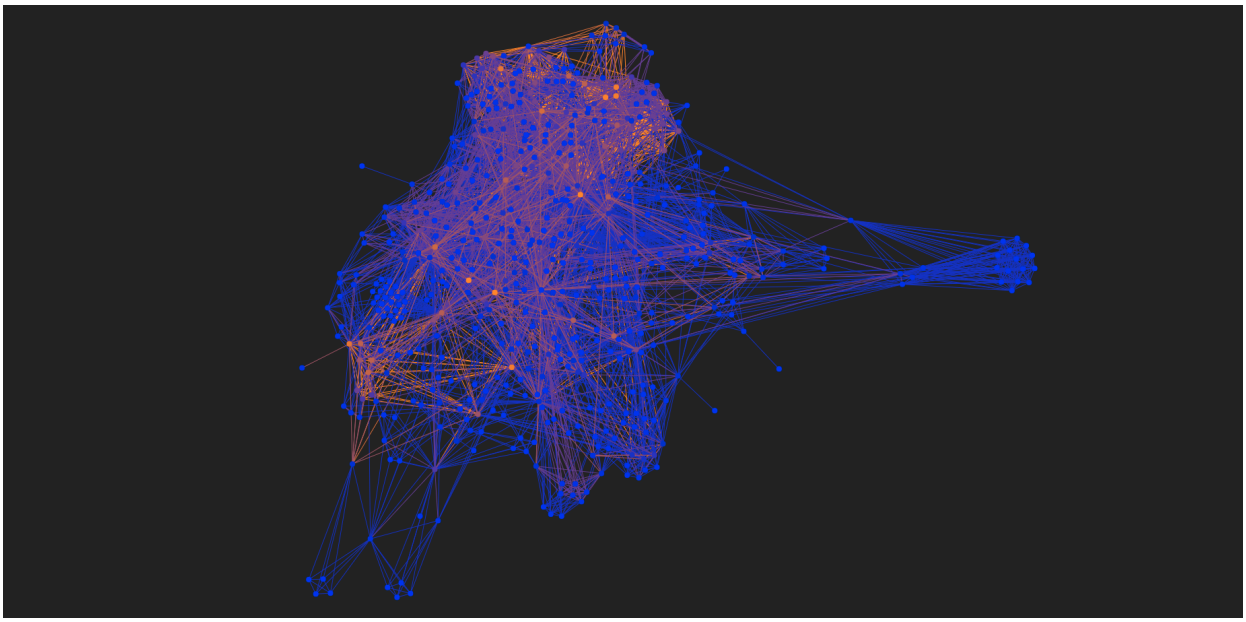
Release 0.1

6r1d

Aug 06, 2021

CONTENTS:

Online demo



CHAPTER

ONE

INTRODUCTION

CLIP and VQGan allow you to generate beautiful images from text. The descriptions of these images should be more specific than in natural language and are called prompts[1]. The goal I have while making this document and code is to document how to reach the best results using prompts.

At Jun 1st., 2021, Aran Komatsuzaki tweeted that mentioning "Unreal Engine" changes the visual style and quality of an image. Since CLIP learned on the images from the Internet, the "Unreal Engine" can be called one of its many sources of inspiration. Even before that, many looked for tags, words that change how CLIP draws things.

I've experimented with many CLIP prompts using a Discord bot by BoneAmputee and decided to build a list of words I use often.

Then I experimented more, especially with a pencil style and understood I will need more than one list, because co-occurences of the words create a graph! I have also added many prompts by other users, often with some editing and pre-processing to make them more uniform. The `prompts` directory contains two files with mostly cleaned up prompt samples.

1.1 Uses

1.1.1 Counting tags

The basic use for the `tagnet` utility is to count tags and display the counted occurences for each of the tags.

You need to provide a directory with text files containing the prompts in a `path` command-line argument.

```
tagnet.py --path ./prompts --mode count_tags
```

Filtering

You may also need to filter tags by the number of occurences.

For now, these are the supported modes (you can put whitespaces between mode and a number):

=, >, <, >=, <=

Examples:

```
tagnet.py --mode count_tags --filter "=1"
```

[1] ArXiV: Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm by Laria Reynolds and Kyle McDonell. This article talks about the GPT-3 language model, but the same term applies to GPT-2, GPT-3, GPT-j and CLIP itself.

```
tagnet.py --path ./prompts --mode count_tags --filter ">1"
```

```
tagnet.py --path ./prompts --mode count_tags --filter "< 3"
```

```
tagnet.py --path ./prompts --mode count_tags --filter "<= 8"
```

```
tagnet.py --path ./prompts --mode count_tags --filter ">=5"
```

1.1.2 Tag graph

Displaying an approximate graph

Often, a prompt contains several tags, for example:

```
Sunset in a forest ; VRay ; 3D ; High detail
```

We've got two co-occurences:

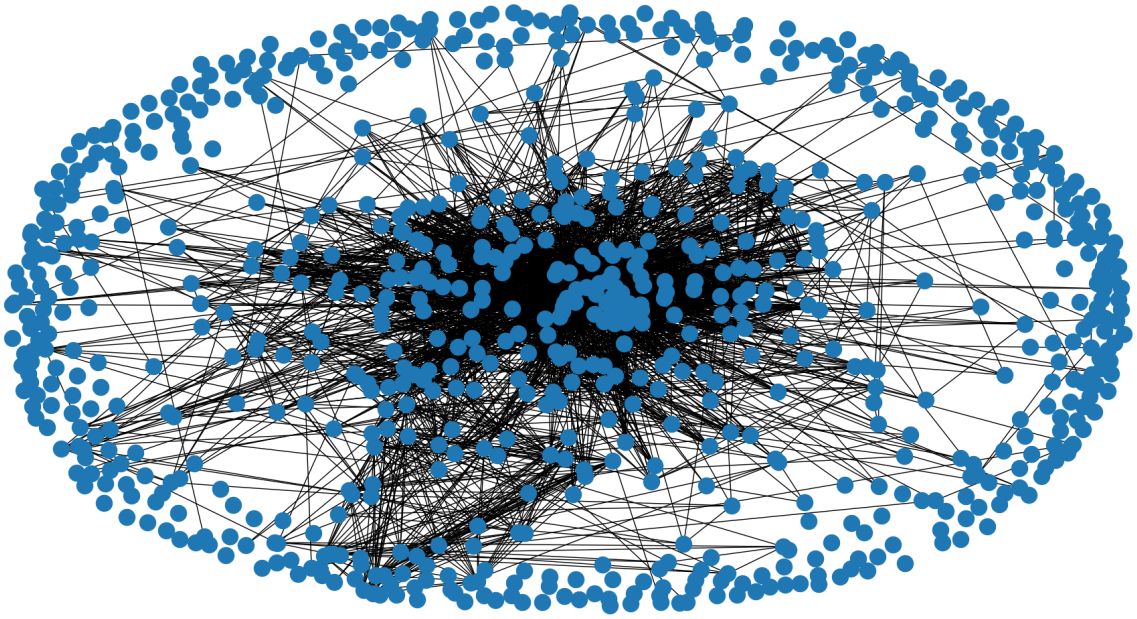
- `VRay` and `3D`
- `High detail` and `3D`

Edges for this command are weighted, based on an amount of said co-occurences in all available prompts.

To generate and see it, write:

```
tagnet.py --mode display_graph --path ./prompts
```

The graph is using Matplotlib and WxWidges and looks like that.



Displaying a web graph

There's a frontend side of the project: CLIP graph visualized. You may want to watch an online demo with existing tags or build your own tag graph and watch how it differs:

```
# Replace "your_path" with a path containing prompt directory and available for JSON␣
↪file export
# --path is a prompt directory
# --output_file is a path to a new JSON output file
tagnet.py --path ~/your_path/prompt_directory --mode export_graph --output_file ~/your_
↪path/graph.json
```

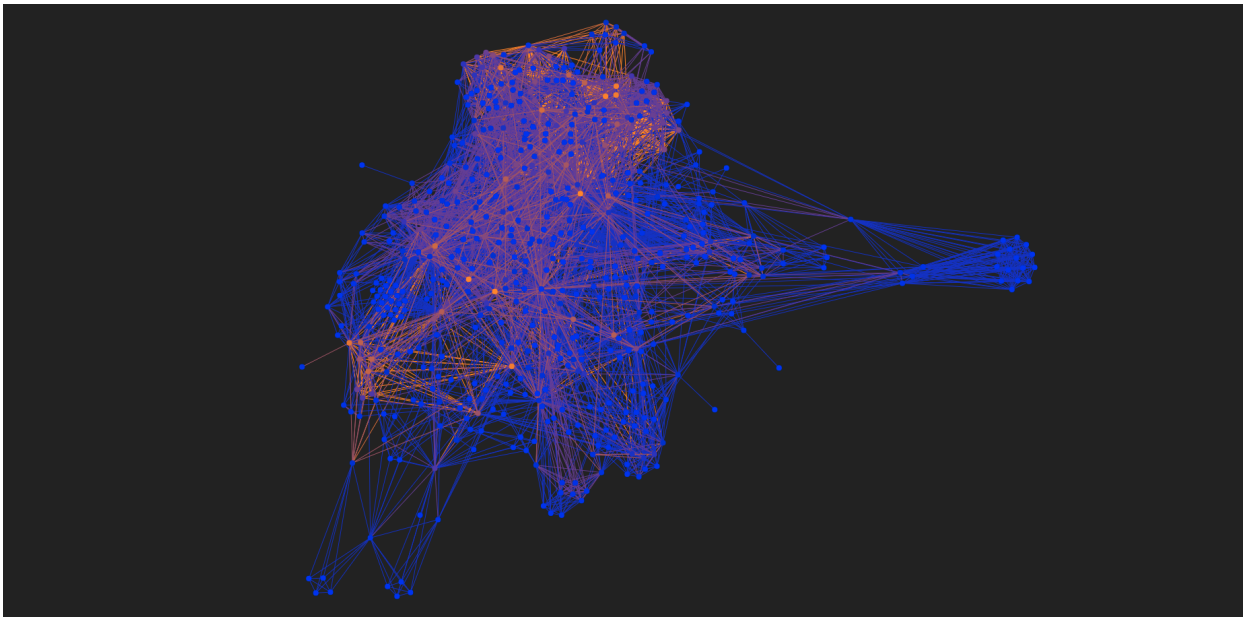
Now you can clone the visualization repository to use it locally and copy the generated `graph.json` as a data source.

```
# Clone a repository
git clone git@github.com:6r1d/CLIP_graph_visualized.git
cd CLIP_graph_visualized

# Copy a graph.json
cp ~/your_path/graph.json ./graph.json

# Run a Python 3 webserver locally on a 8080 port
# (any other webserver with static file support might work)
python3 -m http.server --bind 0.0.0.0 8080
```

Now, by visiting "http://0.0.0.0:8080" or "http://127.0.0.1:8080", you'll be able to see your own graph version. The visualizer is using a force-graph library by Vasco Asturiano. It allows you to zoom in and out, see tag names and shift the workspace.



1.2 Code documentation

Documents the code to make it easier to navigate and maintain.

1.2.1 Tagnet utility

1.2.2 Lib directory

cmd_args module

This module contains code for configuring commandline argument support and related `argparse` actions.

class lib.cmd_args.**NumberFilterAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

An `argparse.Action` subclass that validates the number filters. Accepts inputs like `<x`, `= x` or `>=x`, where `x` is an integer.

Ignores a space in the middle.

Raises **ValueError** – if an incorrect format is provided

class lib.cmd_args.**ReadableDirectoryAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

An `argparse.Action` subclass that checks if a directory is readable.

Raises

- **ArgumentTypeError** – if a path is invalid
- **ArgumentTypeError** – if a directory is unreadable

lib.cmd_args.**configure_parser**()

Configures `argparse` to accept arguments needed by the `tagnet` utility like "path", "output_file", "mode", "filter", etc.

process module

plot module

graph_util module

prompts module

Contains a function to load prompts from available files.

lib.prompts.**load_prompts**(*dir_path*)

Looks up a directory path, takes a full path for it, lists for directory contents and loads all available prompts.

Example

```
>>> from lib.prompts import load_prompts
>>> prompts = load_prompts('./prompts')
>>> prompts[-3:]
[
    '.imagine -pinene pool ; vray ; PBR ; HDR ; closeup ; DSLR ; hyperrealistic',
    '.imagine omicron ; vray ; hdr illumination ; contest winner',
    '.imagine the night ; vray ; isonoise ; contest winner ; highly sought art'
]
```

Parameters **dir_path** (*str*) – a path to the prompt directory

Returns a list of strings containing CLIP prompts

lib.prompts.**prompt_split**(*prompt*, *maxsplit=0*)

Split to unique prompts.

Examples

```
>>> prompt_split('.imagine the Fresnel lens ; in fine detail ; rendered in charcoal␣
↪| realistic', 1)
['.imagine the Fresnel lens', 'in fine detail ; rendered in charcoal | realistic']
```

```
>>> prompt_split('in fine detail ; rendered in charcoal | realistic')
['in fine detail', 'rendered in charcoal', 'realistic']
```

Parameters

- **prompt** (*str*) – a prompt to split
- **maxsplit** (*int*) – a maximum number of splits

Returns a list of strings containing prompt elements

tags module

This module contains:

- a generic tag processing class that corrects case, stores a tag list, counts tags
- a function that extracts a list of tags from a CLIP prompt string

class lib.tags.**Tag_processor**

Used to store tag indices, proper tag cases, global count of the tags.

Variables

- **case_fix_dict** (*dict*) – associates the lowercase string with properly cased ones
- **tag_list** (*list*) – a list of enumerated lowercase strings
- **global_tag_count** (*int*) – a count of all the tags added

add_tags(*tag_list*)

Works like put_tags, but returns nothing

Parameters **tag_list** (*list*) – a list of strings with tag names (case-insensitive)

Example

```
>>> from lib.tags import Tag_processor
>>> tp = Tag_processor()
>>> tp.add_tags(['SFX', 'high detail', 'light transport sharpening'])
```

get_tag_list()

Returns A list of dictionaries with "id", "name" and "rank" attribute. ID is an integer, name is a string, a rank is a float value containing the quotient of tag count divided by the global tag count.

Example

```
>>> from lib.tags import Tag_processor
>>> tp = Tag_processor()
>>> tp.put_tags(['landscape', 'beautiful', 'neon'])
[0, 1, 2]
>>> tp.get_tag_list()
[
    {'id': 0, 'name': 'landscape', 'rank': 0.3333333333333333},
    {'id': 1, 'name': 'beautiful', 'rank': 0.3333333333333333},
    {'id': 2, 'name': 'neon', 'rank': 0.3333333333333333}
]
```

get_tag_numbers()

Iterate a list of tags with their count.

Returns a list of tuples, containing tag names and numbers

Example

```
>>> from lib.tags import Tag_processor
>>> tp = Tag_processor()
>>> tp.put_tags(['landscape', 'beautiful', 'neon'])
[0, 1, 2]
>>> tp.get_tag_numbers()
[('landscape', 1), ('beautiful', 1), ('neon', 1)]
```

get_tag_rank(*tag_id*)

Parameters **tag_id** (*int*) – a tag index

Returns a rank of a tag, the quotient of tag count divided by the global tag count

put_tag(*tag*)

Parameters **tag** (*str*) – a tag name, case-insensitive

Returns a tag ID

Example

```
>>> from lib.tags import Tag_processor
>>> tp = Tag_processor()
>>> tp.put_tag('VFX')
0
>>> tp.put_tag('HDR')
1
>>> tp.put_tag('DSLR')
2
```

put_tags(*tag_list*)

Parameters **tag_list** (*list*) – a list of strings with tag names (case-insensitive)

Returns a list of tag IDs

Example

```
>>> from lib.tags import Tag_processor
>>> tp = Tag_processor()
>>> tp.put_tags(['SFX', 'high detail', 'light transport sharpening'])
[0, 1, 2]
```

lib.tags.**extract_tags**(*prompt*)

Extract a list of the tags from a single prompt.

Parameters **prompt** (*str*) – a prompt for the CLIP neural network

Example

```
>>> from lib.tags import extract_tags
>>> extract_tags('.imagine the color clash ; HDR ; hyperrealistic ; contest winner')
[
    'HDR',
    'hyperrealistic',
    'contest winner'
]
```

1.3 Plans

1.3.1 Modes

Currently, there's two modes to display the graph: Python's WxWidgets interface and a web interface.

There exists a potential to get more information out of the dataset by expanding available modes.

Community detection

I should try several community detection[1][2] methods.

Adjacency graph

A mode for an adjacency graph will require a bit more work, for example, exporting only a top N tags and limit tag lengths so everything can be displayed.

- StackOverflow: Method to save networkx graph to json graph?
- NetworkX: Reading and writing graphs » JSON

Word2Vec

I am not sure if it can be used as-is, but there were some works that remind me it can be useful to try later.[3][4]

1.3.2 Experiments

Edge weighting

Edge weights in pair_mgr are currently divided by an `edge_count` parameter. I am not sure it is an ideal option that allows to see the maximum amount of details.

Weighting by relation

- Will add edges between tags like `Abstract style` and `Abstract` add more context?
- How to weight those edges properly?

1.3.3 Argparse

- Save contents for `tag_manager` and `pair_manager`

---

[1] Understanding Community Detection Algorithms with Python NetworkX
[2] Louvain
[3] Node2vec: Scalable Feature Learning for Networks; How node2vec works — and what it can do that word2vec can't
[4] Paper2vec: Citation-Context Based Document Distributed Representation for Scholar Recommendation by Han Tian and Hankz Hankui Zhuo

PYTHON MODULE INDEX

INDEX