# DIVA-DAF: A Deep Learning Framework for Historical Document Image Analysis


Lars Vögtlin
lars.voegtlin@unifr.ch
University of Fribourg
Fribourg, Switzerland

Anna Scius-Bertrand
anna.scius-bertrand@unifr.ch
University of Fribourg
Fribourg, Switzerland

Paul Maergner
paul.maergner@unifr.ch
University of Fribourg
Fribourg, Switzerland

Andreas Fischer
andreas.fischer@unifr.ch
University of Fribourg
Fribourg, Switzerland

Rolf Ingold
rolf.ingold@unifr.ch
University of Fribourg
Fribourg, Switzerland



## ABSTRACT

Deep learning methods have shown strong performance in solving tasks for historical document image analysis. However, despite current libraries and frameworks, programming an experiment or a set of experiments and executing them can be time-consuming. This is why we propose an open-source deep learning framework, DIVA-DAF, which is based on PyTorch Lightning and specifically designed for historical document analysis. Pre-implemented tasks such as segmentation and classification can be easily used or customized. It is also easy to create one's own tasks with the benefit of powerful modules for loading data, even large data sets, and different forms of ground truth. The applications conducted have demonstrated time savings for the programming of a document analysis task, as well as for different scenarios such as pre-training or changing the architecture. Thanks to its data module, the framework also allows to reduce the time of model training significantly.



## CCS CONCEPTS

• **Software and its engineering** → Object oriented frameworks.

## KEYWORDS

deep learning framework, document image analysis, historical documents, deep neural networks

**ACM Reference Format:**

Lars Vögtlin, Anna Scius-Bertrand, Paul Maergner, Andreas Fischer, and Rolf Ingold. 2023. DIVA-DAF: A Deep Learning Framework for Historical Document Image Analysis. In *7th International Workshop on Historical Document Imaging and Processing (HIP '23), August 25–26, 2023, San Jose, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3604951.3605511


## 1 INTRODUCTION

Automatically analyzing collections of historical documents provides strong support for the preservation of our cultural heritage. Although great progress has been made in recent years, this field of research remains a difficult challenge, especially due to the high variability of document collections both in terms of form and content [8]. Deep learning methods have shown a strong potential for historical document image analysis, achieving state-of-the-art results for different tasks ranging from layout analysis over handwriting recognition to information retrieval.

In order to use deep learning more quickly and efficiently, several libraries were created (e.g., Tensorflow, Keras, PyTorch, ...). Such libraries are built in a very open and general fashion to allow good programmers to take advantage of the full potential of this new technology. However, due to their generality, these libraries have a steep learning curve. Additionally, the user usually has to take care of the whole hardware orchestration, such as moving data to the GPU or aggregating certain information across a number of devices. Therefore, software frameworks built on top of the general deep learning libraries may significantly facilitate the application of deep learning to specific types of data and tasks.

In the context of historical document analysis, a deep learning framework must be able to deal with large images and support different ground truth formats. Furthermore, one of the biggest challenges for historical documents is the lack of training data. It is often necessary to use different learning strategies such as transfer learning, self-learning, or data generation. Conducting scientific experiments with advanced learning mechanisms requires a high flexibility from the deep learning framework, to be able to iteratively test different parameters and configurations without having to reprogram the whole experiment. Moreover, a prerequisite of any experiment is its reproducibility.

In this paper, we introduce a new deep learning framework, DIVA-DAF[1], which is specifically designed for conducting experiments in the domain of image analysis for historical documents. The framework is based on PyTorch Lightning, which allows us to benefit from its flexibility and hardware management. We have added features to conduct document analysis experiments with a view to reproducibility, maintainability, efficiency, and increased flexibility. These features allow, among others: rapid prototyping, faster runtime, use of transfer learning and self-learning, simplified change or swap of network parts, adding own code with unit tests, using external libraries, simplified change (without reprogramming) of networks, tasks, and datasets in an experiment, and keeping track

[1] https://github.com/DIVA-DIA/DIVA-DAF




*HIP '23, August 25–26, 2023, San Jose, CA, USA*





ACM ISBN 979-8-4007-0841-1/23/08…$15.00
https://doi.org/10.1145/3604951.3605511

of previous experiments. These features make our framework a powerful tool to conduct experiments in the context of historical document image analysis.

DIVA-DAF supports all types of deep neural networks and learning strategies, including supervised, semi-supervised, and unsupervised learning. Therefore, all typical document image analysis tasks can be implemented with our framework, such as layout analysis, line segmentation, keyword spotting, transcription alignment, handwriting recognition, etc. At the moment, two tasks are already implemented and readily available: segmentation and classification. For example, to perform semantic segmentation for layout elements in a historical document, it is sufficient to indicate the data and ground truth folders in a configuration file. In this configuration file, it is also possible to change the network model, loss function, optimizer, and evaluation metrics, among others. In less than 5 minutes, a custom experiment can be configured and started.

In the remainder of this paper, we provide an overview of related work in Section 2. Then, we introduce the DIVA-DAF framework and describe its features in Section 3. Afterwards, we present a case study using the framework in Section 4. Finally, we provide some conclusions and an outlook to future work in Section 5.

## 2 RELATED WORK

In our literature research, we came across different end-to-end frameworks for Computer Vision and Document Image Analysis with which we share the general motivation as well as ideas.

Transkribus is a well-known platform designed to automatically transcribe historical documents. The platform offers the possibility to train a model with specific data and also provides trained text recognition models ready to be used. An interface is available. But the platform is not open source and provides only a restricted number of document analysis tasks.

Another well-known platform for automatic transcription of historical documents is eScriptorium [12], which is open source. The text recognition system is based on Kraken [11]. Other models can be integrated into the platform. But as Transkribus, eScriptorium is also limited in the number of document analysis tasks covered.

Chainer [18] provides a wide range of Deep Learning (DL) models for researchers in a flexible and intuitive fashion. The framework's focus is high-performance and distributed training, which is achieved with the help of standard Python libraries. But the framework no longer gets updates (last release June 2022), it misses an easy definition of an experiment, and networks can not be loaded in parts. Additionally, as it is based on plain NumPy, there is a lack of compatibility with other DL-frameworks.

Orhei et al. [13] introduced an End-to-End CV Framework (EECVF) to tackle the problem of creating a true end-to-end Machine Learning (ML) platform that allows combining DL approaches together with classical Pattern Recognition (PR) methods. Their platform is constructed to be easily usable for research and educational purposes. It uses multiple configuration files to define the behavior of the different parts without the need to write code. The biggest problem with this framework is that it is no longer available, does not support different hardware accelerators, has limited logging, and no flexible loading of a network's weights.

Goyal et al. [9] from Meta Research created a PyTorch-based framework named VISSL for self- and unsupervised pretraining of neural networks for natural images. The main idea of this framework is to provide a fast and easy way to pretrain neural networks with natural images in a self-supervised fashion with the help of a configuration system. Additionally, hardware acceleration with GPUs, logging with Tensorboard, and a large variety of preimplemented methods and datasets are a part of it. This project also has some issues: It is no longer maintained (last release November 2021), it is not possible to fine-tune a network on some final tasks, logging is limited to Tensorboard, and introducing new networks, functionalities, or other parts is very tedious work.

DeepDIVA was introduced by Alberti et al. [2, 3] as an out-of-the-box deep learning framework for Computer Vision (CV). The focus of the framework was to provide reproducible experiments that the user can redefine based on existing networks, datasets, and parts, but they have the possibility to add their own. It also provides e.g., hardware acceleration with GPUs, logging with Tensorboard, reproducibility by versioning the code, and different visualizations of the data. However, there are several problems with this framework: Introducing custom network parts is difficult as the framework is not built in a modular fashion, all parameters are handed over via Command Line Interface (CLI), which makes it difficult to read, the weight loading functionalities are limited, and logging is only provided for Tensorboard.

A library approach was taken by Shen et al. [15] with their LayoutParser. Their goal is to simplify the construction of Deep Learning workflows within the Document Image Analysis (DIA) domain. They additionally provided a platform to share models, code, and weights. This project seems no longer actively supported as their last change is from August 2022.

Falcon et al. [7] started 2019 the modular PyTorch-based general DL framework PyTorch-Lightning (PL). It focuses on rapid prototyping, wide hardware integration, and maximal flexibility. Additionally, it takes advantage of the large ecosystem, providing implementations for metrics, models, data modules, and other state-of-the-art functionalities. As the framework is not focusing on CV or DIA, it lacks the support to handle large images and does not provide an easy way to create an experimental setup.

To have an overview of the lack of modularity and flexibility of the different frameworks, see Table 1. With DIVA-DAF, we added all this modularity and flexibility of the different categories that are relevant for deep learning experiments in the context of historical documents. They are further introduced in the following section.

## 3 DIVA-DAF - DOCUMENT ANALYSIS FRAMEWORK

To be able to conduct scientific experiments on document image analysis, our framework has the following characteristics: flexibility, efficiency, reproducibility, and maintainability. The framework is mainly designed for experienced programmers in document image analysis, but thanks to a configuration system, non-experts with few programming skills can also create, launch, and interpret experiments. In this section, we introduce the deep learning framework DIVA-DAF and explain its main attributes.

**Table 1: Modularity of Different Frameworks in Different Categories**

| Name | DeepDIVA | VISSL | Chainer | EECVF | PL | DIVA-DAF |
|---|---|---|---|---|---|---|
| Input | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Output | ✓ | ✗ | ✓ | (✗) | ✓ | ✓ |
| Hyper-parameters | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Network | ✗ | ✗ | ✗ | (✗) | ✗ | ✓ |
| Monitoring | ✗ | ✓ | ✓ | (✗) | ✓ | ✓ |
| Evaluation | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ |
| Reproducibility | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Experimental setup | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ |

## 3.1 Flexibility

The framework consists of several components following an object-oriented programming paradigm. The general architecture is presented in Figure 1.

Each component is independent and can be easily changed. Like classical frameworks, data is loaded into a data module. But unlike other platforms, the data module manages the different datasets needed for each stage (training, validation, testing, prediction) in a modular way. Furthermore, it calculates data statistics and defines special data handling, such as data augmentation and transformations. Besides, the data module is able to load large images thanks to two strategies: scaling down the image (adapting also the ground truth) and patch-based approaches. Different ground truth formats can already be easily handled: images (color encoded, index encoded, channel encoded) and classes based on folder structures.

Compared to PyTorch Lightning, the LightningModule is separated into two components: the model, which defines the neural network architecture, and the task, which describes the task to be solved. By defining these components independently, the same task can be solved using different models, or the same model can be used to solve different tasks.

The model specifies the neural network architecture by defining the backbone and the header. The backbone acts as the encoder part of the network, and the header as the classifier. By defining these two parts separately, the framework can save them independently and combine them with other backbones or headers.

The task defines the workflow during training, validation, testing, and prediction. It requires four inputs: a loss function, an optimizer, metrics, and a model. All these components can be easily customized by the user. Also, it produces the test output and provides the needed method to bring the network output into a specific loss or metric format.

The trainer connects the different components of our framework and runs them. It executes the different stages - training, validation, testing, and prediction - and runs the neural network. It is responsible for initializing the different hardware devices and moving data and models to the correct device. The default implementation of PyTorch Lightning is used, but users could also exchange or modify this part if required. Trainers are also connected to a logger and a callback component which will be described in subsection 3.3, and a plugin component which enables changing the behavior of the trainer, e.g., custom precision or cluster environment implementations.

## 3.2 Efficiency

The first obstacle to using a new framework is the difficulty related to its installation.

DIVA-DAF is easy to set up: The user clones the code from the GitHub repository (https://github.com/DIVA-DIA/DIVA-DAF) and creates a new Python environment (e.g., Anaconda[1] environment) based on the shipped requirement file. This requirement contains all dependencies with the corresponding versions to run the framework. If a user wants to run the framework with GPU, TPU, HPU, MPS, or IPU support, the appropriate drivers and the correct PyTorch support packages (CUDA, ROCm, etc.) must be installed.

Thanks to the modular structure of the framework, users can do rapid prototyping. The user can easily combine existing modules into a new experiment. The hyperparameters of an existing one can be changed without writing a single line of code. To create or introduce new modules and swap them out with existing ones, just a few lines of code are needed, as the framework provides templates for the different modules. An example is given in Section 4.

By using the full potential of the underlying PyTorch Lightning framework, DIVA-DAF takes full advantage of any hardware provided by the host system, optimized data-loading strategies, and distributed computing. With an efficient implementation of our datasets, we were able to further reduce the runtime of the experiments.

## 3.3 Reproducibility

Another important attribute of a framework is its ability to create reproducible research experiments. To make each experiment as reproducible as possible, we store the configuration file alongside the results and network weights of each run in its output directory. It also saves the seed used to initialize the pseudo-random generators used during training to initialize the neural network weights and other environmental information. Using the configuration file with this seed, anyone can quickly reproduce published results with this framework.

To keep track of experiments, we use the logging functionality of PyTorch Lightning. They provide the most common loggers like Weights and Biases [5] or Tensorboard. The framework allows using multiple loggers simultaneously. Besides using a cloud-based logger, like Weights and Biases, the user can also use a local logger like the CSV logger. The CSV logger writes all the logging information into the local experiment folder for later use. Logging is not limited to scalar data like metrics or loss information. It is also possible to

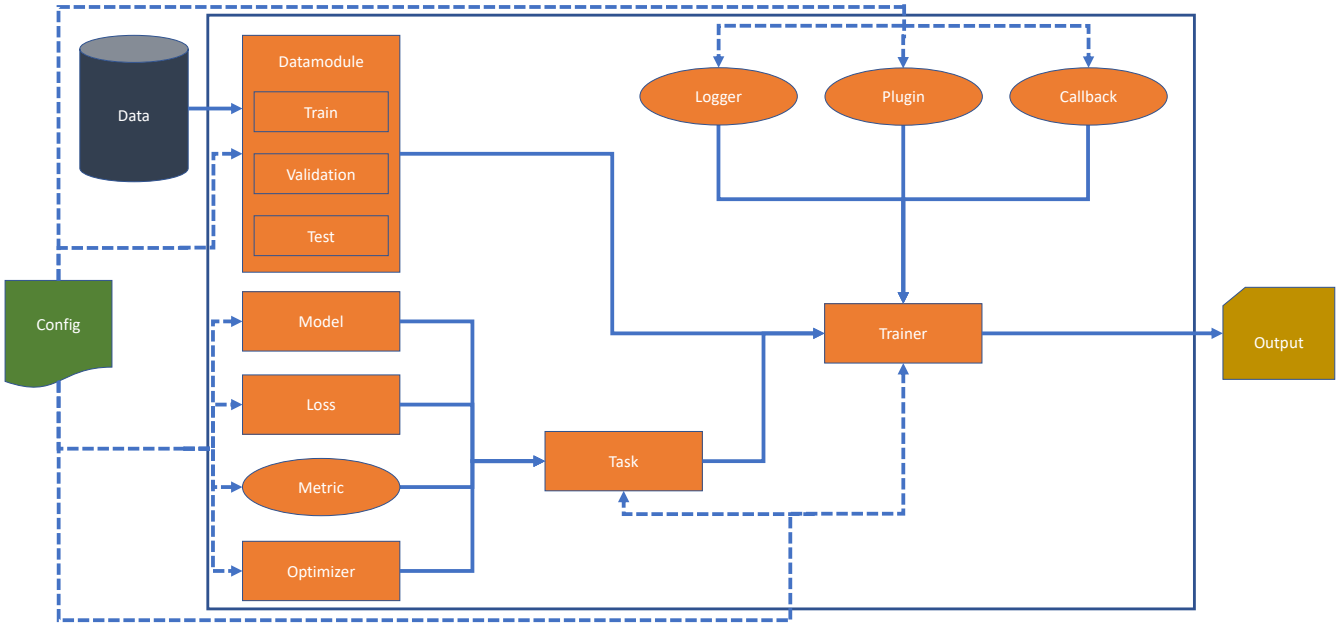



**Figure 1: The module schema of DIVA-DAF. Rectangles represent required components, ovals represent optional components, and green is the configuration.**

log figures, images, or histograms, but each logger needs to do this individually. As with all the other modules of our framework, users can also implement their own logger or adapt an existing one.

### 3.4 Maintainability

Maintainability is a key concept to create a long-lasting framework. DIVA-DAF provides two maintainability factors: injecting external code and intercompatibility. Users can inject code via callbacks provided by the underlying PL framework. Callbacks hook into predefined methods and can be used at each stage of the experiment. The main advantage of callbacks is that the user does not have to change the core code of the framework. Hence, it provides extendability without the cost of damaging the integrity of the system.

Additionally, DIVA-DAF uses GitHub actions to provide Continuous integration (CI). For every change in the framework, the different modules get extensively tested with unit tests, and the code quality (duplication, bugs, complexity) gets checked. This ensures a good code base and gives the user the possibility to check if these changes break anything in the framework.

Thanks to its modularity and compatibility with the PyTorch ecosystem, any module from PyTorch-based libraries can be integrated into DIVA-DAF.

## 4 APPLICATIONS

In this section, we compare the programming time and execution time of a document analysis experiment using DIVA-DAF and PyTorch-Lightning (PL).

### 4.1 Methods

To compare the programming time, a baseline experiment was performed and then broken down into three scenarios.

The baseline experimentation consists of semantic segmentation for layout elements in historical documents. Each pixel of an input image gets assigned one of the predefined classes. An experienced PL programmer timed each of the development steps using the two different frameworks.

Scenario 1 - S1: Pre-training / transfer learning: When the volume of training data is too small, pre-training or transfer learning can improve the performance of the network. In this case, the first $n$ layers of an already pre-trained U-Net [14] (here n=3) were loaded into a randomly initialized U-Net, and afterward fine-tuned on the additional data.

Scenario 2 - S2: Comparison of the network architecture: Often in research, it is necessary to compare several networks for the same task on identical data. Here the task is to replace the U-Net with DeepLabV3 [6], an architecture already implemented in Torchvision.

Scenario 3 - S3: Visual control during training: training a network can lead to a "black box" effect. Visualizing an intermediate result during training can be useful for understanding the behavior of the network. The task here is to be able to save n images randomly from the validation set (here 1 image).

To compare the execution time, we replicated the experiment by Studer et al. [17] (same network running on the same hardware), with the difference that we performed the experiment with DIVA-DAF.

## 4.2 Data

The dataset is the Codex Bodmer 55 of the DIVA-HisDB [16] dataset (see Fig. 2). The dataset contains 20 pages for training, 10 pages for validation, and 10 pages for testing. Each page has a dimension of 4872×6496 pixels with a resolution of 600 dpi. Each pixel belongs to one of 8 classes (background, main text body, decoration, comment, main text body + comment, main text body + decoration, comment + decoration, main text body + decoration + comment).

## 4.3 Results

The baseline implementation is split into three parts: data loading, network, and execution. Data loading includes the dataset, calculating statistics on the training data, normalizing the data, and creating a data module. The network part is just the implementation of the network and its behavior in the different stages. Last, in the execution part, we combine the two parts from above into a runnable experiment. For the time we used to implement this and its code duplication, see Table 2.

The time to implement the baseline experiment in PL is nearly 15x longer compared to the same experiment in DIVA-DAF. To load the data, which is the DIVA-HisDB [16] format, it just takes a few lines of YAML (see Listing 1). In our main experiment configuration, we need to specify the data module we want to use (`_target_`), the path to the data (`data_dir`), how big our crops should be (`crop_size`), and the batch size (`batch_size`).

In contrast, in PL, we have to implement the whole data loading logic, as well as take care of calculating statistics, multi-device training, and applying transformations. These parts are very crucial to have a correctly working experiment and so take a lot of time to implement.

**Listing 1: The config describing the data module**

```
datamodule:
    _target_: src.datamodules.DivaHisDB.
        datamodule_cropped.
        DivaHisDBDataModuleCropped

    data_dir: /net/research-hisdoc/datasets/
        semantic_segmentation/
        datasets_cropped/CB55
    crop_size: 256
    batch_size: 16
```

The other part that takes more time in plain PL is the implementation of the network. We can take advantage of the U-Net class from the Torchvision library, but we still have to implement the behavior of the network during the training, validation, and testing stages. In DIVA-DAF, we do not have to do that because the network's behavior is defined in the task and not the network.

To implement the execution part of the scenario, the time difference is not significant. In PL, it takes a few lines of code to create a `Trainer` object and hand it over to the data and the network. In DIVA-DAF, we have to adapt the experiment configuration.

For the first scenario, we can use the preimplemented functionality of DIVA-DAF, where we can define in the configuration the layers of the network we want to load. The same in PL takes more time as you need to filter out the layers we want to use from the checkpoint file and load them into the network. For an experienced PL programmer, this is not a complicated but a time-consuming task.

**Table 2: Programming time in minutes**

| Tasks | PyTorch L. | DIVA-DAF |
|---|---|---|
| Data loading | 150 | 2 |
| Network | 30 | 5 |
| Execution | 10 | 5 |
| S1. Pre-training | 15 | 2 |
| S2. Comparing network | 20 | 2 |
| S3. Visualizing | 25 | 20 |

In the second scenario, we use the Torchvision library again to apply a DeepLabv3 model with a ResNet50 [10] backbone to our task. As the task stays the same, in DIVA-DAF, we have to create a config file (see Listing 2) for the new network and adapt the experiment. In PL, we can take advantage of the U-Net implementation and copy the code defining the behavior of the network during the different stages. This creates code duplication, which makes the code harder to maintain and adapt.

**Listing 2: The config for the DeepLabv3 network with a resnet50 backbone**

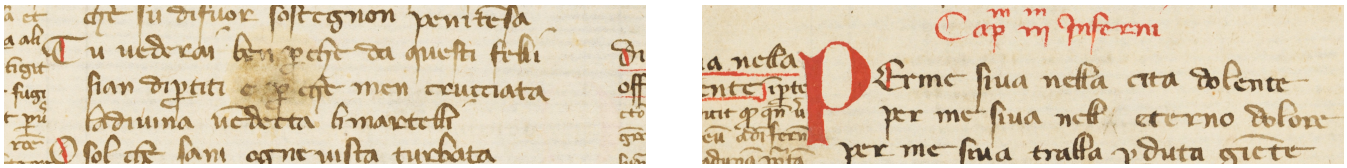
```
_target_: torchvision.models.segmentation.
    deeplabv3_resnet50
num_classes: ${datamodule:num_classes}
```

Copying the code in the PL, implementation becomes a problem in the third scenario, as we want to change the behavior of the network in the validation stage to save a random image. We have to copy the code again into both network implementations, which creates more code duplication and increases the complexity. In DIVA-DAF, the code has just to be changed in the task class. This could, in both cases, also be solved with the help of a callback, which would reduce code duplication but takes our programmer still less time to implement in DIVA-DAF than in PL.

For the execution time comparison (see Table 3), we used the same hardware and hyperparameter as Studer et al. [17], as well as the SegNet [4] and DeepLapv3 [6] architecture. The hardware is a server with 4 × NVIDIA 1080 GTX with 8GB of GPU memory each, an Intel i7-5960X CPU, and 64 GB of RAM. The hyperparameters are available here [2].

The experiments conducted with DIVA-DAF were significantly faster compared to the implementation of Studer et al. For both architectures, we achieved similar results. The performance difference of SegNet [4] is probably due to different default parameters not mentioned by Studer et al. However, we have time savings for DeepLabv3 [6] and SegNet of more than 55% and 45%, respectively. We think this is caused by the efficient data loading in DIVA-DAF (from the file system but also into the GPU) and the improvements in driver technology.

[2] https://bit.ly/2I8c3dX



(a) CB55, p. 25r (b) CB55, p. 5v

**Figure 2: Sample pages of the medieval manuscripts Codex Bodmer 55 of DIVA-HisDB.**

**Table 3: Results of semantic segmentation on the test set of DIVA-HisDBs CB55. All our networks were trained for 50 epochs. All experiments are conducted on the same hardware.**

| Authors | Year | Model | Runtime | mIoU[%] |
|---|---|---|---|---|
| [17] | 2019 | SegNet | ~8h | 86.90 |
| [17] | 2019 | DeepLabV3 | ~8h | 92.90 |
| Ours | 2023 | SegNet | ~4.5h | 92.61 |
| Ours | 2023 | DeepLabv3 | ~3.5h | 93.04 |

## 5 CONCLUSION AND FUTURE WORK

In this paper, we introduce DIVA-DAF. It is an open-source PyTorch-Lightning-based deep learning framework designed to create rapid prototypes and reproducible experiments for the historical document analysis community.

The framework offers pre-implemented tasks that are easily adaptable, including segmentation, classification, and object detection. It is also possible to implement custom tasks and data modules in a straightforward way due to the framework's abstract classes. As shown in the application part, DIVA-DAF allows users to gain efficiency during implementation as well as model execution.

However, the framework has certain functional limitations like conducting multi-runs within the framework, doing hyperparameter optimization, running tasks with multiple headers or losses, and downloading datasets in an automatic fashion.

To encourage a larger public to use the framework, a user interface could be developed. To improve the framework, we envisage implementing new tasks in document analysis, adding new networks, integrating new ground truth formats, and improving its documentation, which this paper is part of. To further support the users in analyzing their results, it would be interesting to add classification activation maps and filter visualization techniques to the framework.

## REFERENCES


[1] 2020. Anaconda Software Distribution.

[2] Michele Alberti, Vinaychandran Pondenkandath, Lars Vögtlin, Marcel Würsch, Rolf Ingold, and Marcus Liwicki. 2019. Improving Reproducible Deep Learning Workflows with DeepDIVA. In *2019 6th Swiss Conference on Data Science (SDS)*. 13–18.

[3] Michele Alberti, Vinaychandran Pondenkandath, Marcel Würsch, Rolf Ingold, and Marcus Liwicki. 2018. DeepDIVA: A Highly-Functional Python Framework for Reproducible Experiments. In *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 423–428.

[4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. 2017. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence* 39, 12 (2017), 2481–2495.

[5] Lukas Biewald. 2020. Experiment Tracking with Weights and Biases.

[6] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. 2017. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587* (2017).

[7] William Falcon and The PyTorch Lightning team. 2019. PyTorch Lightning. https://doi.org/10.5281/zenodo.3828935

[8] Andreas Fischer, Marcus Liwicki, and Rolf Ingold (Eds.). 2020. *Handwritten Historical Document Analysis, Recognition, and Retrieval — State of the Art and Future Trends.* World Scientific.

[9] Priya Goyal, Quentin Duval, Jeremy Reizenstein, Matthew Leavitt, Min Xu, Benjamin Lefaudeux, Mannat Singh, Vinicius Reis, Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Ishan Misra. 2021. VISSL. https://github.com/facebookresearch/vissl.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[11] Benjamin Kiessling. 2019. Kraken-an universal text recognizer for the humanities. In *ADHO, Éd., Actes de Digital Humanities Conference.*

[12] Benjamin Kiessling, Robin Tissot, Peter Stokes, and Daniel Stökl Ben Ezra. 2019. eScriptorium: an open source platform for historical document analysis. In *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*, Vol. 2. IEEE, 19–19.

[13] Ciprian Orhei, Silviu Vert, and Muguras Mocofan. 2021. End-To-End Computer Vision Framework: An Open-Source Platform for Research and Education. *Sensors* 21, 11 (2021), 3691.

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention.* Springer, 234–241.

[15] Zejiang Shen, Ruochen Zhang, Melissa Dell, Benjamin Charles Germain Lee, Jacob Carlson, and Weining Li. 2021. LayoutParser: A unified toolkit for deep learning based document image analysis. In *Document Analysis and Recognition–ICDAR 2021: 16th International Conference, Lausanne, Switzerland, September 5–10, 2021, Proceedings, Part I 16.* Springer, 131–146.

[16] Foteini Simistira, Mathias Seuret, Nicole Eichenberger, Angelika Garz, Marcus Liwicki, and Rolf Ingold. 2016. DIVA-HisDB: A Precisely Annotated Large Dataset of Challenging Medieval Manuscripts. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR).* 471–476.

[17] Linda Studer, Michele Alberti, Vinaychandran Pondenkandath, Pinar Goktepe, Thomas Kolonko, Andreas Fischer, Marcus Liwicki, and Rolf Ingold. 2019. A Comprehensive Study of ImageNet Pre-Training for Historical Document Image Analysis. In *2019 International Conference on Document Analysis and Recognition (ICDAR).* 720–725.

[18] Seiya Tokui, Ryosuke Okuta, Takuya Akiba, Yusuke Niitani, Toru Ogawa, Shunta Saito, Shuji Suzuki, Kota Uenishi, Brian Vogel, and Hiroyuki Yamazaki Vincent. 2019. Chainer: A Deep Learning Framework for Accelerating the Research Cycle. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19).* Association for Computing Machinery, New York, NY, USA, 2002–2011.