Distant Viewing Toolkit: A Python Package for the Analysis of Visual Culture

Taylor Arnold[1] **and Lauren Tilton**[2]

1 University of Richmond, Department of Mathematics and Computer Science **2** University of Richmond, Department of Rhetoric and Communication Studies

Links

- Repository
- Documenation
- Project

Compiled: 04 September 2019

Summary

The Distant Viewing Toolkit is a Python package designed to facilitate the computational analysis of visual culture. It addresses the challenges of working with moving images through the automated extraction and visualization of metadata summarizing the content (e.g., people/actors, dialogue, scenes, objects) and style (e.g., shot angle, shot length, lighting, framing, sound) of time-based media. The software allows users to explore media in many forms, including films, news broadcasts, and television, revealing how moving images shape cultural norms.

Many open-source projects provide implementations of state-of-the-art computer vision algorithms. However, there are limited options for users looking to quickly build end-to-end pipelines that link together common visual annotations. Different algorithms require varying dependencies, different input formats, and produce output using different schemas. Also, it is not easy to keep up with recent advances across a the many sub-fields of computer vision, it can be difficult to determine which algorithms to use, and a significant amount of work to manually test every available option. These problems are exacerbated when working with moving images because most available computer vision libraries take still images as inputs. The Distant Viewing Toolkit fills this need by (1) constructing an object-oriented framework for applying a collection of algorithms to moving images and (2) packages together common sound and computer vision algorithms in order to provide out-of-the-box functionality for common tasks in the computational analysis of moving images. Currently provided algorithms include functionaltiy for: shot detection (Pal et al., 2015), object detection (Li, Zhao, & Zhang, 2018), face detection (Jiang & Learned-Miller, 2017), face identification (Cao, Shen, Xie, Parkhi, & Zisserman, 2018), color analysis (Karasek, Burget, Uher, Masek, & Dutta, 2015), image similarity (Szegedy, Ioffe, Vanhoucke, & Alemi, 2017), optical flow (Farnebäck, 2003), and shot distance analysis (Butler, 2012).

The Distant Viewing Toolkit provides two interfaces. The first is a high-level command line interface designed to be accessible to users with limited programming experience. The second interface, a direct Python API, provides for customized and advanced processing of visual data. The package includes a custom JavaScript visualization engine that can be run on a user's machine to visualize the metadata for search and discovery. Metadata produced by either interface can also be further aggregated and analyzed to find patterns across a corpus. Together, these provide tools for the increasingly popular application of computational methods to the study of visual culture (Wevers & Smits, 2019).

The following sections give a general introduction to the ideas behind the package and its development. Detailed documentation and tutorials are provided in the package's documentation: https://distant-viewing.github.io/dvt/.

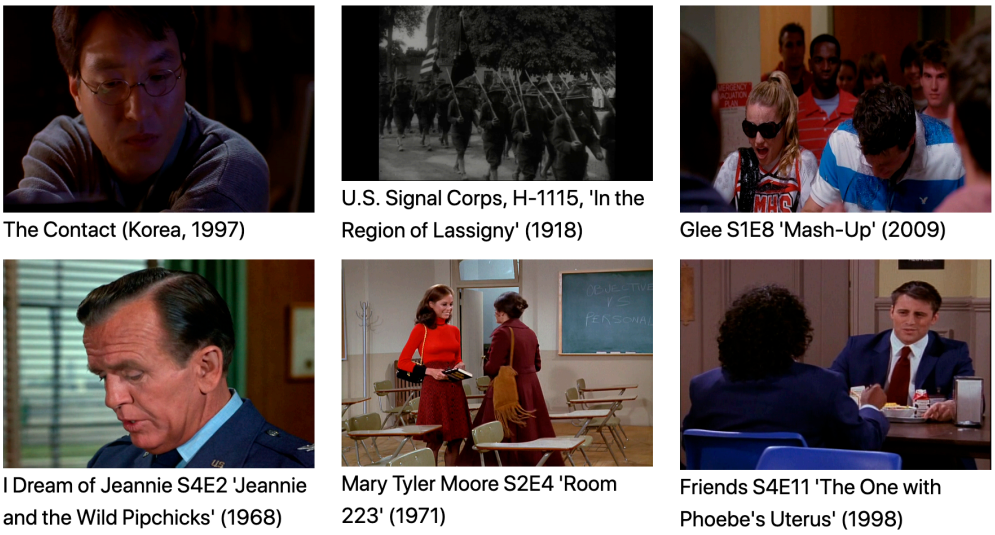

Figure 1: Example index page from the Distant Viewing Toolkit's command line video visualization. Clicking on an image opens a new page to display the extracted data from each individual input.

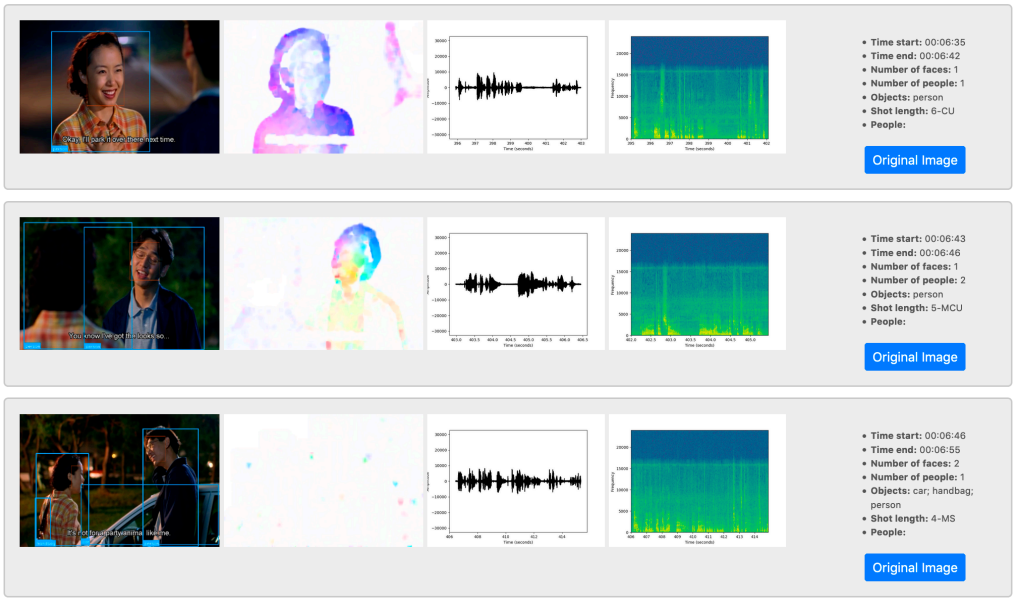

Figure 2: Example video page from the Distant Viewing Toolkit's command line video visualization from the South Korean movie *The Contact* (1997). Three shots are shown; scrolling up and down shows the other detected shots. For each shot, an annotated thumbnail, optical flow, audio tone, and spectrogram are shown along with metadata such as the start and stop time, number of detected faces, and the estimated shot length. Clicking on any image opens a pop-over div with a larger version of the image. The original image can be shown by clicking on the blue button marked "Original Image".

High-level Command Line Interface

The command line tools provide a fast way to get started with the toolkit. It can be utilized by users with no experience programming or knowledge of machine learning. It is ideal for quickly getting meaningful results. Users call the command line by directly executing the Python module (e.g., "python -m dvt"), specifying the desired pipeline, and pointed to a video file or directory of images. The output data can be visualized using a local webserver. Figure 1 shows an example landing page with six annotated video files. Clicking on an image opens a video-specific page (Figure 2), visualizing metadata about each cut. While the command line interface is meant to be easy to run out-of-the-box, it also affords a high-level of customization through command line options. These are documented within the toolkit using the **argparse** package. It is also possible to modify the visualization using custom CSS and JavaScript code. This makes the command-line interface particularly well-suited for classroom use, following the task-driven paradigm popular in Digital Humanities pedagogy (Birnbaum & Langmead, 2017).

Low-level Python API

While command line tools provide a fast way to get started with the toolkit, there is much more functionality available when using the full Python API. Using the distant viewing toolkit starts by constructing a `DataExtraction` object that is associated with input data (either a video file or a collection of still images). Algorithms are then applied to the extraction object, with results are stored as Pandas DataFrames that can be exported as CSV or JSON files. There are two distinct types of algorithms:

- **annotators**: algorithms that work directly with the source data but are able to only work with a small subset of frames or still images
- **aggregators**: algorithms that have access to information extracted from all previously run annotators and aggregators across across the entire input, but cannot directly access the visual data

The separation of algorithms into these two parts makes it easier to write straightforward, error-free code and closely mirrors the theory of *Distant Viewing* (Arnold & Tilton, 2019):

> Distant viewing is distinguished from other approaches by making explicit the interpretive nature of extracting semantic metadata from images. In other words, one must 'view' visual materials before studying them. Viewing, which we define as an interpretive action taken by either a person or a model, is necessitated by the way in which information is transferred in visual materials. Therefore, in order to view images computationally, a representation of elements contained within the visual material—a code system in semiotics or, similarly, a metadata schema in informatics—must be constructed. Algorithms capable of automatically converting raw images into the established representation are then needed to apply the approach at scale.

The annotator algorithms conduct the process of 'viewing' the material whereas the aggregator algorithms perform a 'distant' (e.g., separated from the raw materials) analysis of the visual inputs.

There are many annotators and aggregators currently available in the toolkit. Pipelines—pre-bundled sequences of annotators and aggregators—are also included in the package. Details of these implementations can be in API documentation. Users can construct custom Annotator and Aggregator objects, as described in the documentation's tutorial.

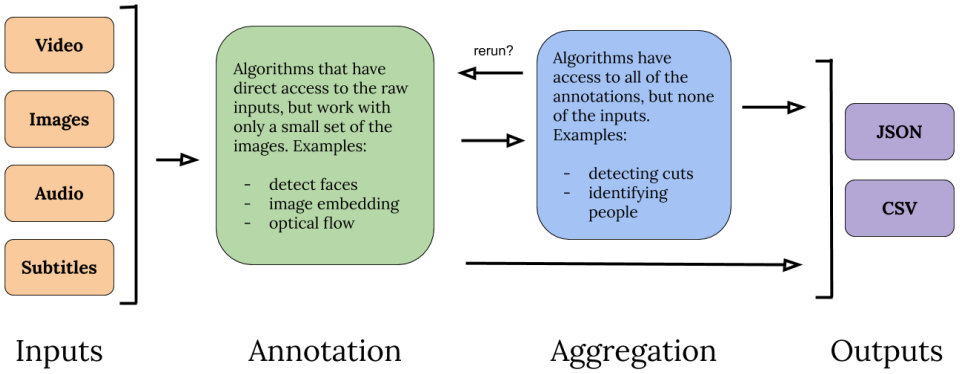

Figure 3: Schematic of the Distant Viewing Toolkit's internal architecture. Algorithms are split into two types: annotators that have access to small chunks of the raw inputs and aggregators that have access to all of the extracted annotations but not the input data itself.

Process and Development

Development of the Distant Viewing Toolkit follows best-practices for open-source software development (Wilson et al., 2014). Development of the software is done publicly through our GitHub repository, which has both a stable master branch and experimental development branch. The project includes an open-source license (GPL-2), uses the Contributor Covenant Code of Conduct (v1.4), and has user templates for submitting issues and pull requests (Tourani, Adams, & Serebrenik, 2017). We make use of integrated unit testing through the **pytest** package and TravisCL. The code passes checks for conforming to the common Python coding styles (checked with **pycodestyle**) and the relatively aggressive checks provided by **pylint** (Reitz & Schlusser, 2016). JavaScript coding style was verified with the static checkers JSHint and JSLint (Santos, Valente, & Figueiredo, 2015). Stable versions of the package are posted to PyPI as both source packages and pre-compiled wheels to make installation as straightforward as possible.

Because much of our audience is relatively non-technical, we have made every attempt to keep the installation process as simple as possible. The package can be installed using a fresh version of Anaconda Python and packages available through the pip command on PyPI (both of which can be installed through GUIs if preferred). We have kept dependencies to a minimum, and have avoided software that is known to be difficult to install. For example, all of the included deep-learning algorithms are built using Tensorflow to avoid requiring multiple deep-learning libraries (Abadi et al., 2016). While Tensorflow can occasionally throw errors, we have found the CPU-version to be relatively error-free for non-technical users to install on both Windows and macOS relative to popular alternatives such as Caffe and Torch.

As version 0.2.0, we consider the core architecture of the toolkit to be relatively stable. We plan to continue work on both specific algorithms available and improvements to the interactive, web-based interface. We also continue to utilize the toolkit for specific applications with research partnerships. Our first published example application looks at Network-Era Sitcoms in the United States (Arnold et al., 2019).

Acknowledgements

The Distant Viewing Toolkit is supported through a Digital Humanities Advancement Grant from the National Endowment for the Humanities (HAA-261239-18).

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {usenix} symposium on operating systems design and implementation ({osdi} 16)* (pp. 265–283).

Arnold, T. B., & Tilton, L. (2019). Distant viewing: Analyzing large visual corpora. *Digital Scholarship in the Humanities*. doi:10.1093/digitalsh/fqz013

Arnold, T. B., Tilton, L., & Berke, A. (2019). Visual style in two network era sitcoms. *Cultural Analytics*. doi:10.22148/16.043

Birnbaum, D. J., & Langmead, A. (2017). Task-driven programming pedagogy in the digital humanities. In *New directions for computing education* (pp. 63–85). Springer.

Butler, J. G. (2012). *Television: Critical methods and applications*. Routledge.

Cao, Q., Shen, L., Xie, W., Parkhi, O. M., & Zisserman, A. (2018). Vggface2: A dataset for recognising faces across pose and age. In *2018 13th ieee international conference on automatic face & gesture recognition (fg 2018)* (pp. 67–74).

Farnebäck, G. (2003). Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on image analysis* (pp. 363–370).

Jiang, H., & Learned-Miller, E. (2017). Face detection with the faster r-cnn. In *2017 12th ieee international conference on automatic face & gesture recognition (fg 2017)* (pp. 650–657).

Karasek, J., Burget, R., Uher, V., Masek, J., & Dutta, M. K. (2015). Color image (dis) similarity assessment and grouping based on dominant colors. In *2015 38th international conference on telecommunications and signal processing (tsp)* (pp. 756–759).

Li, X., Zhao, H., & Zhang, L. (2018). Recurrent retinanet: A video object detection model based on focal loss. In *International conference on neural information processing* (pp. 499–508). Springer.

Pal, G., Rudrapaul, D., Acharjee, S., Ray, R., Chakraborty, S., & Dey, N. (2015). Video shot boundary detection: A review. In *Emerging ict for bridging the future-proceedings of the 49th annual convention of the computer society of india csi volume 2* (pp. 119–127). Springer.

Reitz, K., & Schlusser, T. (2016). *The hitchhiker's guide to python: Best practices for development*. O'Reilly Media, Inc.

Santos, A. L., Valente, M. T., & Figueiredo, E. (2015). Using javascript static checkers on github systems: A first evaluation. In *Proccedings of the 3rd workshop on software visualization, evolution and maintenance (vem)* (pp. 33–40).

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first aaai conference on artificial intelligence*.

Tourani, P., Adams, B., & Serebrenik, A. (2017). Code of conduct in open source projects. In *2017 ieee 24th international conference on software analysis, evolution and reengineering (saner)* (pp. 24–33).

Wevers, M., & Smits, T. (2019). The visual digital turn: Using neural networks to study historical images. *Digital Scholarship in the Humanities*.

Wilson, G., Aruliah, D. A., Brown, C. T., Hong, N. P. C., Davis, M., Guy, R. T., Haddock, S. H., et al. (2014). Best practices for scientific computing. *PLoS biology*, *12*(1).