Proceedings of the 13th Conference on Language Resources and Evaluation (LREC 2022), pages 2528–2535
Marseille, 20-25 June 2022

Quevedo: Annotation and Processing of Graphical Languages*

Antonio F. G. Sevilla[1,2], **Alberto Díaz Esteban**[1,3], **José María Lahoz-Bengoechea**[2]
[1]Department of Software Engineering and Artificial Intelligence,
Facultad de Informática, c/ Profesor José García Santesmases, 9 28040 Madrid, Spain
[2]Department of Spanish Linguistics and Literary Theory,
Facultad de Filología, edificio D, c/ Prof. Aranguren s/n, 28040 Madrid, Spain
[3]Knowledge Engineering Institute,
Facultad de Psicología, Lateral 2, Campus de Somosaguas 28223 Pozuelo de Alarcón, Spain
Universidad Complutense de Madrid
afgs@ucm.es, albertodiaz@fdi.ucm.es, jmlahoz@ucm.es

Abstract

In this article, we present Quevedo, a software tool we have developed for the task of automatic processing of graphical languages. These are languages which use images to convey meaning, relying not only on the shape of symbols but also on their spatial arrangement in the page, and relative to each other. When presented in image form, these languages require specialized computational processing which is not the same as usually done either for natural language processing or for artificial vision. Quevedo enables this specialized processing, focusing on a data-based approach. As a command line application and library, it provides features for the collection and management of image datasets, and their machine learning recognition using neural networks and recognizer pipelines. This processing requires careful annotation of the source data, for which Quevedo offers an extensive and visual web-based annotation interface. In this article, we also briefly present a case study centered on the task of SignWriting recognition, the original motivation for writing the software. Quevedo is written in Python, and distributed freely under the Open Software License version 3.0.

Keywords: Graphical Languages, Annotation, Datasets, Machine Learning, Open Software

1. Introduction

The human language capacity is flexible and very powerful. It gives us not only the usual languages that are the focus of linguistics, i.e. natural, oral languages, but also artificial systems for communication that have some or most of the features of languages and can therefore be studied and processed with linguistic tools.

An example of these systems are graphical languages, which rely on images for communication. Musical notation (Figure 1), Feynman diagrams, elementary arithmetic notation, or the Unified Modeling Language (UML (Technical Committee: ISO/IEC JTC 1 Information technology, 2012), Figure 2) are systems which share some of the important characteristics of languages: signs with a signifier-signified nature, syntactic rules for combining them, and meaning which is compositional, a whole resulting from the consideration of the symbols but also their context and relative arrangement.

However, graphical languages have an important characteristic which is not common in natural languages: they are visual, and exploit the two dimensions of the page as a fundamental feature for codifying meaning. Location of symbols in the two dimensions and their relative arrangement is key to the correct understanding of graphical languages, but it is a problem beyond the tools habitual to natural language processing.

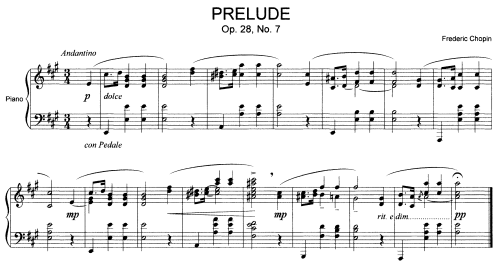

Figure 1: Modern musical notation as an example of a graphical language. Image by Prof.rick[1] (public domain).

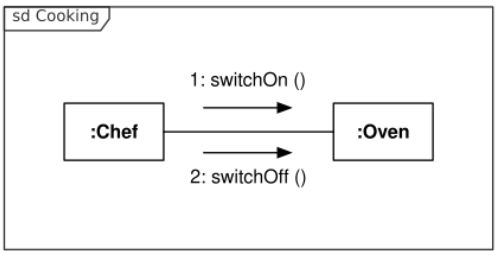

Figure 2: An UML communication diagram. Image by Oemmler[2], distributed under a CC BY-SA 3.0 license.

*Quevedo is available online at `https://github.com/agarsev/quevedo`

[1]`https://commons.wikimedia.org/wiki/File:Chopin_Prelude_7.png`

[2]`https://commons.wikimedia.org/wiki/File:UML_Communication_diagram.svg`

Figure 3: SignWriting transcription of the Spanish Sign Language sign for "coffee", performed iconically as stirring the beverage in a cup. A video can be seen online at SpreadTheSign[3].

If a graphical language is not stored in an abstract, semantic representation, but rather in its graphic realization, it is a challenge to process it automatically. It is necessary to locate symbols within the page, storing their relative locations since these are meaningful data, as well as finding the meaning intended for each symbol, meaning which can be encoded with overlapping graphical features such as size, rotation, color, etc.

In this article we present a software tool we have developed to deal with the annotation and automatic processing of images of graphical languages: Quevedo.

Quevedo is an open source python library and application with both command line and web interfaces. It can create, organize and manage sets of images containing samples of a graphical language. The web application provides a way of visually inspecting and exploring the datasets, as well as featuring an extensive annotation interface. It can be run locally, or deployed as a server for access or annotation by a team.

Processing of the images and their annotations can then be performed, or they can be used in other programs by importing Quevedo as a library. The processing allowed by Quevedo includes training and testing computer vision neural networks for the recognition of the target graphical language. These networks are configured using a declarative format, and can then be organized into a pipeline, allowing more complex processing and automatic recognition to be performed.

We strive for Quevedo to be a complete solution for the computational processing of graphical languages, guided by our own efforts in that domain. However, we are aware that every task has its own quirks, so Quevedo is very configurable, and extensible with user scripts. The organization of the datasets uses standard file formats and is straightforward and accessible, so the source data, annotations, and neural networks can also be consumed externally or enhanced with other tools.

In our research, we use Quevedo for managing SignWriting data. SignWriting (Sutton and Frost, 2008) is a graphical system for transcribing sign languages, using the two dimensional possibilities of the page to encode the movement and use of space of sign languages. In Figure 3, an example SignWriting transcription can be seen, and in Section 5 we give more detail into this use case.

Before that, in Section 2 we examine some related work that is relevant, Section 3 gives an overview of the different features in Quevedo, and Section 4 gives some notes into how to use Quevedo as a library and application. After we have presented our SignWriting case study in Section 5, Section 6 summarizes our conclusions with Quevedo and the research it has enabled, and Section 7 explains our future development and ideas for how we want to improve it and expand the range of problems it can solve.

2. Related Work

Recognizing graphical language images might look, at first glance, somewhat similar to the task of Optical Character Recognition (OCR). Images of writing, be they handwritten, or maybe printed, also consist of shapes in a particular arrangement which need to be recognized by taking into account the natural variability of the graphical realization. Open source tools such as Tesseract (Smith, 2013) exist to solve this problem, and indeed often use machine learning and similar algorithms to what we will later propose. The problem with OCR solutions is that they assume an underlying linear ordering to characters, that is, words are formed by characters in a sequence, and sentences by words linearly arranged. Graphical languages such as the ones Quevedo is suited to study are intrinsically two-dimensional, so OCR tools can not be practically used to process them.

In the realm of artificial vision, however, finding and recognizing objects in 2D (or even 3D) is a well studied task. Deep learning neural networks are a common and successful approach to this problem, and there is a wealth of software dedicated to this task, such as PyTorch (Paszke et al., 2019) or TensorFlow (Abadi et al., 2015). Indeed, Quevedo uses one such software, Darknet (Redmon, 2013 2016), to train neural networks on the annotated graphical language data, and uses it for inference on new data. These tools often give access to GPUs and other optimizations to make deep learning algorithms usable in reasonable time frames, and provide high-level abstractions to the networks' internal details. However, they still require the data to be prepared for the task, and accessory processing beyond the algorithm itself to be coded by the user.

The labeling task is an important part of this, and there is existing software to make this highly visual task efficient and correct. YOLO Mark[4] is a tool by the authors of the Darknet software that allows visually tagging image files with the objects within, marking their bounding boxes and their assigned class. However, this tool

[3] https://www.spreadthesign.com/es.es/word/6209/cafe/0/?q=caf%C3%A9

[4] https://github.com/AlexeyAB/Yolo_mark

presupposes that the task consists of single class labeling: each object has a tag and that is what is to be stored. The rich annotation often necessary when dealing with linguistic data, and which Quevedo makes possible, is not directly possible with YOLO Mark or other similar tools.

This problem is also visible regarding dataset organization and formatting. Datasets for image recognition and understanding, such as ImageNet (Deng et al., 2009) and COCO (Lin et al., 2015), do not need to deal with such a rich annotation as linguistic data often presents. It is often sufficient to present raw images on disk, with a properly formatted name that contains the single label, or maybe a text file beside the image containing the annotation. This straight-forward approach facilitates sharing and reproduction of results, and using it requires a small amount of code that is reasonable to expect every researcher to write themselves. But when data begin to be more complex, and annotations richer, it is also reasonable to have a tool to load the data from disk and access the annotation, a tool tailored for the problem at hand, such as Quevedo and its custom dataset organization architecture.

Data Version Control[5] (Kuprieiev et al., 2021) is another tool that provides some dataset organization and experiment preparation tools, but again is a generalist tool, requiring the researcher to write the specific code for their domain. Its concerns are, however, orthogonal to Quevedo's, so Quevedo datasets are very compatible with DVC due to their architecture and the use of regular files. Quevedo commands can be tracked in DVC pipeline files, and DVC can understand the parameters in Quevedo configuration files thanks to using TOML[6] as configuration language.

As we have seen, and as often happens in ML, there exist many independent but related tools, some more general and some more specialized. Many of them can be applied to our problem, but require a non-trivial amount of code and design to make them work for our purposes. To alleviate this problem, Quevedo is a tool that understands a more specialized domain, providing features for graphical language processing, while delegating to other tools when necessary.

3. Features

Quevedo can help organize the dataset, storing the source data, metadata and annotations. Quevedo datasets are file system directories, with a `config.toml` configuration file in the top level. This configuration file keeps common information about the dataset, such as annotation schema (an array of possible tags to give to the symbols of the graphical language), number of splits and their use for training or testing, or web interface configuration parameters. A title and description of the dataset can also be given.

The top level directory of the dataset is also a perfect place to have non-Quevedo files such as a "readme", license, or other information. It can also function as a Git[7] and/or DVC repository for better distribution.

Source images are organized into directories, keeping them as raw images on disk. Beside the image files, their annotations are stored as JSON files, allowing easy interoperability. Additional directories are used to store neural network configuration and trained weights, as well as inference results, or user-defined scripts and programs. This straightforward organization into directories and files is easy for other tools to consume if necessary, but Quevedo creates and manages it automatically for the user's convenience.

3.1. Annotation Features

Since Quevedo deals with visual data, source files in Quevedo datasets are images in bitmap format. These images are divided into two types: logograms and graphemes.

Graphemes are atomic, individual symbols that represent some particular meaning in the target graphical language, while logograms are images made of graphemes in complex and meaningful spatial arrangements. In the UML example in Figure 2, the different boxes, arrows and characters are graphemes. In the SignWriting example (Figure 3), the hand symbols along with the arrows indicating movement are the graphemes. In the sheet music excerpt in Figure 1, one can identify the notes, accidentals and other symbols as graphemes.

Both logograms and graphemes have dictionaries of tags, following a global schema defined for each dataset. Using a dictionary permits having more complex annotations than just a single label per object, for example having tags for different independent features, or a hierarchy of tags where some values depend on the values of other tags. Graphemes can be independent, or part of a logogram.

Logograms are comprised of graphemes, but the meaning of the logogram is not just the concatenation of the individual graphemes' meanings, but rather is derived from their geometric arrangement in the page. Logograms therefore have a list of contained graphemes, each with their own annotation, but these "bound" graphemes also have box data, representing the coordinates in the image where they can be found. Location data is fundamental for graphical languages, since the relative positions and sizes of the graphemes can have important repercussions on meaning.

Since annotation is a highly visual process, especially for the kind of data in Quevedo datasets, Quevedo can launch a web interface as shown in Figure 5. This web interface allows editing tags and metadata for all annotations, and drawing bounding boxes in logograms. Custom functions can also be run from the web interface, either aiding with the annotation process, or let-

[5] `https://dvc.org/`

[6] `https://toml.io/en/`

[7] `https://git-scm.com/`

ting users visualize the results of these functions without having to run any code.

Annotation files (logograms or independent graphemes) can also be assigned metadata, such as source of the data, annotators, or other custom values to aid in the use of the dataset. Additionally, they can be automatically assigned to different "splits". These splits can then be used to train and test on different subsets of the data, or even perform cross-validation.

3.2. Processing features

Quevedo can be used as a library, giving access to the annotations in an easy and organized way, so user code can perform custom processing without having to worry about files and directories or storage formats. However, there is also some higher-level functionality implemented, providing an abstraction over complex tasks that the owner of the dataset may want to carry on.

In the field of Computer Vision, a number of algorithms have been developed to deal with the task of automatically recognizing images or finding relevant objects in them. Quevedo can train neural networks for this task using the data annotated in the dataset. High-level configuration for the neural network is specified in the dataset configuration file, mainly the task to solve, the annotations to use for training, and which tags to learn to recognize. Based on this high-level description, the necessary Darknet configuration files are generated, and the data prepared so Darknet can process it. The neural network can then be trained with a single command, and a simple evaluation can also be performed. The resulting weights can be used by other applications, or directly from Quevedo.

However, linguistic processing is often not as simple as a single labelling task. There may be different preprocessing steps to run, or some analysis required beyond a machine learning algorithm. This is especially true when the available data is scarce, so rule-based processing is necessary alongside whatever data-based processing is possible. Moreover, language is often seen as organized in layers, and processing mimics these layers by building aggregated representations one step at a time. For this purpose, Quevedo can run pipelines, configured again in a declarative and high-level format in the dataset. Pipelines consist of series of steps, including neural network inference, custom processing scripts, or branching pipelines depending on tag values.

4. Usage

Quevedo is freely available on the Python Package Index (PyPI[8]) so the latest version can be installed with the command `python3 -m pip install quevedo`. Source code is available on GitHub[9] under the Open Software License 3.0[10] and documentation is also maintained using GitHub Pages[11].

To build a dataset, we need a collection of images to annotate. We can then use the command line to create the dataset, configure it, and add the images to the relevant subset:

```
[path/to]$ python3 -m pip install quevedo
[path/to]$ quevedo -D dataset create
[path/to]$ cd dataset
[path/to/dataset]$ quevedo add_images -i
↪  source_image_directory -g triangles
```

At this point, we will want to annotate the images. The first step is to decide on an annotation schema, an array of tags to give to logograms and graphemes, and the metadata schema, additional data which we will want to store about each file. This is configured in the dataset configuration file, which is in TOML format so easily editable with a text editor. An example configuration could be the following:

```
title = "Example dataset"

description = """
    This dataset is an imaginary example
    for how to use Quevedo. Annotations
    would be similar to those in vector
    graphics format such as SVG.
"""

tag_schema = [ "shape", "fill", "stroke" ]
meta_tags = [ "filename", "meaning" ]
...
```

With this, the web interface can be launched, useful for both visualization of the source images and annotation of their meaning:

```
[path/to/dataset]$ quevedo web --host
↪  'localhost' --port 8080
```

Once the data have been annotated, the dataset can be accessed from user code to compute corpus statistics, perform user processing, or train machine learning algorithms. In the following example, we find the most common colors used in our imaginary dataset:

```
from collections import Counter
from quevedo import Dataset

colors = Counter()
ds = Dataset('path/to/dataset')
```

[8] https://pypi.org/project/quevedo/
[9] https://github.com/agarsev/quevedo
[10] https://opensource.org/licenses/OSL-3.0
[11] https://agarsev.github.io/quevedo/latest/

```
for a in ds.get_annotations():
    fill = a.tags['fill']
    stroke = a.tags['stroke']
    colors[fill] += 1
    colors[stroke] += 1

print(colors.most_common(5))
```

To use the machine learning functionality provided with Quevedo, first we have to configure the networks and pipelines in the dataset configuration file:

```
[network.monochrome]
subject = "Classify black and white shapes"
task = "classify"
tag = [ "shape" ]
subsets = [ "squares", "triangles",
            "circles", "other" ]

[network.monochrome.filter]
criterion = "fill"
include = [ "black", "white" ]
# With this filter, only graphemes with a
# 'fill' tag of black or white will be
# used for training. This lets us have
# different networks for different tasks.
```

With the network configured, we can then use the command line to train it. This will take a bit of time, and at the end the network weights will be stored in the `network/monochrome` directory. These weights can be used to predict the "shape" tag of new data, and we can do a basic test of its accuracy on our own data:

```
[p/t/dataset]$ quevedo -N monochrome train
Neural network 'monochrome' trained

[p/t/dataset]$ quevedo -N monochrome test
Annotations tested: 136
{
    "overall": 0.9632352941176471,
    "det_acc": 1.0,
    "cls_acc": 0.9632352941176471
}
```

This is a basic introduction to Quevedo usage, and more detailed documentation can be found online at `https://agarsev.github.io/quevedo/latest/`. The command line interface can also list the available commands and parameters with the command `quevedo --help`[12].
In the next section we will give a brief overview of our own dataset and research using Quevedo, including an example of the web annotation. This dataset can be used to follow along with the explanations in this section or on the online documentation. A simpler example dataset is also provided with the Quevedo source code, and can be found in the `examples/toy_arithmetic` directory. This dataset also serves as an example of the how Quevedo can be used for the annotation of different graphical languages, as it contains examples of elementary arithmetic operations —additions, substractions, etc., performed visually, as would be performed by students. An example can be seen in Figure 4.

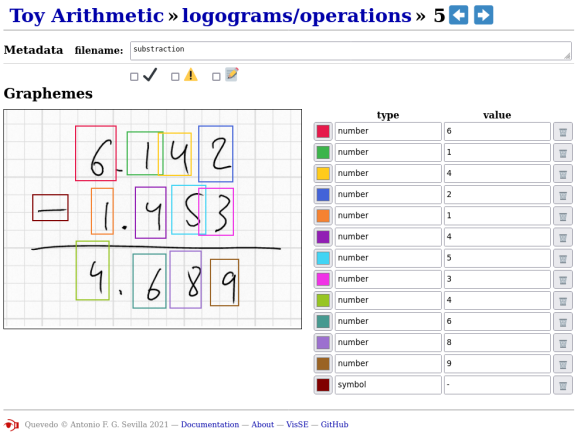

Figure 4: Example of the use of Quevedo to annotate the graphical language of elementary arithmetic, where the bidimensional position of elements is semantically relevant. Section 5 describes a real example of annotation and processing of a different graphical language, but this example shows that the same techniques can be used for different languages and systems.

If the source code of Quevedo is downloaded, the latest development features can be tested. For this, we recommend using `Poetry`[13], a python environment and dependency manager. For example, we could clone the source code repository and use `Poetry` to install dependencies, allowing us to examine the example "toy arithmetic" dataset using the web annotation interface. This would give a result similar to Figure 4, accessible using our own local browser. The following sequence of commands, adapted to our own environment, could be used to this end:

```
[~]$ git clone
↪  https://github.com/agarsev/quevedo
[~]$ cd quevedo
[quevedo]$ poetry install --extras "web"
[quevedo]$ cd examples/toy_arithmetic
[toy_arithmetic]$ poetry run quevedo info
[toy_arithmetic]$ poetry run quevedo web
```

[12] Quevedo's command line functionality is implemented with the excellent "Click" library: `https://github.com/pallets/click`.

[13] `https://python-poetry.org/`

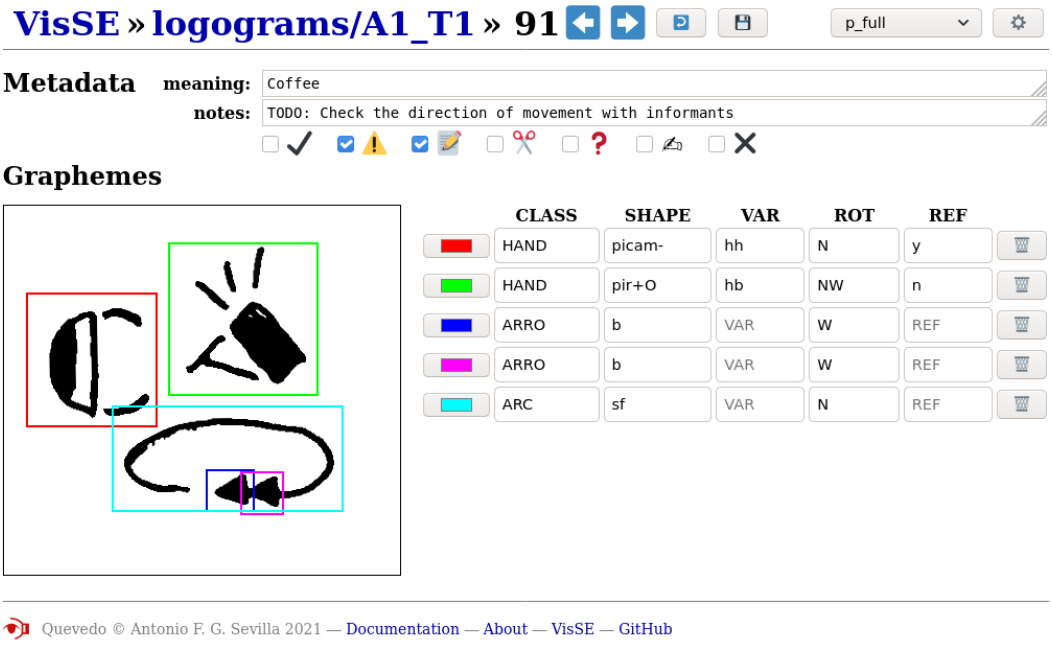

Figure 5: Annotation of a logogram using the Quevedo web interface. The sign, the same as in Figure 3, means coffee in Spanish Sign Language. The graphemes marked and annotated within the image represent the two hands and their relative location and orientation, along with their finger configuration, and the circling movement they perform. Metadata can help with the annotation process, and user-defined flags can be configured to represent task-specific messages for other annotator or reviewers. More details on the labels and their values can be found in the corpus annotation guide at `https://zenodo.org/record/6337885`.

5. Use Case

Quevedo is one result of our project "Visualizando la SignoEscritura"[14] (VisSE, "Visualizing SignWriting" in Spanish). One goal of the project was to create tools capable of automatically processing SignWriting, a graphical system for transcribing sign languages. SignWriting uses iconic symbols to represent the hands, body parts, and their movements and configurations, as the example in Figure 3 shows. Quevedo and its features have enabled us to create a system capable of automatically extracting the graphemes within a SignWriting transcription, assigning each of them a label representing their meaning. This required us to collect a corpus, publicly available at `https://zenodo.org/record/6337885` (Sevilla et al., 2022) and formatted as a Quevedo dataset. The corpus was manually and visually annotated, and a complex hierarchy of neural networks was trained on these annotations to be able to find the correct tag set for every grapheme in new instances.

For every step of this process we used Quevedo, each step guided by a few configuration lines or command line options, and the ground truth kept in the annotation files. The different steps in the process and their results can be recorded with DVC, easing experimentation and storing of measurements, as well as increasing reproducibility.

Our SignWriting data are handwritten transcriptions of Spanish Sign Language signs, organized into sets to be able to do incremental annotation. Metadata for each file store the annotation progress, doubts and errors, and possible problems in the source images to take into account. Within each logogram, the different graphemes are marked and tagged with a set of features that captures their meaning. This set of features, the annotation schema of the dataset, includes a coarse-grain class for graphemes (CLASS), a fine-grain label (SHAPE) and a possible variation (VAR). Rotation (ROT) and reflection (REF) are also specified, since they can alter the meaning of the sign. The relative location of the graphemes is marked with the bounding boxes, which are shown visually in the web interface. An example of this annotation process can be seen in Figure 5.

Splitting the grapheme labels into a coarse classification (hands, head symbols, arrows, and some others) and then a finer one has allowed us to perform automatic recognition of the graphemes in steps, instead of with one single recognizer. Storing rotation and reflection of graphemes in the tags, combined with a script to "straighten" them before recognition, reduced the number of classes of graphemes to be recognized 16-fold, de-sparsifying our data and again dividing the problem of recognition into smaller steps. The trained neural networks, each specialized in a different task, were then collected into a Quevedo pipeline, making the full automatic recognition process available as a single com-

[14] `https://www.ucm.es/visse`

mand line invocation or a few lines of glue code.

The use of domain knowledge and deterministic rules to do part of the processing was a necessary step in our research, turning a very difficult problem into a tractable one. "Dividing and conquering" is a common strategy in computer science, and is especially necessary if the amount of data available is not as extensive as the complexity of the problem requires for a purely data-based, blind approach. Not having "Big Data" at our disposal means we can not rely on ready made, single-shot solutions to our problem, but we can still use some of the algorithms coming from the state of the art. This requires thoughtful preparation and organization of data, and building custom processing sequences. Quevedo was born to help us in doing this for our SignWriting research, but is general enough that it can be useful for other similar tasks of graphical language recognition, where data is not plentiful and requires careful annotation and processing.

For the VisSE project, the results of the data collection, annotation and machine learning were integrated into a user-facing application that explains SignWriting instances[15]. The application works by using Quevedo's recognition pipeline to find the graphemes, and creating textual descriptions for each of them from the predicted tag set. While we have given a shallow overview of our SignWriting pipeline here, researchers interested in reproducing our research or extending it to new problems can find a more in-depth description in our forthcoming work "Automatic SignWriting Recognition".

The success of the VisSE project in achieving its goals serves as an initial evaluation of Quevedo as useful software for the processing of graphical languages, as well as a demonstration of its potential for solving similar problems in the future.

6. Conclusion

Graphical languages use symbols arranged in the page to convey meaning, but in contrast to the mostly linear writing systems of oral languages, the two-dimensional placing of symbols is fundamental to their decoding. To properly process and recognize images of graphical languages, we can use techniques from artificial vision, but also need the rich annotation of meaning found in natural language processing tools.

Existing software can help in many parts of the process, but none is focused on the concrete task of graphical language processing, requiring researchers to write much code to account for its unique features. Quevedo is our answer to this problem, a high-level python library and application that can help in building datasets of graphical language images, and annotating them with the necessary information for their automatic recognition and processing.

Quevedo has enabled our research into SignWriting, a complex writing system for transcribing sign languages which is a graphical language in itself. However, Quevedo is general and domain-agnostic, so it can be used for other tasks and with other datasets. It offers the researcher tools for performing the chores of dataset collection and organization, a visual and fully featured annotation interface, and functions for performing common tasks such as machine learning algorithms training. These tasks are all part of modern data science, but take time and expertise, time which is also needed for the domain-specific tasks of deciding on an annotation schema, relevant processing pipelines, and actual annotation of the data.

We have briefly shown how we use Quevedo, and given a quick primer on its usage for other researchers. There is more documentation available online, and our code is freely distributed at GitHub. We believe that there are many other graphical languages which could be processed with similar techniques to ours, and sharing our software may be a way to help other researchers do so.

7. Future Work

In the future, there are many improvements we want to make, and we are conducting lines of research that will contribute more functionality to Quevedo. We want to integrate with alternative deep learning platforms, such as TensorFlow, to be able to utilize the wide array of features they provide, as well as making the interaction of Quevedo with other software easier.

While our focus is on SignWriting, where there is still much work to be done for its fully automatic processing, we already have many ideas of languages where it might be interesting to try to apply our techniques. The examples given in the introduction, such as musical notation and UML diagrams, are only some of them.

Finally, there is current work on developing the next step in Quevedo processing. When representing graphical languages "semantically", often the chosen representation is a list of symbols with their attributes and positions. This is indeed the representation that Quevedo is currently geared to extract, and the one majorly used when dealing with SignWriting computationally. However, this representation leaves interpretation of the image meaning to the human reader. To computationally extract the semantics of graphical language images, further processing is needed, taking into account the whole context of the logograms, as well as the functional relations between the graphemes. This is our current research, and the related code is already under development in the Quevedo repository.

8. Acknowledgements

Initial development of Quevedo was part of the project "Visualizando la SignoEscritura", reference number PR2014_19/01, funded by Indra and Fundación Universia in the IV call for funding aid for research projects with application to the development of accessible technologies.

[15] `https://holstein.fdi.ucm.es/visse/` (in Spanish).

Current development and improvements are part of the project "SSL Signary: A parametric dictionary of Spanish Sign Language"[16], reference number IN[21]_HMS_LIN_0070, supported by a 2021 Leonardo Grant for Researchers and Cultural Creators from the BBVA Foundation. The BBVA Foundation accepts no responsibility for the opinions, statements and contents included in the project and/or the results thereof, which are entirely the responsibility of the authors.

9. Bibliographical References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Deng, J., Li, K., Do, M., Su, H., and Fei-Fei, L. (2009). Construction and Analysis of a Large Scale Image Ontology. Vision Sciences Society.

Kuprieiev, R., Pachhai, S., Petrov, D., Redzyński, P., da Costa-Luis, C., Rowlands, P., Schepanovski, A., Shcheklein, I., Taskaya, B., Orpinel, J., Santos, F., Gao, Sharma, A., de la Iglesia Castro, D., Zhanibek, Hodovic, D., Kodenko, N., Grigorev, A., Earl, Dash, N., Vyshnya, G., maykulkarni, Hora, M., Vera, Mangal, S., Baranowski, W., Wolff, C., and Benoy, K. (2021). DVC: Data Version Control - git for data & models.

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. (2015). Microsoft coco: Common objects in context.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, et al., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Redmon, J. (2013–2016). Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`.

Smith, R. W. (2013). History of the Tesseract OCR engine: what worked and what didn't. In *Document Recognition and Retrieval XX*, volume 8658, pages 1–12. SPIE, February.

Sutton, V. and Frost, A. (2008). *SignWriting: sign languages are written languages!* Center for Sutton Movement Writing.

Technical Committee: ISO/IEC JTC 1 Information technology. (2012). Information technology – Object Management Group Unified Modeling Language (OMG UML) – Part 2: Superstructure. Standard, International Organization for Standardization.

10. Language Resource References

Sevilla, A. F. G., Lahoz-Bengoechea, J. M., and Díaz Esteban, A. (2022). VisSE corpus of Spanish SignWriting. `https://zenodo.org/record/6337885`.

[16] `https://www.ucm.es/signariolse`