

DAL-ART: Deep Active Learning Tool for Multimodal Image Analysis Applied to Crack Detection in Paintings

Sebastiaan Verplancke

Student number: 01604354

Supervisors: Prof. dr. ir. Aleksandra Pizurica, Prof. dr. Maximiliaan Martens
Counsellors: Yoann Arhant, Srdan Lazendic

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2022-2023

DAL-ART: Deep Active Learning Tool for Multimodal Image Analysis Applied to Crack Detection in Paintings

Sebastiaan Verplancke

Student number: 01604354

Supervisors: Prof. dr. ir. Aleksandra Pizurica, Prof. dr. Maximiliaan Martens
Counsellors: Yoann Arhant, Srdan Lazendic

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2022-2023

PERMISSION OF USE OF LOAN

The author gives permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In all cases of other use, the copyright terms have to be respected, in particular with regard to the obligation to state explicitly the source when quoting results from this master dissertation.

Sebastiaan Verplancke
25th Mai 2023

PREFACE

With the end of this master's dissertation I am also writing an end to my book as a Computer Science Engineering student. I have gained invaluable knowledge and skills during my time in this program that I will carry with me throughout my career. As I move on to the next chapter of my life, I am excited to apply what I have learned and continue to grow as a professional in the field. I am grateful for the opportunity to pursue my passion for technology and look forward to contributing to the ever-evolving world of computer science.

In this preface, I would be delighted to take the opportunity to express my gratitude for those who were willing to offer me a helping hand to accomplish my thesis:

First and foremost, I would like to thank my supervisors: Prof. dr. ir. Aleksandra Pižurica and Prof. dr. Maximiliaan Martens for their advice and guidance, which allowed me to bring this thesis to an even better level.

Secondly I would like to thank my counsellors: Srdan Lazendić and Yoann Arhant without whom my research would be impracticable in the first place and for sharing their point of view, which lead to interesting conversations broadening my field of knowledge.

Furthermore, I want to thank Roman Sizyakin for his previous research to build upon and Philippe Serbruyns for providing the technical capacity to make the work come to life.

Last but not least I want to thank my family, friends and girlfriend for the proofreading and all other kinds of support they offered to me.

I hope this thesis may help extend the research space, and I hope you enjoy reading it.

Sebastiaan Verplancke

ACKNOWLEDGEMENTS

The DAL-ART deep active learning tool resulted from long-term research on crack detection in paintings using deep learning in the research group GAIM at Ghent University, led by Prof. Aleksandra Pizurica. At the core of this tool is the method for detecting features of interest (cracks) from multimodal images, developed in the PhD thesis of Roman Sizyakin: “Deep learning methods for crack detection and image restoration with application to digitized paintings”, Ghent University (2022), supervised by Prof. A. Pižurica. The current web-based application allowing active learning was developed in the Master thesis of Sebastiaan Verplancke: “DAL-ART: Deep Active Learning Tool for Detecting Features of Interest in Images with Application to Digital Painting Analysis”, Ghent University 2023, promoters Prof. A. Pižurica and Prof. Max Martens, supervisors Yoann Arhant and Srđan Lazendić. Hosting of the web-based tool was made possible with the help of Philippe Serbruyns, system administrator at the Department TELIN of Ghent University.

Team

Prof. Aleksandra Pižurica, PI, Scientific leader and coordinator of the project.

Dr. Roman Sizyakin, Doctoral research on deep learning for crack detection in paintings using multimodal images (2018-2022). PhD thesis: “Deep learning methods for crack detection and image restoration with application to digitized paintings”, Ghent University, 2022 (Promotor A. Pizurica).

Sebastiaan Verplancke, Master thesis research and developing DAL-ART tool: deep active learning tool and web-based application. Master thesis: DAL-ART: Deep Active Learning Tool for Detecting Features of Interest in Images with Application to Digital Painting Analysis, Ghent University, 2023 (Promoters: A. Pizurica and M. Martens).

Yoann Arhant, Master thesis supervisor and contributor to DAL-ART tool: deep active learning design and web-based application.

Srđan Lazendić, Master thesis supervisor and contributor to DAL-ART tool: deep active learning design and web-based application

Prof. Max Martens, Inputs and feedback from art-historical point of view, user requirements for the interface, analysis of the results

Philippe Serbruyns, System administrator Department TELIN@UGent, inputs and advice for the web-based application, enabled hosting the tool.

GAIM@UGent

DAL-ART: Deep Active Learning Tool for Multimodal Image Analysis Applied to Crack Detection in Paintings

Sebastiaan Verplancke

Supervisors: Prof. dr. ir. Aleksandra Pižurica, Prof. dr. Maximiliaan Martens

Counsellors: Yoann Arhant, Srđan Lazendić

Master's dissertation submitted in order to obtain the academic degree of

Master of Science in Computer Science Engineering

Academic year 2022-2023

Abstract

The digitization of artworks has led to the growing importance of digital painting analysis, particularly in crack detection for restoration and conservation treatments. In the context of the ongoing restoration project of the Ghent Altarpiece since 2012, the manual identification of crack detection and other defects for damage mapping proves to be a time-consuming task. To address this challenge, the GAIM research team has developed several State-Of-The-Art deep learning methods that can automatically detect cracks in digital paintings. While these models deliver excellent performance when trained on the adequate datasets with sufficient labelled data, applying them on new paintings is always challenging. The primary objective of this thesis is to develop a deep active learning tool that can continuously learn and improve from user interactions. Moreover, we want this tool to be easily accessible by the users as a web-application, allowing robust performance. We believe that our tool named DAL-ART can be utilized by art restorers in the future to actively improve the current models. To this end, a user interface has been designed and developed to facilitate manual annotations by the user, enabling the model to be actively learned. Several active learning and retraining strategies were analyzed in this study. The obtained results not only highlight the potential of active learning strategies for crack detection but also demonstrate the possibility of extending these strategies to other areas of digital image processing. The findings of this research open avenues for future applications of active learning techniques in various domains of art restoration and digital painting analysis.

Index Terms — digital painting analysis, restoration, conservation, active learning, deep active learning, DAL-ART, CNN, U-Net

DAL-ART: Deep Active Learning Tool for Multimodal Image Analysis Applied to Crack Detection in Paintings

Sebastiaan Verplancke

Supervisors: Prof. dr. ir. Aleksandra Pižurica, Prof. dr. Maximiliaan Martens

Counsellors: Yoann Arhant, Srdan Lazendić

Abstract—The digitization of artworks has led to the growing importance of digital painting analysis, particularly in crack detection for restoration and conservation treatments. In the context of the ongoing restoration project of the Ghent Altarpiece since 2012, the manual identification of crack detection and other defects for damage mapping proves to be a time-consuming task. To address this challenge, the GAIM research team has developed several State-Of-The-Art deep learning methods that can automatically detect cracks in digital paintings. While these models deliver excellent performance when trained on the adequate datasets with sufficient labelled data, applying them on new paintings is always challenging. The primary objective of this paper is to develop a deep active learning tool that can continuously learn and improve from user interactions. Moreover, we want this tool to be easily accessible by the users as a web-application, allowing robust performance. We believe that our tool named DAL-ART can be utilized by art restorers in the future to actively improve the current models. To this end, a user interface has been designed and developed to facilitate manual annotations by the user, enabling the model to be actively learned. Several active learning and retraining strategies were analyzed in this study. The obtained results not only highlight the potential of active learning strategies for crack detection but also demonstrate the possibility of extending these strategies to other areas of digital image processing. The findings of this research open avenues for future applications of active learning techniques in various domains of art restoration and digital painting analysis.

Index Terms—digital painting analysis, restoration, conservation, active learning, deep active learning, DAL-ART, CNN

I. INTRODUCTION

Digital painting analysis is the process of examining and evaluating digitized paintings to automatically discover features of interest. The ability to detect these features can have significant implications for the analysis and understanding of paintings. It can aid in authenticating the origins of a painting and detect potential instances of plagiarism or copying, as well as preserving the integrity of the artwork [1]. One particular challenge here is the accurate and efficient detection of craquelure, which is the web of cracks that forms on the surface of paintings over time resulting in permanent damage to valuable artworks like the Ghent Altarpiece.

Created by the Early Dutch speaking painters Hubert and Jan van Eyck in the mid-15th century, the Ghent Altarpiece - also known as the Adoration of the Mystic Lamb - is an

intricate masterpiece of European art. The altarpiece took approximately seven years to complete and can be found in St Bavo's Cathedral in Ghent, Belgium. It is regarded as one of the world's most prized treasures and is particularly noteworthy for being the "first major oil painting" [2]. The Ghent Altarpiece, shown in Figure 1, is currently undergoing a major restoration campaign where crack detection has proven to be very useful in assisting art restorers during the restoration and conservation treatments.

The research group GAIM, at Ghent University, has an extensive expertise in the field of digital image processing and in particular in digitized paintings analysis. A series of important results contributing to the preservation of the Ghent Altarpiece has been published during the last decade [3]–[5], starting from the original article of prof. Aleksandra Pižurica and her student Tijana Ružić. [6] as a joint work with Bruno Cornelis [3]. The research group addressed a wide range of problems, with most of the results related to the automatic detection of cracks in digitized paintings were obtained by Roman Sizyakin whose publications led to a PhD thesis [7] in 2022 in research group GAIM at Ghent University. In this PhD study, the researchers presented models that demonstrate exceptional performance when trained on certain parts of the Ghent Altarpiece. However, when it comes to applying these models to new paintings or different Panels from the Ghent Altarpiece, it poses a significant challenge. To overcome that challenge, this paper introduces the DAL-ART tool that



Figure 1: The Ghent Altarpiece, Hubert and Jan Van Eyck, completed in 1432 [2].

facilitates effortless and efficient deep active learning. The tool is hosted on a UGent server and can be accessed through the following URL: <https://dal4art.ugent.be/>.

The paper is structured in the following manner: Section II provides the essential information required for the remainder of the paper including the SOTA crack detection and an review of deep active learning. The problem formulation is presented in Section III where we dissect the requirements of the DAL-ART tool. The development and uses of DAL-ART tool are outlined in Section IV. In Section V, we examine the quantitative and qualitative experimental results obtained on different patches of the Ghent Altarpiece. Finally, Section VI presents the concluding remarks of this paper.

II. PRELIMINARIES

A. State-Of-The-Art crack detection

The SOTA in crack detection is build upon Convolutional Neural Networks (CNNs). CNNs are a type of deep learning algorithm specifically designed for analyzing visual data such as images and videos. The key component of CNNs is the convolutional layer, which applies a set of filters to the input, capturing local patterns and spatial dependencies. These learned features are then passed through pooling layers to reduce dimensionality and increase invariance to small variations. CNNs have revolutionized various fields, including computer vision, by achieving state-of-the-art performance in tasks like image classification, object detection, and image segmentation [8].

Sizyaking et al. [5] proposed a CNN as SOTA for crack detection in 2020. The model consist of a first step where morphological features are extracted [3]. This step is added to enable efficient and safe eliminations of areas where it makes little sense to run the learning process. Because of the small amount of data is the CNN model rather small consisting of only two layers, the kernel sizes are 4×4 and 3×3 respectively. At the end of the network a FCNN with 300 neurons is added. The model was trained and compared on different parts on the panel which showed that a model trained on one part of the panel, did not perform too well on another part.

B. Deep Active learning

Active learning (AL) is a machine learning approach that focuses on improving the efficiency of the learning process by involving a human in the training of the model. It aims to reduce the labeling cost of large datasets and more efficiently and accurately train a model. Deep Active Learning (DAL) is AL applied on deep learning models. It is particularly useful when dealing with data that is difficult to annotate, as is the case for the crack detection problem.

The key idea behind DAL is to train a model on a small labeled subset of data initially, and use it to select the most informative samples for further labeling by a human. This iterative process continues until the desired performance is achieved, with the model being retrained on the newly labeled data at each step. The main advantage of this method is that it significantly reduces the human effort required for labeling

the data, as only the most informative samples are selected for labeling [9]. Furthermore, the model's predictions can improve the annotation speed, enabling the human to validate the generated annotation rather than creating it from scratch. In summary, DAL provides a powerful strategy for efficient data labeling and model training, allowing for better performance with less human effort.

When actively learning a deep learning model, there are three things to consider. Firstly, how do we choose the samples which will improve the model's accuracy most, alias which are most informative to the model. These are called the **querying strategy**. Secondly, the **training strategy** decides how we will retrain the model. At last, we need some measure to decide when a model will not improve anymore and is ready to be solely used for predictions and does not need new annotations from the oracle. This is called the **stopping strategy** [10].

III. PROBLEM FORMULATION

In the work of Syziakin [5], the aspects of transfer learning and pretraining the model with newly added annotations were already explored. However, the presented results and discussions were still in a preliminary form, leaving room for further investigation and optimization. The promising results obtained from their approach warrant a more in-depth study and optimization, with the aim of making it applicable to a wider range of digital painting analysis tasks. For continious improvement and integration of these kind of models, we needed to develop the DAL-ART tool with the following requirements:

- The front-end must be user-friendly and have tools to edit and compare images and masks ensuring that users with diverse hardware specifications can seamlessly engage in model training or retraining.
- The platform must enable users to annotate multiple models simultaneously and use existing machine learning models to improve the annotation process.
- The platform must support various programming languages and machine learning frameworks to use different models for different tasks.
- The platform must be web-based to be accessible from anywhere.

IV. PROPOSED METHODS

The DAL-ART tool has been developed to apply to any multi-modal digital painting analysis task. One key element of this is ensuring that the platform can support a wide range of models written in various programming languages. In addition to supporting different programming languages, the platform is also designed to support a wide range of machine learning frameworks and libraries. This includes popular options such as TensorFlow, PyTorch, and Keras, as well as more specialized libraries for tasks like image processing and segmentation.

To review the DAL-ART tool and its features, we start where any user starts: in the login screen. This screen requires

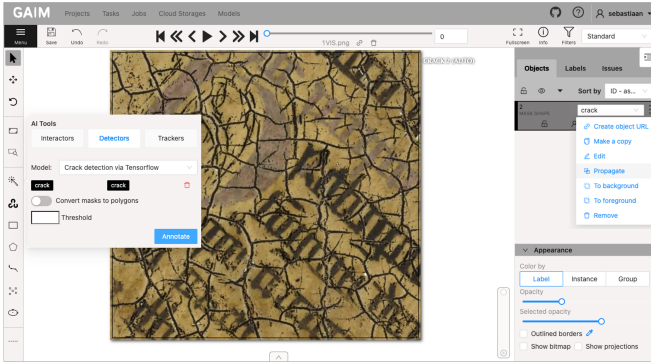


Figure 2: Annotate new data with the previously trained model.

users to provide login credentials to access the data and trained models.

After, a user can create a project to actively train a model. Logically we would have a 1-to-n relationship to the existing models where we start one project per feature of interest that we want to detect. Here the color can be set for the labels to easily differentiate the drawn masks from the original image. On the project, data with different modalities would be uploaded.

Annotating data can be done with the screen shown in Figure 2. To make sure you have the clearest view, you can use the arrows on top of the screen to scroll through the different modalities. Clicking on the pencil tool will prompt a pop-up where you can select the size of the pencil. You can also select the gum tool to correct any errors that have been made by you or the model. For areas that are more difficult to annotate, you can zoom in using the mouse-pad or scroll-wheel. If you want a clearer view of a certain part, you can adjust the image's saturation, contrast, or brightness. This can be done through the panel that pops-up at the bottom of the screen. It is best to annotate areas where the features of interest, such as cracks, are clearest to annotate when creating the mask. For instance, when detecting cracks, the infra-red image may show the straight lines more clearly, while the RGB image provides a cleaner overview of the corners where the cracks intersect. At last, there is an x-ray image, showing variations in height, highlighting the valleys formed by the cracks. If you have initiated the annotation process on the RGB modality, but wish to continue annotating on the X-ray image for example, you can click the propagate button located on the right-hand side of the screen. It is crucial to delete the previous mask when making corrections, as we utilize an OR operator on all annotated masks at prediction time. Failing to delete the old mask could result in the older mask from a different modality overwriting the correction made on the newer mask. In Figure 2 the last and maybe most powerful feature of the annotation screen is visualised. Here, the patch is automatically annotated with a CNN model in the first iteration of its active learning process. To use this feature, you should click on the wand on the left-hand side of the screen. Once clicked, a pop-up will appear allowing you to select a model to use for automatic

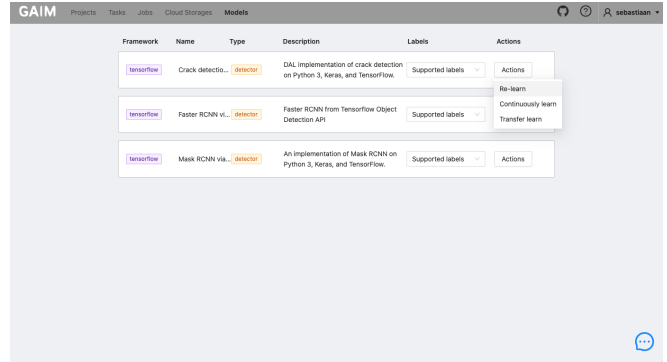


Figure 3: Choose how to actively train a model in the DAL-ART tool.

annotations. By doing this, the annotation task is reduced from full annotations to cleaning of the automatically generated mask.

At last, a training process can be started through the UI in Figure 3. There, you will be able to see a list of available models, along with a description, supported labels and their corresponding framework. At the end of the list, there is an actions drop-down that provides the option to start an active training session for the model. If you select a training option, a modal will appear prompting you to select the version that requires retraining and from which project the training data needs to be taken. In the DAL flow, a user would then go back and annotate more data using the previously trained model. To streamline the development process, we have created an interface that enables AI engineers to seamlessly incorporate new models. By doing so, the DAL model handler ensures the handling of model saving, loading, and versioning. Additionally, it simplifies the process of handling data by abstracting it away.

The tool is deployed on an UGent server and can be accessed via the following link: <https://dal4art.ugent.be/>. By utilizing this web address, users can easily access the tool and take advantage of its functionalities.

V. EXPERIMENTAL RESULTS AND DISCUSSION

When following an active learning flow, our iterative process begins with manual annotation of new data. Subsequently, we actively train a model and repeat this cycle. In our experiments, we will follow the same flow for one of these iterations. Initially, we start from new data along with a previous checkpoint from a crack detection model developed by Sizyakin et al. [5]. Next, we will employ the tool to actively annotate a portion of this data and assess the process and its outcomes. After, we deploy three active learning techniques that have been identified and developed, evaluating their performance in terms of active learning, generalization, and catastrophic forgetting. For the experiments we used data taken from the Book, Joos Vijd and Singing Angels panels from the Ghent Altarpiece.

A. Actively Annotating

Before, we discussed the different challenges faced by current crack detection models [5], [11], such as limited data availability and lack of generalization. To address these challenges, we need a way for augmenting the dataset. We compared the usability, annotation time, and visual quality of annotations using three different methods. For this, we annotated three patches using GIMP, the DAL-ART tool without active learning, and the DAL-ART tool with active learning.

Previously, annotations were made using tools like GIMP, which is a powerful and popular open-source image editing software. Unfortunately, we do not have precise time measurements for the GIMP annotations, but it was estimated to take around 10-20 minutes per patch. The DAL-ART tool without active annotating took slightly longer, averaging around 20 minutes per patch. Compared to GIMP, DAL-ART active annotating showed a 49% improvement in annotation time and a 58% gain compared to manual annotation. All the measurements are shown in Table I.

An explanation for the longer annotations time when using the DAL-ART tool without active annotating could be the quality of the annotations, which are a lot more complete compared to using GIMP. Visually comparing the annotations, the DAL-ART tool with active learning assistance produced the most precise and detailed annotations. It captured finer lines, because the model tended to over predict slightly, allowing to almost only use the gum option when creating the mask.

Usability-wise, both DAL-ART and GIMP offer similar basic editing features, but DAL-ART’s web-based nature allows collaboration and easier setup on different systems.

B. Actively Learning

From literature, we derived three active learning strategies: re-learning, continuously learning, and transfer learning [12], [13]. We tested these strategies using the CNN model created by Sizyakin et al. [5] with the DAL-ART tool. In the re-learning strategy, we retrained the model from scratch using the previously newly annotated patch along with the old dataset. The results showed that adding the new patch through active learning improved the model’s performance on both the new patch, shown in Figure 4b, and a generalization patch. However, the model suffered from catastrophic forgetting, losing some of its previous knowledge, which can be seen in Figure 5b. Therefore, re-learning is not recommended with the current dataset.

In the continuously learning strategy, we fine-tuned the model using the weights of the old model. The results were similar to re-learning, with improved performance on the new patch, shown in Figure 4c, and generalization patches. However, in this case the model did almost not suffer from catastrophic forgetting, retaining most of its previous knowledge, which can be seen in Figure 5c.

For transfer learning, we modified the pre-trained CNN model by only fine-tuning the last 4 layers (the FCNN part)

while keeping the CNN part frozen. The results show in Figure 4d that transfer learning did not significantly improve the model’s performance on any of the new patches. So, we froze too many layers and did not allow enough flexibility for the model to learn new features. Since continuously learning already gives us good very good results without almost any catastrophic forgetting, it is not worth to explore transfer learning further for this model as it would not give us any benefit.

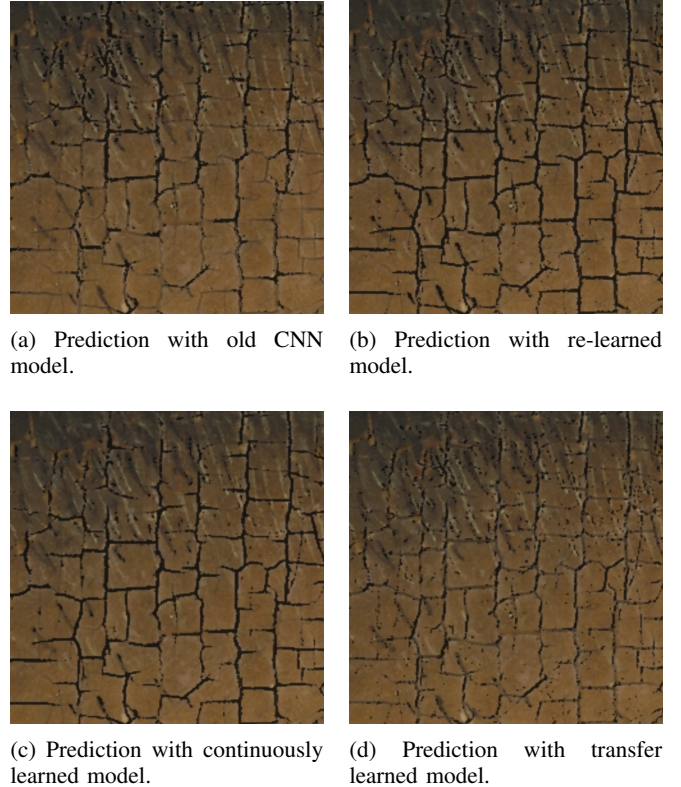


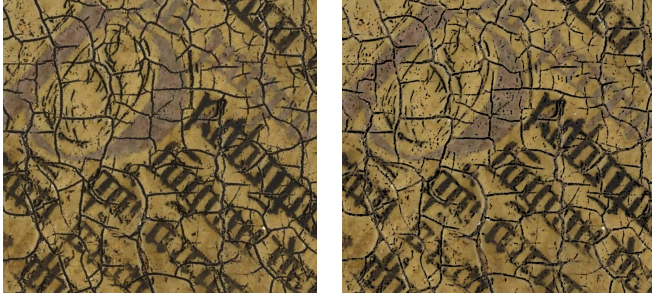
Figure 4: Different predictions after active learning on the JoosVijd panel patch 1438 taken from the Ghent Altarpiece.

C. Discussions

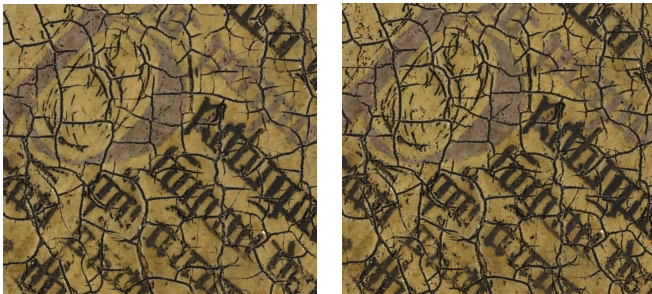
Our experiments show that the DAL-ART tool and the active learning of the CNN model can be very effective. The tool also makes it easy and fast to conduct further experiments, as it offers a smooth user experience. For the case of actively annotating, the DAL-ART tool shows, particularly when combined with active learning assistance, improved annotation efficiency and precision compared to traditional methods like GIMP. Additionally its usability is slightly better because of the DAL-ART’s web-based nature. It is important to note that these findings are specific to the CNN model used in this evaluation. Different models employed for other digital painting analysis tasks or in earlier iterations of active learning process may yield different results, potentially with slightly lower performance gain.

	Annotation time 1	Annotation time 2	Annotation time 3	Average time
GIMP	10-20 min	10-20 min	10-20 min	15min
Manual DAL-ART	16min 20s	18min 31s	20min 15s	18min 22s
Active DAL-ART	6min 41s	8min 55s	7min 9s	7min 35s

Table I: Measured annotation times for three patches of the Ghent Altarpiece.



(a) Prediction with original model. (b) Prediction with re-learned model.



(c) Prediction with continuously learned model. (d) Prediction with transfer learned model.

Figure 5: Different predictions after active learning on an old training patch taken from the Ghent Altarpiece.

When actively learning the CNN model, continuously learning came out on top, as it retained most information and showed good generalization capabilities. It has some drawbacks however where active learning does not seem to resolve the problems with the noisy predictions. Further research is needed to see if a bigger model or newer model like U-Net [11] can reduce the noise in the predictions while actively learning. Another drawback of the model is that the optimal threshold is different after every training process. This makes evaluating the model quite difficult.

At last we have a look at the robustness of the DAL-ART tool and the CNN model. Ensuring the robustness of software research is a critical factor when transitioning from the research phase to a practical implementation, particularly for a UI designed to be actively used by diverse groups with varying backgrounds. Throughout the course of our experiments, we did a thorough assessment of the code’s robustness with the outcome of a good user experience without any identified bugs.

Moreover, we specifically focused on testing the robustness of the CNN implementation within the DAL-ART tool. Our attention turned to addressing a common challenge encoun-

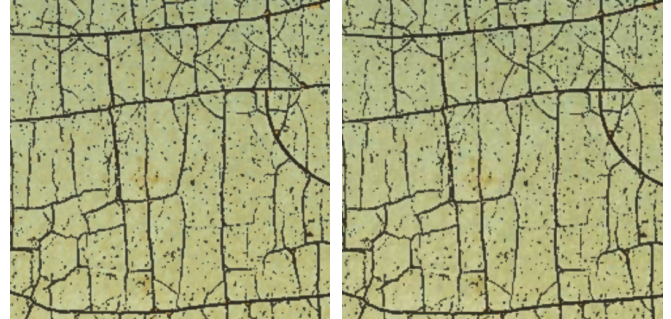


Figure 6: Predictions based on 3 modalities. Figure 7: Predictions based on 1 modality.

tered in real-world scenarios, where certain digital panels may lack specific modalities. To accommodate this, we extended the implementation to allow the model to make accurate predictions solely based on the visual modality.

Figure 6 visually demonstrates the predictions based on the integration of three modalities, while Figure 7 showcases the predictions based on a single modality. Upon observation, the differences between the two are barely visible to the human eye. There is a slight change of only 15 pixels, which translates to a negligible loss of 0.02% in accuracy.

VI. CONCLUSION

In this research paper, we developed the DAL-ART framework, a practical and user-friendly tool for multi-modal digital painting analysis. By seamlessly integrating a deep learning model into an active learning environment, the framework enables users to view and correct model predictions, leading to active learning over time. The tool’s effectiveness has been demonstrated by integrating a state-of-the-art crack detection model and actively training it on patches. We have confirmed the tool’s and CNN’s reliability and effectiveness in generating accurate predictions, even when dealing with incomplete modalities in new panels. This ensures the tool’s practicality and its potential for successful other real-world applications on various feature detection tasks in digital paintings. This work not only advances active learning and crack detection research but also paves the way for improved collaboration between AI developers and end-users in the field of art conservation.

REFERENCES

- [1] B. Cornelis, A. Doms, J. Cornelis, F. Leen, and P. Schelkens, "Digital painting analysis, at the cross section of engineering, mathematics and culture," in *2011 19th European Signal Processing Conference*. IEEE, 2011, pp. 1254–1258.
- [2] The ghent altarpiece by jan van eyck. [Online]. Available: <https://www.worldhistory.org/image/12908/the-ghent-altarpiece-by-jan-van-eyck/>
- [3] B. Cornelis, T. Ružić, E. Gezels, A. Doms, A. Pižurica, L. Platiša, J. Cornelis, M. Martens, M. De Mey, and I. Daubechies, "Crack detection and inpainting for virtual restoration of paintings: The case of the ghent altarpiece," *Signal Processing*, vol. 93, no. 3, pp. 605–619, 2013.
- [4] A. Pizurica, L. Platiša, T. Ruzic, B. Cornelis, A. Doms, M. Martens, M. De Mey, and I. Daubechies, "Virtual restoration and mathematical analysis of pearls in the adoration of the mystic lamb," in *Het Lam Gods Series of Lectures*, 2014.
- [5] R. Sizyakin, B. Cornelis, L. Meeus, H. Dubois, M. Martens, V. Voronin *et al.*, "Crack detection in paintings using convolutional neural networks," *IEEE Access*, vol. 8, pp. 74 535–74 552, 2020.
- [6] T. Ruzic, B. Cornelis, L. Platiša, A. Pizurica, A. Doms, M. Martens, M. De Mey, and I. Daubechies, "Craquelure inpainting in art work," in *Vision and material : interaction between art and science in Jan van Eyck's time, Abstracts*, 2010.
- [7] Sizyakin, Roman, "Deep learning methods for crack detection and image restoration with application to digitized paintings," Ph.D. dissertation, Ghent University, 2022.
- [8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.
- [9] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *Journal of big data*, vol. 2, no. 1, pp. 1–21, 2015.
- [10] Y. Yang, Z. Ma, F. Nie, X. Chang, and A. G. Hauptmann, "Multi-class active learning by uncertainty sampling with diversity maximization," *International Journal of Computer Vision*, vol. 113, pp. 113–127, 2015.
- [11] R. Sizyakin, V. Voronin, and A. Pižurica, "Virtual restoration of paintings based on deep learning," in *Fourteenth International Conference on Machine Vision (ICMV 2021)*, vol. 12084. SPIE, 2022, pp. 422–432.
- [12] C. Shui, F. Zhou, C. Gagné, and B. Wang, "Deep active learning: Unified and principled method for query and training," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 1308–1318.
- [13] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, B. B. Gupta, X. Chen, and X. Wang, "A survey of deep active learning," *ACM computing surveys (CSUR)*, vol. 54, no. 9, pp. 1–40, 2021.

Contents

List of Figures	xvii
List of Tables	xviii
1 Introduction	1
1.1 Objective and Research Methodology	2
1.2 Main Contributions	5
1.3 Thesis Outline	7
2 Motivation and Background	8
2.1 Computer Vision	8
2.1.1 Fully Connected Neural Network	12
2.1.2 Convolutional Neural Network	14
2.1.3 U-Net	16
2.2 Deep Active Learning	17
2.2.1 Querying strategies	18
2.2.2 Training Strategies	19
2.2.3 Stopping Strategies	19
2.3 Conclusion	20
3 Proposed Methods and Implementation	21
3.1 DAL-ART Active Learning	21
3.2 DAL-ART Design	23
3.2.1 Requirements	23
3.2.2 Technologies	24
3.2.2.1 User Interface	25
3.2.2.2 REST Backend	26
3.2.2.3 Model Training API	27
3.2.2.4 Deployment	28
3.2.3 Integration	28
3.3 DAL-ART Characteristics	31
3.3.1 Login Screen	31
3.3.2 Project Screen	31
3.3.3 Task Screen	37
3.3.4 Annotation Screen	37
3.3.5 Model Screen	38

4	Experimental Results and Discussion	44
4.1	Experimental Setup	44
4.2	Actively Annotating	45
4.3	Actively Learning	47
4.3.1	Re-learning	47
4.3.2	Continuously Learning	48
4.3.3	Transfer Learning	50
4.4	Discussions	51
5	Conclusion and Future Work	54
	Bibliography	56
A	Appendix	62

List of Figures

1.1	The Ghent Altarpiece, Hubert and Jan Van Eyck, completed in 1432 [17].	2
2.1	Object detection. Source: [23]	9
2.2	Original digital painting.	9
2.3	Semantic segmentation.	9
2.4	Instance segmentation.	9
2.5	Fully connected network [33].	12
2.6	Convolutional Neural Network [40].	15
2.7	U-Net architecture [42].	16
2.8	A visualisization of how DAL could work in practice [51].	18
3.1	Flow an oracle would follow to actively train a crack detection model.	22
3.2	Architecture of the DAL-ART tool.	25
3.3	Nuclio API to add a model.	29
3.4	Login screen of the DAL-ART tool.	33
3.5	Project screen of the DAL-ART tool.	34
3.6	Screen to start a new project in the DAL-ART tool.	35
3.7	Screen to add new data to the project in the DAL-ART tool.	36
3.8	Screen to have an overview of the tasks.	39
3.9	Screen to annotate new data in the DAL-ART tool.	40
3.10	Annotate new data with the previously trained model.	41
3.11	Choose how to actively train a model in the DAL-ART tool.	42
3.12	Pop-up for propagating a mask to the following page.	43
3.13	Error handling in the case of an invalid configuration.	43
4.1	Comparison of annotated patch 3 of Book 16 taken from the Ghent Altarpiece. .	46
4.2	Different predictions after active learning on the JoosVijd panel patch 1438 taken from the Ghent Altarpiece.	48
4.3	Different predictions after active learning on the JoosVijd panel patch 1518 taken from the Ghent Altarpiece.	49
4.4	Different predictions after active learning on an old training patch taken from the Ghent Altarpiece.	50
4.5	Predictions based on 3 modalities.	52
4.6	Predictions based on 1 modality.	52

List of Tables

4.1	Measured annotation times for three patches of the Ghent Altarpiece.	45
-----	--	----

1

INTRODUCTION

Over the past few decades, the field of computer vision has seen an enormous advancement due to the development of deep learning models. These models have revolutionized the way computers process images and have played a crucial role in various image analysis tasks. Among these tasks, object detection has been one of the most extensively studied areas in computer vision, enabling numerous practical applications such as self-driving cars, medical image analysis, and digital painting analysis [1].

Digital painting analysis is the process of examining and evaluating digitized paintings to automatically discover features of interest. The ability to detect these features can have significant implications for the analysis and understanding of paintings. It can aid in authenticating the origins of a painting and detect potential instances of plagiarism or copying, as well as preserving the integrity of the artwork [2]. One particular challenge here is the accurate and efficient detection of craquelure, which is the web of cracks that forms on the surface of paintings over time resulting in permanent damage to valuable artworks like the Ghent Altarpiece.

Created by the Early Dutch speaking painters Hubert and Jan van Eyck in the mid-15th century, the Ghent Altarpiece - also known as the Adoration of the Mystic Lamb - is an intricate masterpiece of European art. The altarpiece took approximately seven years to complete and can be found in St Bavo's Cathedral in Ghent, Belgium. It is regarded as one of the world's most prized treasures and is particularly noteworthy for being the "first major oil painting" [3]. The Ghent Altarpiece, shown in Figure 1.1, is currently undergoing a major restoration campaign where crack detection has proven to be very useful in assisting art restorers during the restoration and conservation treatments.

The research group GAIM, at Ghent University, has an extensive expertise in the field of digital image processing and in particular in digitized paintings analysis. A series of important results contributing to the preservation of the Ghent Altarpiece has been published during the last decade [4-9], starting from the original article of prof. Aleksandra Pižurica and her student



Figure 1.1: The Ghent Altarpiece, Hubert and Jan Van Eyck, completed in 1432 [17].

Tijana Ružić. [10] as a joint work with Bruno Cornelis [4]. The research group addressed a wide range of problems, from the pearl detection and style analysis [11], image inpainting [12], to paint-loss [13–15] and crack detection [4, 7, 8]. GAIM’s, most important results in the automatic detection of cracks in digitized paintings based on deep learning were obtained by Roman Sizyakin whose publications led to a PhD thesis [16] in 2022 in research group GAIM at Ghent University.

The research group GAIM has a longstanding involvement in the digital analysis of the Ghent Altarpiece, spanning over 15 years. They have collaborated closely with renowned experts, including art historian Prof. Max Martens from Ghent University, art restorers from KIK-IRPA (Hélène Dubois and Bart Devolder), Prof. Ingrid Daubechies from Duke University, KVAB representatives (Em. Prof. Marc De Mey and Prof. M. Martens), and scholars from VUB (B. Cornelis and A. Doods). For this master’s thesis, we have continued our collaboration with them and the tool that we will develop will be made accessible for further research.

1.1 Objective and Research Methodology

The digitization of artistic paintings has sparked an increasing demand for virtual restoration techniques, particularly in the field of crack detection in digital paintings. To address this need, the GAIM research group has developed a series of deep learning models for crack detection, in the PhD thesis of R. Sizyaking [16] and reported in [7, 8]. However, supervised, and in particular deep learning, models typically require substantial amounts of data to achieve good performance. Next to that poses acquiring new labeled data for digital painting analysis a significant challenge,

as it relies on art historians to annotate the data. This process can be both time-consuming and expensive, making it difficult to obtain a sufficient quantity of labeled data for supervised learning. The limited availability of labeled data hampers the training of deep learning models for specific tasks in such domains.

Transfer and active learning techniques help address the challenge of limited labeled data by enabling models to leverage knowledge acquired during pre-training or from a limited set of labeled samples. These techniques offer promising avenues for improving model performance in domains where data scarcity is a concern. In the work of Syziakin [7], the aspects of transfer learning and pretraining the model with newly added annotations were already explored. However, the presented results and discussions were still in a preliminary form, leaving room for further investigation and optimization. The promising results obtained from their approach warrant a more in-depth study and optimization, with the aim of making it applicable to a wider range of digital painting analysis tasks. By delving deeper into the potential of transfer and active learning techniques, it is possible to unlock their full capabilities and enhance the performance of the already existing deep learning models. Further research and optimization efforts can contribute to the development of new practical frameworks that empower art historians and researchers to actively improve models and deploy it on new images.

To address the challenge of limited availability of labeled data, Deep Active Learning (DAL) techniques offer a viable solution. Deep active learning combines active learning with deep learning algorithms, aiming to enhance the learning process's efficiency by involving a user in the training of the model. This approach is particularly advantageous when dealing with challenging data annotation tasks, as it is the case in crack detection in paintings. The fundamental concept behind deep active learning is to train the model initially on a small subset of the available data. Instead of using the entire dataset, the model is trained on a selected subset, and then the most informative samples are chosen for labeling by the user. This iterative process continues, with the model being retrained on the newly labeled data at each step, until the desired performance is achieved [18]. Deep active learning offers several key benefits. Firstly, it enables the model to continuously learn from newly labeled data, mitigating the necessity of annotating a vast amount of data upfront. This adaptive learning approach allows for a more efficient use of resources and accelerates the annotation process. Additionally, the model's predictions can be utilized to improve annotation speed. Instead of annotating from scratch, a user can validate and refine the model's generated annotations, streamlining the annotation workflow.

The second challenge pertains to the continuous improvement and use of the improved model. To address this challenge, the development of a user interface is essential. This interface should prioritize reactivity and responsiveness, ensuring that users with diverse hardware specifications can seamlessly engage in model training or retraining. It is crucial for the tool to possess a modular design, facilitating the easy integration of deep learning models in a plug-and-play

fashion. Moreover, the user interface should support different types of input data and accommodate any number of multi-model input channels. An additional vital feature for the user interface is the capability to fine-tune models specifically on selected areas of paintings. This functionality holds great significance for art historians who wish to apply the already trained models in practical scenarios. By enabling fine-tuning on specific parts of the paintings, the user interface empowers art historians to refine the models according to their expertise and domain-specific requirements. Furthermore, in order to expand the usability of our model, our objective is to develop a specialized web-based user interface tool that caters to the needs of domain experts. Creating a user interface is in itself a challenging task, and developing a web-based user interface introduces additional complexities. The development of a web-based tool requires addressing multiple additional aspects, including intuitive interface design, optimized frontend and backend performance, cross-browser compatibility, scalability, data management, and security measures. All of the aforementioned factors contribute to the challenge of developing an optimized web-based user interface that effectively meets the needs of the user.

Notably, in the classical active deep learning approach, researchers have proposed various sampling strategies such as uncertainty sampling, query-by-committee, and expected model change [19]. These strategies aim to select samples that can provide valuable information to the model during the training process. However, the choice of strategy depends on the specific problem domain and dataset characteristics. In our work, we take a different approach. Rather than relying on predefined sampling strategies, we emphasize the expert user's control in selecting the parts of the crack map to be corrected and adding new annotations. This user-centric approach allows for more flexibility and customization in the annotation process. By giving the expert user the ability to make targeted corrections and add annotations without utilizing sampling strategies beforehand, we empower them to leverage their expertise effectively. It is worth noting that while there are various deep active learning approaches discussed in the literature [19], none of them have been specifically applied or validated for problems similar to the one addressed in our manuscript. Our focus lies on crack detection in very high-resolution and multimodal images, which presents unique challenges and necessitates tailored solutions.

In conclusion, the main objective of this master's thesis is to create a practical DAL framework with an interactive interface. To develop such a tool, we will pursue the following research methodology:

- In order to enhance crack detection research and actively improve model learning, we will conduct thorough research in order to identify the limiting factors in the current State-Of-The-Art (SOTA) approaches. Specifically, deep active learning techniques will be investigated to discover effective ways of actively training a model.
- To facilitate deep active learning, certain modifications to the SOTA crack detection models will be made. These modifications will enable continuous learning, re-learning, and

transfer learning processes.

- Once we have a functional and well-organized deep active learning model at our disposal, we will proceed to design and develop the DAL-ART tool. This will involve carefully selecting the appropriate software packages to allow for the required implementation. The framework should integrate the deep learning models developed by research group GAIM into an active learning environment, enabling users to interact with the model's predictions on selected images and correct any errors by adding new annotations.
- After deploying the models in the DAL-ART tool, we will conduct thorough testing and evaluation to ensure its effectiveness. For this, we will evaluate what the best active learning method was and made sure the model was robust for practical use.

By developing this framework, the results of this thesis will contribute to the advancement of deep learning models in real-world applications. The interactive interface fosters a collaborative environment, allowing AI developers to closely collaborate with end-users, such as art historians, in refining the models and tailoring them to specific tasks.

1.2 Main Contributions

The presented motivation and problem formulation highlight the significant need for an automated web-based user interface for crack detection. To the best of our knowledge, no existing tool has been specifically developed to address these requirements, making our tool the first efficient and user-friendly interface tailored to the needs of field experts. Moreover, it is worth noting that our developed tool is versatile enough to be extended to other digital painting analysis tasks, beyond crack detection.

In this thesis, our main contributions can be summarized as follows:

- To enable deep active learning, we began by utilizing the deep learning models previously developed within the GAIM research group [7] as a foundation. We made crucial additions to these models, including re-learning and continuous learning techniques, while also identifying the most effective approach for transfer learning. By incorporating these enhancements, we ensured that the models can be easily compared with future iterations, promoting efficient and accurate evaluation of their performance.
- Subsequently, we designed and developed the DAL-ART tool, which serves as a practical framework for the continuous development and fine-tuning of deep learning models using various active learning techniques. The interface of the DAL-ART tool was carefully

designed to facilitate efficient annotations. It includes important features such as zooming into the picture, scrolling through multi-modal input, and adjusting transparency settings for annotated masks. These features aim to enhance the annotation process and provide a seamless user experience, ensuring that users can easily correct errors and contribute to the continuous improvement of the model's performance.

- Another contribution of this work is the web-based development of the DAL-ART tool, which is specifically designed to meet the needs of domain experts. Developing a web-based tool adds multiple aspects, including intuitive interface design, optimized frontend and backend performance, cross-browser compatibility, scalability, data management, and implementing robust security measures. Furthermore, it involved ensuring that the DAL-ART tool was made production-ready and successfully deployed. This process entailed thorough testing, debugging, and refining the tool to ensure its stability, reliability, and efficient performance in a real-world environment.
- To offer a comprehensive overview, we integrated the previously mentioned deep learning models into the DAL-ART tool. As part of this integration, we developed a DAL model handler that simplifies the process of adding new models to the deep active learning framework. This model handler significantly reduces the time-to-market for new research, enabling the rapid incorporation of novel models into the tool within a day. This efficient system ensures that the DAL-ART tool remains up-to-date with the latest advancements in crack detection research.
- Finally, we conducted a comprehensive comparative analysis of various deep active learning techniques using the enlisted models. This evaluation allowed us to assess the strengths and weaknesses of each technique, providing valuable insights for further enhancing the performance of these models. Additionally, we demonstrated the robustness of the models when dealing with data that has missing modalities. Moreover, we showcased a significant improvement in annotation speed which will lead to faster active learning by utilizing the DAL-ART tool.

The DAL-ART tool, which has been successfully developed, is now deployed and available for use in a production environment. It can be accessed through the following link: <https://dal4art.ugent.be/>.

1.3 Thesis Outline

The thesis is organized as follows:

- In Chapter 2: *Motivation and Background*, we start with an introduction computer vision and deep learning. After, we have a look at the SOTA deep learning models developed for crack detection. Finally, we examine how active learning can help deep learning and have a look at different DAL techniques.
- In Chapter 3: *Proposed Methods and Implementation*, we will explain the approach we took to design and implement the DAL-ART tool. We then also deploy some SOTA deep learning models and explain how others can be added in the future.
- In Chapter 4: *Experimental Results and Discussion*, we will put the DAL-ART tool to the test and see how different models can be actively learnt. Here we will do a comparison on how well different models can be fine-tuned for practical use.
- In Chapter 5: *Conclusion and Future Work*, we summarize the conducted research and obtained results. To improve our proposed methods, we list several possible directions for future research.

2

MOTIVATION AND BACKGROUND

This chapter will focus on the motives and give an overview of the knowledge required for the remainder of this masters thesis. It also includes an overview of the related work. Firstly, in Section 2.1, we discuss the fundamental concepts of computer vision and its important components. We explore various learning techniques, including supervised, unsupervised, and reinforcement learning. After, we review the most common Deep Learning (DL) architectures used for feature detection in digital paintings and zoom in on semantic classification/segmentation. Additionally, we review the SOTA models used for crack detection. In Section 2.2, we delve into deep active learning and the various techniques employed in this field. Finally, we conclude the chapter in Section 2.3.

2.1 Computer Vision

Artificial Intelligence (AI) is a wide discipline that involves creating computer systems that can perform tasks that would normally require human intelligence, such as recognizing speech, understanding natural language, making decisions based on complex data and computer vision. One area of computer vision that has seen significant progress in recent years is object localization. These algorithms aim to identify and locate a single object within an image, group of images or video [20]. We can accommodate this by drawing bounding boxes around the objects of interest, which we would call object detection. This can be seen in Figure 2.1. We could also use image segmentation where we will divide the image in various parts called segments where each pixel will be assigned to a class or to the background [21]. There are different types of image segmentation, including semantic segmentation and instance segmentation. Semantic segmentation involves assigning a label to each pixel in an image to identify the category of the object it belongs to. This is visualised in Figure 2.3. Instance segmentation on the other

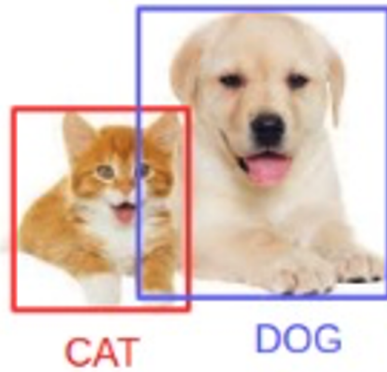


Figure 2.1: Object detection. Source: [23]



Figure 2.2: Original digital painting.

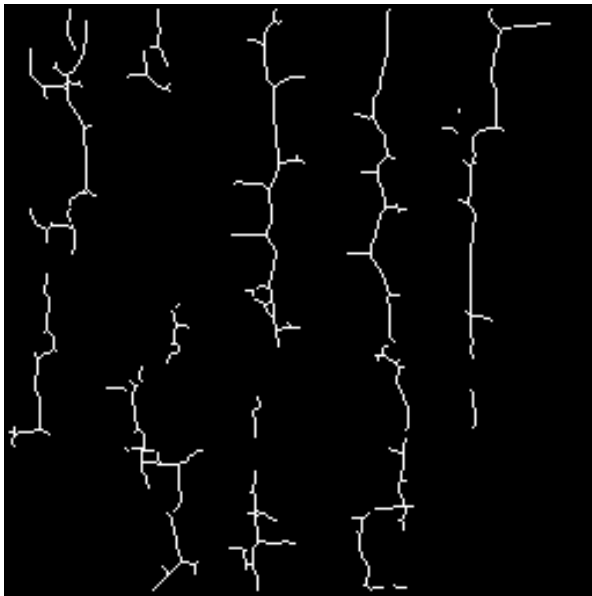


Figure 2.3: Semantic segmentation.

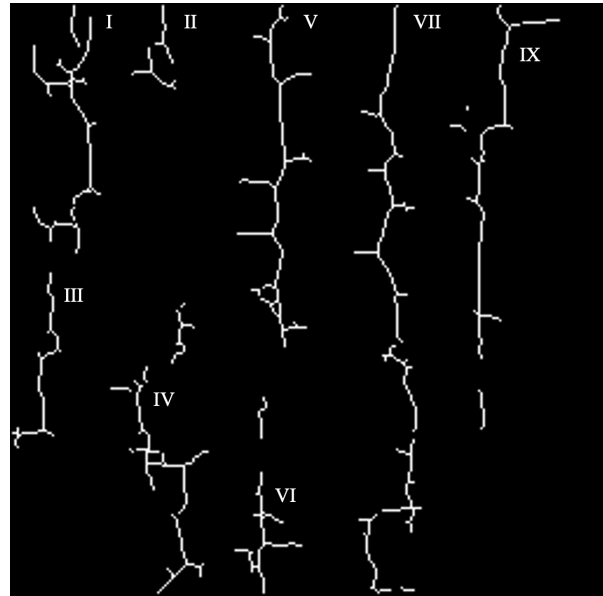


Figure 2.4: Instance segmentation.

hand, entails identifying each individual instance of an object represented by its mask within an image [22]. This is depicted in Figure 2.4 where we would try to identify and separate each crack indicated with Roman numerals.

In this masters thesis we will focus on the group of problems that is considered as computer vision where a mask needs to be generated for an image, thus semantic segmentation, to visualise and test the tool. The tool we will build however, will allow to do any computer vision tasks.

As research develops in this area, numerous mathematical models have been developed to solve these problems. These algorithms we call Machine Learning (ML) algorithms [24]. The idea is to develop statistical models that can learn patterns and relationships from data, and use that knowledge to make predictions about new data. These algorithms can be trained on large datasets and aim to make predictions without being pre-programmed with specific rules or instructions. There are many different types of machine learning algorithms, including supervised learning, unsupervised learning, and reinforcement learning. Each approach has its own strengths and weaknesses, and the choice of algorithm depends on the problem being solved and the available data. Since we are working with crack detection data, we will mostly focus on the subset of supervised ML algorithms in the remainder of this masters thesis. The different subgroups are often intertwined and used together for optimal training. This will be further explained in Section 3.1.

Supervised learning When supervised learning, we will build a model based on a labeled dataset where the outcome variable is known. The algorithm then makes use of this information to forecast the outcome variable for new, unobserved data. [24]. Decision trees, linear regression, logistic regression, and support vector machines are commonly used examples of supervised learning algorithms [25–27]. We can represent supervised learning mathematically as

$$y = f(x), \quad (2.1)$$

where f is the function that we are trying to learn that returns the the output variable y based on an input x . When in the process supervised learning, we are given a set of labeled examples (X, Y) , where X is a matrix of input features and Y is an array of output labels. The goal is to learn the function f that best maps the input features X to the output labels Y . The variable θ denotes the parameters of the function that are learned and are static at inference time in Equation 2.1. We will represent this mathematically as

$$Y = f(\mathbf{X}, \boldsymbol{\theta}), \quad (2.2)$$

with the aim to minimize the error between the predicted output and the actual output. For this, we will use a cost function, which measures the difference between the predicted output and the actual output. The most commonly used ones in machine learning are Mean Squared Error (MSE) and Cross Entropy (CE), also known as negative log-likelihood. MSE is mostly used in regression problems with a model like linear or logistic regression for example. CE on the other hand, is mostly used in multi-class classification problems, with models like neural networks [28]. In Section 2.1.1 we discuss how these functions are used in practice. MSE

and CE are defined respectively as

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2, \quad (2.3)$$

$$CE = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^C y_{i,j} \log(\hat{y}_{i,j}), \quad (2.4)$$

where n is the number of training examples and class and j the number of classes. y_i is the true output, and \hat{y}_i is the predicted output. The objective is then to find the set of parameters θ and function f that minimizes the cost function.

The inputs X and outputs Y can vary significantly depending on the specific problem being addressed. For digital painting analysis and more specifically crack detection, where we do semantic segmentation, the output Y takes the form of a mask, represented by a matrix that matches the size of the input image. The inputs X often contain multiple modalities, with the most common example being an RGB image providing red, green, and blue input channels. When working on the Ghent Altarpiece, we have next to the RGB input, an infrared image that reveals straight lines very clearly. In contrast, the RGB image provides a clearer view of the corners where cracks collide. At last, we also have an x-ray image, showing variations in height and highlighting the valleys formed by the cracks. In total, we will use 5 modalities: red, green, blue, infra-red and x-ray. Using these extra modalities has proven to be very useful during earlier research [29].

Unsupervised learning Unsupervised learning involve models which can learn without explicit labels or supervision. Their objective can be twofold: either to uncover hidden patterns within the data itself or to learn features in the data. The process involves identifying patterns or structure in a set of unlabeled examples, represented as X , with the aim of determining the underlying structure Y that can best explain the observed data. This can be achieved through various techniques, including clustering and dimensionality reduction [24].

Reinforcement learning Reinforcement learning is a type of machine learning algorithm that involves training a model to make decisions in a dynamic environment. The model learns through trial and error by receiving feedback in the form of rewards or penalties, which it utilizes to adjust its behavior and improve its performance over time. The so-called agent interacts with its environment, perceiving and interpreting the surroundings, taking actions, and learning from the consequences of its actions. It uses a combination of exploration and exploitation to optimize its decision-making process. This training method is particularly valuable for tasks like game playing, robotics, and autonomous driving [24]. Recently, reinforcement learning, specifically Reinforcement Learning with Human Feedback (RLHF), was employed to train the SOTA GPT-4 language model, highlighting the ongoing advancements in this field [30].

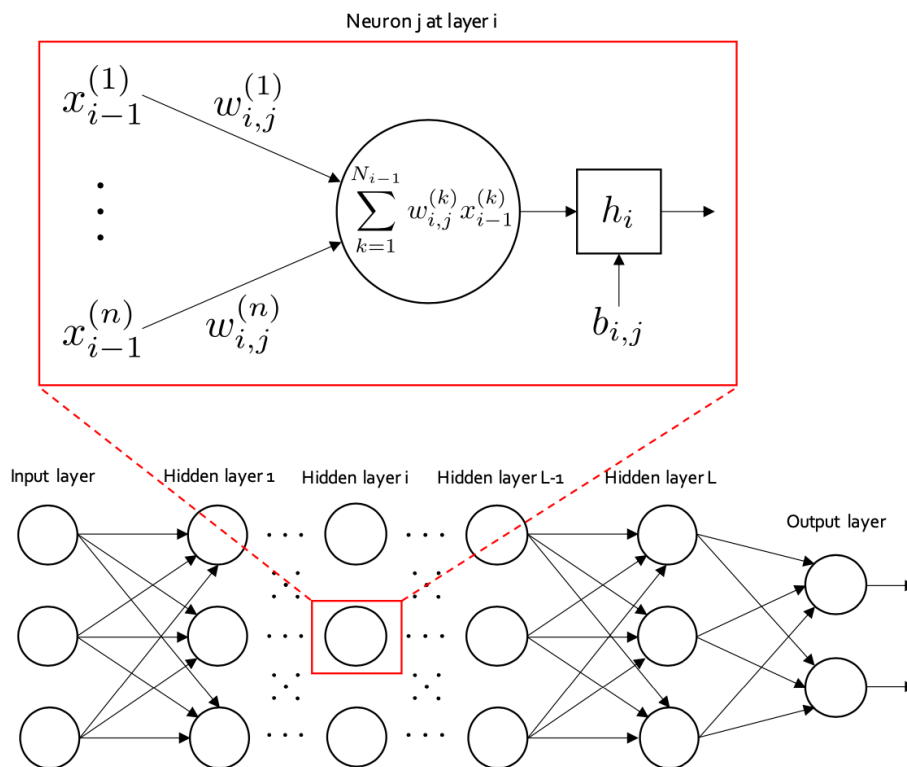


Figure 2.5: Fully connected network [33].

Deep learning is a specialized subset of machine learning that uses neural networks to learn from data. Neural networks are ML algorithms that are modeled after the structure and function of the human brain, and they are able to learn complex patterns and relationships in data by processing large amounts of information through layers of interconnected nodes [31]. It has become particularly popular in recent years because it has been able to achieve SOTA performance in a wide range of tasks. This is mostly due to the fact that deep learning models are able to learn from larger amounts of data than standard machine learning algorithms. In the following paragraphs we will describe and explain different deep learning architectures which are used in the state-of-the-art crack detection research. Firstly we will describe how neurons make up a fully connected network [32]. After, we'll show how Convolutional Neural Networks (CNN) extend these units to achieve better crack detection [7]. We end with describing the current SOTA which is a U-Net architecture composed by Sizyakin et al. [8].

2.1.1 Fully Connected Neural Network

The Fully Connected Neural Network (FCNN), also known as a dense neural network. It is a type of artificial neural network where each neuron in one layer is connected to every neuron in

the previous layer. This creates a fully connected graph between the layers. This is visualised at the bottom of Figure 2.5. Let's say our network is comprised of one single neuron with an input matrix X and output Y , Equation 2.2 can be written as follows

$$Y = f(\mathbf{X}, \boldsymbol{\theta}) = f(\mathbf{X}, \mathbf{W}) = a(\mathbf{W}^T \mathbf{X}) = a\left(\sum_{i=1}^n w_i x_i + b\right). \quad (2.5)$$

Here, \mathbf{W} corresponds to a set of learnable weights $\mathbf{W} = [w_0, w_1, w_2, \dots, w_n]^T \in \mathbb{R}^n$ with a bias b which is often set to 1. The function a is a non-linear activation function which allows, with enough neurons, to construct any relation between X and Y . Without the non-linear activation function the algorithm is the linear regression ML algorithm. The nonlinearity is thus added to be able to learn more complex (nonlinear) correlations between the input matrix and the desired output. Most common examples of activation functions are the sigmoid (binary classification), Rectified Linear Unit (ReLU) and softmax [34]. Their formulas are defined respectively as

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad (2.6)$$

$$\text{ReLU}(z) = \max(0, z), \quad (2.7)$$

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}. \quad (2.8)$$

For Equation 2.8, z is the input to the activation function and n is the number of elements in the input vector for the softmax function.

If we combine multiple of these neurons together we get a FCNN which could in theory approximate any possible relation between input X and output Y [35]. Such an architecture can be seen in Figure 2.5. This network is constructed with an input layer consisting of n units and an output layer consisting of m units. Additionally, there may be multiple hidden layers situated between the input and output layer. The formula in the picture shows that the neurons in each hidden layer i receive inputs from the neurons in the previous hidden layer $i - 1$. The connection between the k^{th} neuron in layer $i - 1$ and the j^{th} neuron in layer i is represented by the weight $w(k)$.

To train such a network when supervised learning, we want to optimise the weights as specified in Equation 2.2. This process is called gradient descent with backpropagation and would go in iterations of 3 steps. Before we can start that, the weights are randomly initialised or in some more recent research set based on the network [36]. Gradient descent can be performed in three ways: stochastic (one sample at a time), batch (entire training set), and mini-batch (small groups of samples). Each has different trade-offs in convergence speed, efficiency, and stability [37]. In the following explanation we will explain the most common approach which is the mini-batch gradient descent, where we call each group of samples a *batch*.

In the first step of the iteration we will pass the batch through the network and use the network to make predictions. This is called the **forward pass**.

After that, we will compute the loss, which is the difference between the predicted output of the neural network and the true output. There are several different types of loss functions to compute this loss, and the choice of which to use depends on the type of problem being solved and the model being used. For example, the loss function CE, specified in Equation 2.4, is used often in neural networks for multi-class classification problems since it penalizes confident but incorrect predictions more heavily than predictions that are uncertain. The second reason this loss function is used often in neural networks is that it is differentiable. This comes in very handy in the next step of the iteration.

In Step 3, the **backward pass**, we calculate the gradient of the loss function with respect to the weights and biases of the network, to then update them in the direction of the negative gradient. This gradient tells us how much each weight and bias contributes to the overall error of the network, and allows us to update the weights and biases to reduce this error. The calculation of the gradient is done using the chain rule of calculus, which allows us to break down the overall gradient into smaller gradients that can be calculated at each layer of the network. Starting from the output layer, we propagate the gradients backward through the network, as the gradient at each layer depends on the gradient of the layer above it [38]. We formulate this mathematically as:

$$W_{k+1} = W_k + \Delta W_k, \quad (2.9)$$

with ΔW_k calculated as

$$\Delta W_k = -\eta \nabla_w E = -\eta \left(\frac{\partial E}{\partial w_{1,k}} \quad \frac{\partial E}{\partial w_{2,k}} \quad \dots \quad \frac{\partial E}{\partial w_{d,k}} \right), \quad (2.10)$$

where, η is the learning rate, $\nabla_w E$ is the gradient of the loss function E with respect to the weights w , and d is the number of weights in the network.

There are several optimization algorithms that can be used instead of, or in combination with gradient descent to improve the efficiency and effectiveness of model training. One such example used in the SOTA crack detection is Adam, which is derived from adaptive moment estimation. Where the classical stochastic gradient descent has a fixed learning rate α , maintains an adaptive learning rate for each model parameter. This adaptive learning rate helps the algorithm converge faster and provides better performance on many types of data and models [39].

2.1.2 Convolutional Neural Network

Some pragmatic implementations have shown that FCNNs can perform quite well on confined problems like digit classification in images [41] where overfitting is allowed. FCNNs lose a lot of spatial information in images however by flattening the data into a 1d vector to be fed to the input layer. This becomes even more of a problem when we use multi-modal images where the pixels of the first image are highly correlated to the same spatial location pixels of the second image.

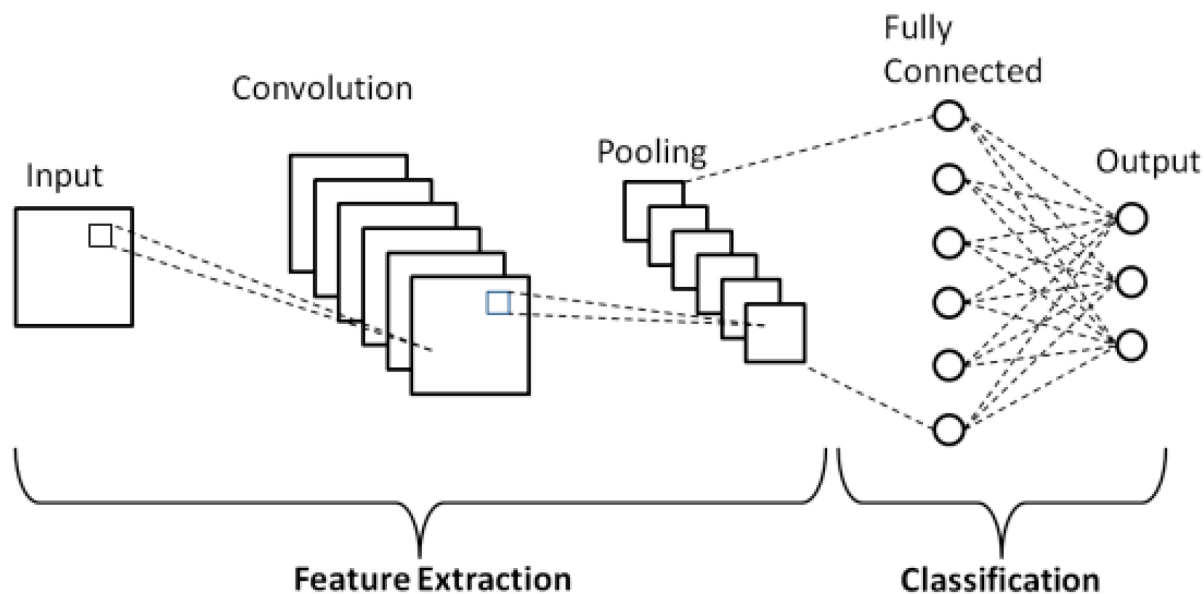


Figure 2.6: Convolutional Neural Network [40].

When flattened into one long vector, a lot of this information would get lost. To better exploit structured data such as images, we must extend the basic fully connected neural network. This is where convolutional neural networks (CNNs) come into play.

The two main components of a CNN, visualised in Figure 2.6, are the convolution layer to extract spatial features, and a pooling layer often used to reduce the dimensionality. Convolution layers apply filters to their input data to extract local patterns while preserving spatial information. These filters are $n \times n$ dimensional matrices called *kernels*. The second hyperparameter of a CNN is the stride s . It determines the spacing between adjacent kernels of the filter. A stride of 1 means that the filter moves one pixel at a time, while a stride of 2 means that the filter moves two pixels at a time, and so on. A larger stride can be used to reduce the computational complexity and dimensionality, but can also be useful for detecting larger features in the image without having a really big kernel. In addition to the convolutional layers, a CNN also typically includes pooling layers, which reduce the dimensionality of the output feature maps by subsampling them. The most common type of pooling layer is the max pooling layer, which takes the maximum value within a pooling region. The pooling operation can also have a stride parameter s , which determines the spacing between adjacent pooling regions. At the end of a CNN we will often add a small FCNN to convert the extracted features to the right output. As denoted in Figure 2.6, the CNN would be considered the feature extraction part and the FCNN the classification part.

Sizyaking et al. [7] proposed a CNN as SOTA for crack detection in 2020. The model consists of a first step where morphological features are extracted [4]. This step is added to enable efficient

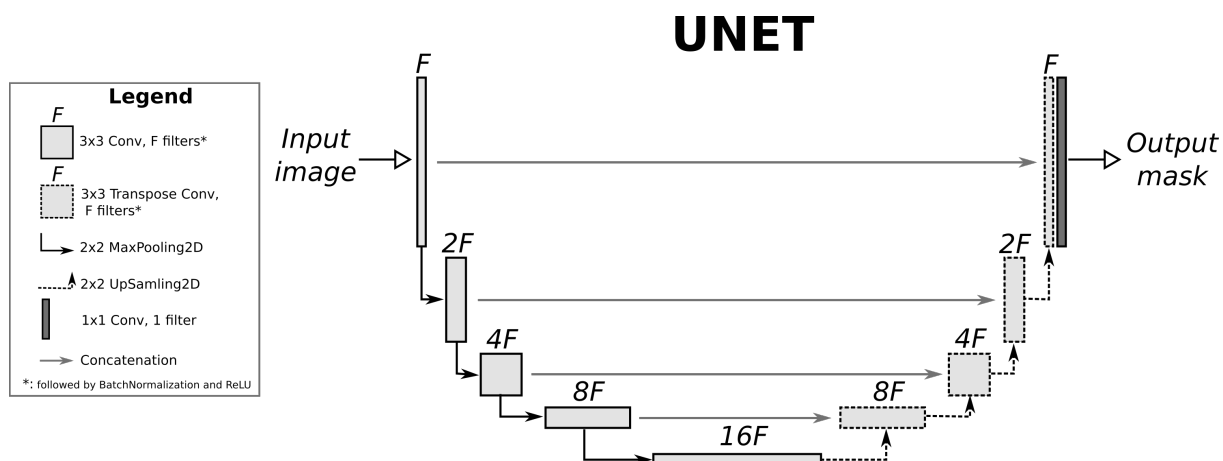


Figure 2.7: U-Net architecture [42].

and safe eliminations of areas where it makes little sense to run the learning process. Because of the small amount of data is the CNN model rather small consisting of only two layers, the kernel sizes are 4×4 and 3×3 respectively. At the end of the network a FCNN with 300 neurons is added. For the loss function they used (Binary)CE loss (Equation 2.4). The activation function is ReLU and for the optimizer they used Adam.

The model was trained and compared on different parts on the panel which showed that a model trained on one part of the panel, did not perform too well on another part. This we will try to solve with Deep Active Learning in Sections 2.2 and 3.1.

2.1.3 U-Net

As a consequence of the recent advancements and improvements in computer vision using CNNs, the research in architecture of such networks has increased a lot. One of the first big CNN networks is called AlexNet and it competed in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012. The network achieved a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the runner up [43]. By utilizing ReLU's and implementing dropout on the fully-connected layers, Sutskever et al. [44] showcased an improvement in training efficiency while also mitigating over-fitting. More recently even bigger networks like VGG16 and VGG19 have been proposed by Simonyan et al. These days they are used in any field where deep learning is applied ranging from medical applications to malware detection [45, 46].

More recent advances are the development of U-Nets proposed by Ronneberger et al. [47]. These U-shaped CNN networks have since then become a popular choice for any image segmentation problem [48, 49]. The architecture's distinctive shape, visualised in Figure 2.7, consists of a contracting path and an expanding path, with a bottleneck layer in between. The contracting path is a traditional CNN architecture consisting of convolutional and pooling layers, which

gradually reduce the spatial size of the input while increasing the number of feature channels. The expanding path consists of upsampling and convolutional layers, which gradually increase the spatial size of the output while reducing the number of feature channels. One of the main advantages of UNETs is their ability to segment images with high accuracy, even when the images have complex and irregular shapes. This is due to the architecture's ability to capture both local and global features of the input.

Sizyaking et al. used this kind of architecture to improve upon their earlier mentioned model [8]. For their network all convolutional layers have a spatial filter size of 3×3 pixels. When training they used the common Adam optimizer with a learning rate of 0.00002. To improve the model's practical utility, they have implemented a smaller $20 \times 20 \times 5$ pixel input tensor with reduced spatial size. Their research has revealed certain advantages of the architecture, including precise crack localization and fast network learning [29]. However, it is crucial to ensure complete and accurate annotations of training data when such a model is used since inadequate or imprecise annotations will result in poor model performance or convergence issues.

2.2 Deep Active Learning

The SOTA in crack detection has come up with a lot of powerful and accurate deep learning models to detect cracks in digital paintings as shown in Sections 2.1.2 and 2.1.3 [7, 8]. Most mention however, that one of the main limitations of their models is the amount of data available to train it and its robustness against data drift. This semantic segmentation problem requires a lot of manually annotated data which is preferably created by experts in the field and annotated on multi-modal data. All this makes data very expensive. This problem will be solved in this masters thesis by using deep active learning (DAL).

Active Learning (AL) is a machine learning approach that focuses on improving the efficiency of the learning process by involving a human in the training of the model. It aims to reduce the labeling cost of large datasets and more efficiently and accurately train a model. DAL is AL applied on deep learning models. It is particularly useful when dealing with data that is difficult to annotate, as is the case for the crack detection problem.

The key idea behind DAL is to train a model on a small labeled subset of data initially, and use it to select the most informative samples for further labeling by a human. This iterative process, visualised in Figure 2.8, continues until the desired performance is achieved, with the model being retrained on the newly labeled data at each step. The main advantage of this method is that it significantly reduces the human effort required for labeling the data, as only the most informative samples are selected for labeling [50]. Furthermore, the model's predictions can improve the annotation speed, enabling the human to validate the generated annotation rather

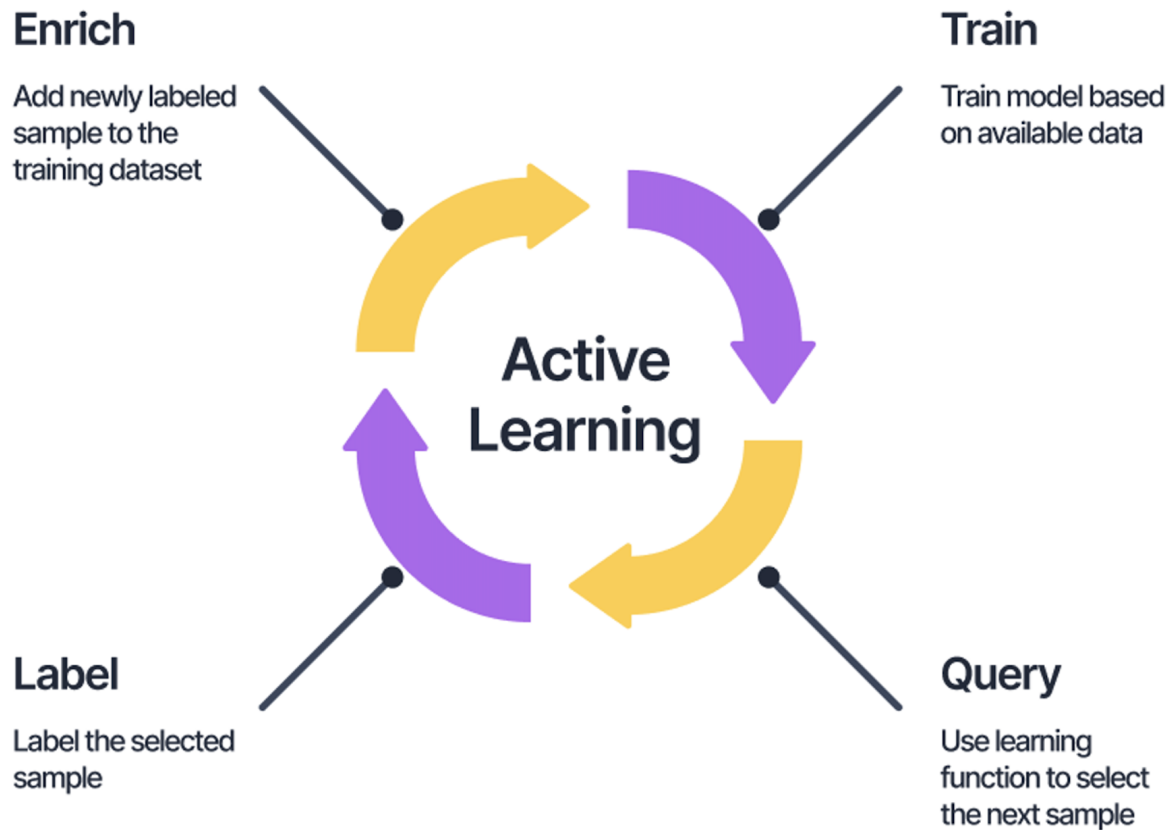


Figure 2.8: A visualization of how DAL could work in practice [51].

than creating it from scratch. In summary, DAL provides a powerful strategy for efficient data labeling and model training, allowing for better performance with less human effort.

When actively learning a deep learning model, there are three things to consider. Firstly, how do we choose the samples which will improve the model’s accuracy most, alias which are most informative to the model. These are called the **querying strategy**. Secondly, the **training strategy** decides how we will retrain the model. At last, we need some measure to decide when a model will not improve anymore and is ready to be solely used for predictions and does not need new annotations from the oracle. This is called the **stopping strategy**.

2.2.1 Querying strategies

Querying strategies in active learning determine how to select the most informative samples from the unlabeled data pool. Deep active learning has several querying strategies that are commonly used to select the next batch of samples for labeling. We identified the following strategies as most interesting:

- **Uncertainty Sampling** This strategy selects the samples for which the model is the most uncertain in its prediction. The model may have low confidence or high entropy about a particular sample. The idea is that labeling these samples will lead to the greatest reduction in uncertainty and therefore the greatest improvement in model performance [52].
- **Diversity Sampling** This strategy selects the samples that are the most dissimilar from the labeled data. The idea is to cover a wide range of feature space to ensure that the model is exposed to a diverse set of examples [53].
- **Query by Committee** This strategy involves training multiple models on the same dataset and selecting the samples that have the most disagreement among the models. The idea is that these samples are more difficult to classify and therefore require additional labeled data to improve model performance [54].

2.2.2 Training Strategies

While the focus of most research lies on the querying strategy, it is crucial not to overlook the training strategy of the model. Based on the literature [18, 55, 56], we have identified three effective training strategies for actively retraining a model. The first strategy involves relearning the model from scratch. In this approach, we incorporate new data into the existing dataset and train a completely new model. The second strategy is known as continuous learning. Here, we take a previous checkpoint of the model and fine-tune it using the new data. At last we have transfer learning. This method is commonly employed in training larger models like Unets and is prevalent in both the deep learning field as the DAL domain to ensure more efficient learning and mitigate catastrophic forgetting. Typically, the feature extractor is frozen or fine-tuned with a smaller starting learning rate, while the classifier part, which may consist of multiple layers, is retrained.

2.2.3 Stopping Strategies

In deep active learning, the stopping strategy refers to the conditions under which we should stop querying training data from the oracle. These strategies are critical because they ensure that the model does not over-fit on the training data and that it generalizes effectively to previously unknown data without terminating the training process prematurely. In literature a fixed stopping criteria for training the model is often defined as the stopping strategy. This can for example be a threshold on the accuracy or loss, together with the maximum number of iterations or minimum number of labeled samples are examples [57, 58]. More advanced

stopping criteria used in deep learning like a minimum improvement of accuracy or a certain validation loss plateau are also transferred to the DAL field [59, 60].

Bloodgood et al. [61] state however that these stopping criteria might go against the idea of AL since stopping too early based on fixed criteria might lead to a big loss in model performance. Next to that defeats annotating a separate set of data for a validation loss plateau the purpose of AL. Instead, the authors propose using the predictions of consecutively learned models on unlabeled examples to identify when to stop the AL. Specifically, the authors suggest monitoring stabilized predictions on a representative set of examples known as the *stop set*, which should be available for querying by the active learner. The authors acknowledge the conflicting factors involved in determining the optimal size of the stop set. They would prefer a small set to speed up the process of active learning and limit the amount of data gathering that needs to be done. On the other hand would a bigger stop set be preferred to accurately measure the agreement between successive models predictions while avoiding statistical variability.

2.3 Conclusion

This chapter has provided an in-depth introduction to the essential background knowledge required for this master's thesis. We started with an overview of the field of computer vision, where we discussed its key components. We then focused on the prevailing DL architectures utilized for feature detection in digital paintings, which included FCNNs, CNNs and U-Nets, as explained in Section 2.1.

Furthermore, in Section 2.2, we delved into the domain of deep active learning, where we explained the various techniques employed with the most common querying strategies, training strategies and stopping strategies. This background knowledge will serve as the foundation for the following chapter, where we will explore the application of DAL for crack detection in digital paintings and design and develop the DAL-ART tool.

3

PROPOSED METHODS AND IMPLEMENTATION

In this chapter, our main focus will be on the creation of the DAL-ART tool. We will explore the various steps involved in its development, starting with an overview of how deep active learning is utilized in our particular use-case. Next, we will delve into the design decisions that were made to elevate this tool to a higher level of functionality. This will include an examination of the different factors that were considered during the design process, as well as the challenges that were faced and overcome. Once we have established the design framework, we will take a closer look at the features that have resulted from it, and how these can be utilized to actively train a DL model. We will explore the different ways in which the tool can be used, as well as the benefits that it offers for practitioners working in the field of digital painting analysis.

3.1 DAL-ART Active Learning

Deep active learning can take on many forms depending on which problem is being solved with which kind of model. The different strategies that can be used are described in Section 2.2. For the general DAL-ART tool we designed we developed a custom flow that would work to develop any model to detect any features of interest in digital paintings. The generalisation of this tool is a very important aspect for it to be applicable to as many problems as possible.

The flow an oracle would take to actively train a ML algorithm can be seen in Figure 3.1. In this flow, an oracle is tasked with training a model using a set of data. The first step is to upload all of the data to a server, which will serve as the training environment. That data is then used to train a new model from scratch. After the first training session the stopping criteria are most likely not reached. So, the oracle can then choose which new data to annotate. To assist with this process, the already trained ML model can be used to pre-compute a mask, which can help the oracle identify which data points may be most useful to annotate. Once the oracle has selected the data points to annotate, they can begin the process of annotating the

data. This involves labeling the data points with their respective classes or categories. After the annotation process is complete, the oracle can then press a button to actively train the machine learning model. If the model's stopping criteria have been reached, then no new data should be selected for annotation. If the criteria have not been met, then the oracle will be redirected to the beginning of the process, where they can choose new data to annotate. Here, it will again be helpful to look at the data where the model made most mistakes.

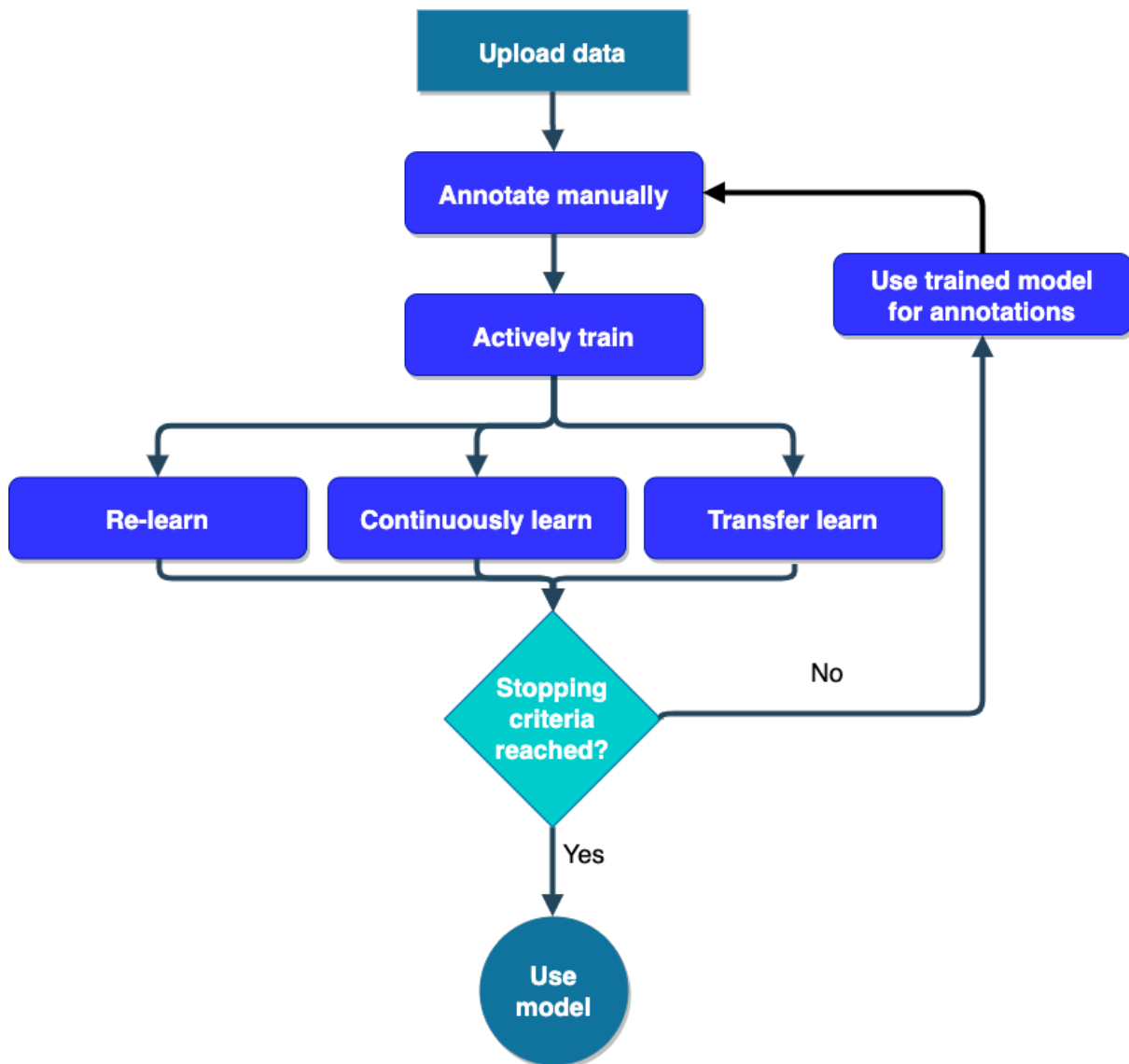


Figure 3.1: Flow an oracle would follow to actively train a crack detection model.

3.2 DAL-ART Design

In this section, we will have a look at the infrastructure of our DAL-art tool and explore the ways in which we implemented our ideas in a pragmatic and user-friendly way. To achieve this, we leveraged a combination of cutting-edge technologies that enabled us to build a powerful and intuitive tool. We start by identifying the requirements of DAL-ART to be able to choose the right tools for the job. After, we give an introduction to Django, Computer Vision Annotation Tool (CVAT), Docker and Nuclio, which are the technologies used in our tool. Next, we look at the design of our tool and how these technologies are integrated to create a great user experience.

3.2.1 Requirements

The process of designing software involves a lot of decisions, from the architecture to the User Interface (UI). Among these decisions, the choice of which technologies to use is undoubtedly one of the most crucial [62]. The success of a software project can largely depend on whether the chosen technologies are appropriate for the project's goals and requirements. Choosing the right technology stack will ultimately contribute to the success of the software project and ensure its longevity in the dynamic landscape of software development.

When considering which technologies to incorporate into a software project, it's essential to have a clear understanding of the project's objectives, the needs of its users and the capacity of the team. For instance, if the software needs to be fast and responsive, the technology stack should be designed to optimize performance. If software stack is too difficult to integrate, and the project is not finished, the DAL-ART tool will most likely never be used.

First and foremost, it is crucial to have a front-end that is user-friendly and intuitive. This means that the interface must be easy to navigate and understand, without requiring extensive training or a lengthy instruction manual. Art historians, who are often busy professionals, may not have the time or patience to learn complex software.

In addition to being user-friendly, the UI should also include a range of tools and features that facilitate the annotation process. These include tools to, change the brush sizes or correct when a mistake is made in the mask, zoom in and out of parts of the image, change the mask and image's properties like opacity, saturation, contrast and brightness, but also give the ability to change the mask's opacity to see on the overlay if an annotation is done correctly.

Another very important feature is the ability to choose the modality on which the crack - or any other feature of interest - is clearest to annotate when creating a mask. For the case of crack detection for example, the infra-red image shows the straight lines more clearly, where the RGB image gives a cleaner overview on the corners where cracks collide.

Next to that, would it be very beneficial to allow users to work on multiple models simultaneously.

This can be achieved by dividing the model's data into separate projects, each with its own set of users and resources. It is also important to maintain links between the different modalities to ensure consistency and avoid duplicate work. To accomplish this, the platform could create separate tasks for each user to complete, assigning them specific sections of each model to annotate.

To further enhance the usability of the annotation platform, it will also be helpful to use the already trained ML algorithms to improve the annotation process. For example, after the first iteration of the AL process or a model from a previous project can be used to annotate. This will save time during the AL process and will reduce the risk of errors or inconsistencies in the annotation process [63].

In order to create an effective and versatile DAL-ART annotation platform, it is important to prioritize flexibility and adaptability in the development process. One key element of this is ensuring that the platform can support a wide range of models written in various programming languages. In addition to supporting different programming languages, the platform should also be designed to support a wide range of machine learning frameworks and libraries. This could include popular options such as TensorFlow, PyTorch, and Keras, as well as more specialized libraries for tasks like image processing and segmentation.

At last, we have chosen to develop a fully web-based tool for a number of reasons:

- **Accessibility and maintenance:** A web-based tool can be accessed from any device with a web browser and does not suffer from operating system incompatibilities. This makes a web-based tool more accessible and convenient for users who need to access the tool from different locations or devices.
- **User experience:** Web-based tools can offer a more modern and interactive user experience, with features such as real-time data updates, online collaboration, and responsive design.
- **Scalability:** A web-based tool can be easily scaled up to support a large number of users, or when having to train a deep learning model, the computing power can be scaled more easily. This is novel compared to currently existing DAL strategies [64, 65].

3.2.2 Technologies

Once the requirements have been taken into account, the technology stack can be determined. This stack includes programming languages, frameworks, libraries, databases, and other tools. Each component of the stack is of course evaluated for its compatibility with the others, as well as its performance and maintainability.

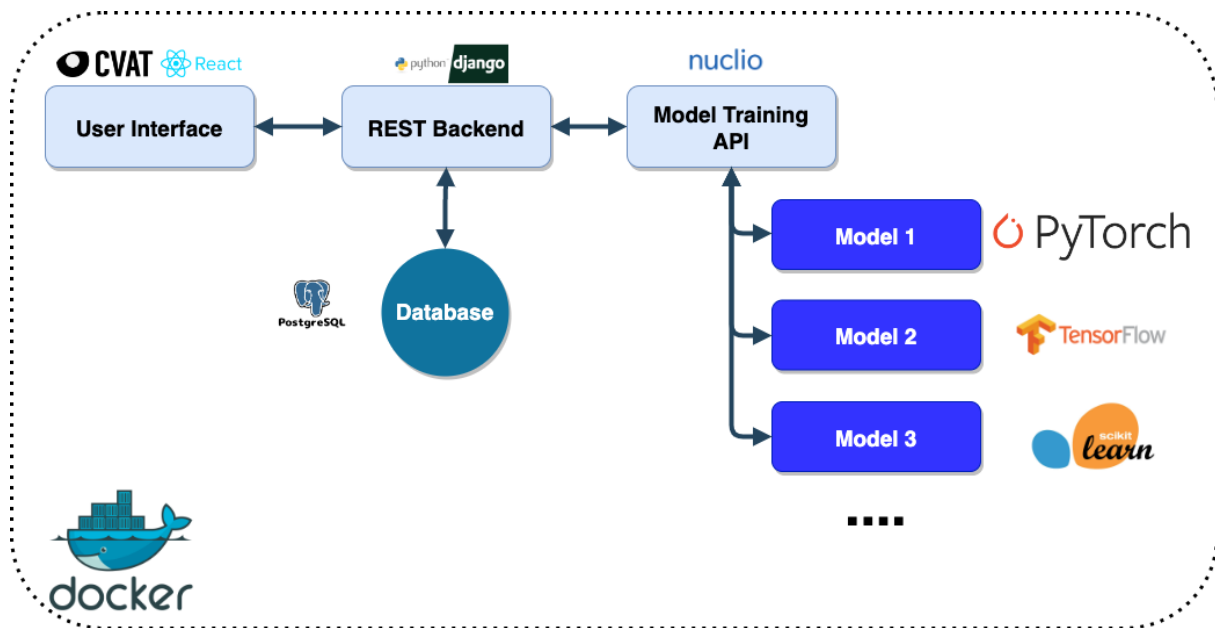


Figure 3.2: Architecture of the DAL-ART tool.

To establish our DAL-ART tool and meet all set requirements, we need three logical components. A UI which allows to annotate masks for the uploaded image. A Representational State Transfer (REST) back-end which allows to save and load the images, masks, and models. And a part which can actively train the model with an Application Programming Interface (API) call. In Figure 3.3, the architecture of the DAL-ART tool is shown with the these 3 different segments shown in light blue boxes. How they work together to create the DAL-ART tool will be further explained in Section 3.2.3 .

3.2.2.1 User Interface

The UI has a lot of requirements it needs to satisfy. With the time constraints of a thesis we need to look for a combination of softwares that will work well together and provide some of the needed features out-of-the-box. However, the need for a good UI can not be overlooked during the development process and thus was one of the main priorities.

For the UI of the tool, we utilize React [66]. React is a widely used JavaScript, and TypeScript, library for building UIs. It offers a range of tools and features that make it easy to build highly interactive, responsive and visually appealing UIs. React’s virtual DOM and component-based architecture make it a very popular choice for building complex web applications which make it used by companies like Facebook and Netflix.

While there are other frameworks and libraries that could have been used to build the UI of our tool, we felt that React was the best fit for our specific needs. Other popular options, such as

Angular [67] and Vue [68], also have their strengths and weaknesses, but we ultimately decided that React offered the right combination of flexibility and community support.

There are quite some software packages available already that provide some of the requirements we have for the annotation process of our DAL-ART tool. Most are not licensed under an open source license or do not provide mask annotations however and become thus unavailable for us. The three software packages that made it to the shortlist are LabelMe [69] and Common Objects in Context (COCO) Annotator [70] and CVAT [71]. LabelMe is a web-based image annotation tool that is widely used in research, but it does not offer the same level of customization and flexibility as CVAT. COCO Annotator, on the other hand, is a lightweight image annotation tool that is specifically designed for annotating images in the COCO format, so it will not be suitable for projects with different annotation requirements.

So, for most of the low level annotation functionality of the tool, we leverage CVAT. CVAT is an open-source platform that is specifically designed for image and video annotation [71]. The tool already provides the majority of the UI requirements outlined in Section 3.2.1 out-of-the-box without requiring extensive customization. A notable drawback however is the project's size and complexity, which may pose challenges when attempting to extend its functionality.

By using CVAT for the annotation functionality and React for the UI, we are able to take advantage of the strengths of both technologies. CVAT provides a powerful and customisable annotation platform, while React provides a flexible and responsive UI. Together, these technologies enable us to build a robust and user-friendly tool that can be customised to meet the specific needs of DAL-ART.

3.2.2.2 REST Backend

We opted for a web-based application, so we need a REST backend that can provide data to our UI. One of the key benefits of using a REST backend is its flexibility. RESTful APIs are designed to be language-agnostic, which means they can be accessed by UI written in any programming language. This makes it easy to build applications that can communicate with the backend, regardless of the technology stack used on the front end. If later during the use of the DAL-ART tool, one would want to change the UI, the interchange should be fairly easy.

When deciding on a technology for the backend of our tool, we evaluated several options, including Flask [72], and Django [73]. All of the options were python frameworks however, because python is currently the most used language for writing backends [74]. This will allow an easy continuation and ensure the longevity of the DAL-ART tool. Ultimately, we choose Django because of its powerful features, ease of use, and strong community support.

Django is a Python-based web framework that provides a wide range of features for build-

ing web applications, including a powerful Object-Relational Mapping (ORM) for interacting with databases, built-in admin interface, and support for user authentication and authorization. These features make it easy to build complex web applications quickly and efficiently.

In addition to its built-in features, Django has a large and active community of developers who have contributed a vast ecosystem of third-party packages and libraries. This means that there are many resources available for developers who use Django, such as tutorials, and support forums.

Another factor that influenced our decision to use Django is its ease of use. Django's documentation is very comprehensive and easy to follow, which makes it easy to get up to speed quickly. Django's built-in admin interface also makes it very easy to manage the application's data and content without writing any custom code for this.

The most commonly used databases used with the Django framework are SQLite [75], and PostgreSQL [76]. While both PostgreSQL and SQLite are popular open-source relational database management systems, they have different strengths and weaknesses. SQLite is a lightweight, and self-contained database that can be easily embedded into an application. It is also the default database used in the Django documentation. PostgreSQL on the other hand, is a more advanced and powerful database that provides support for complex queries, indexing, and transactions. It is designed to handle large amounts of data and can be more easily scaled by adding more servers or instances than SQLite. Therefore, in the end, we decided to go with PostgreSQL.

3.2.2.3 Model Training API

The last and most important part of the DAL-ART tool that is missing, is a way to train and deploy machine learning models. As specified in Section 3.2.1, there are some very hard requirements for this part of the tool. Namely, we want to be able to use any programming language and any ML framework. This is where the Nuclio framework comes in [77].

Nuclio is a serverless framework for deploying and running functions or microservices wherever you want. It provides a simple and scalable way to run code without having to deal with the underlying infrastructure, and it can be easily integrated with other services and platforms. Since a Docker [78] container is created per deployed model, we can use any programming language and any framework. Deploying a new model and actively training it, is as simple as creating only two files and pressing a button, a feature that we have further explained in Section 3.2.3. This way we enable AI developers to focus on writing code rather than worrying about the infrastructure and how to integrate their models into the system.

Nuclio is originally not designed to load and store models. It would preferably load it's models from the cloud. We chose to store the models on disk instead of in a database to save time. While this may not follow best practices, it can be a practical solution for smaller applications

and helped to be able to finish the tool in due time. Next to that, allows storing the models on disk us to quickly and easily retrieve them when needed, without having to worry about the overhead of managing an extra database.

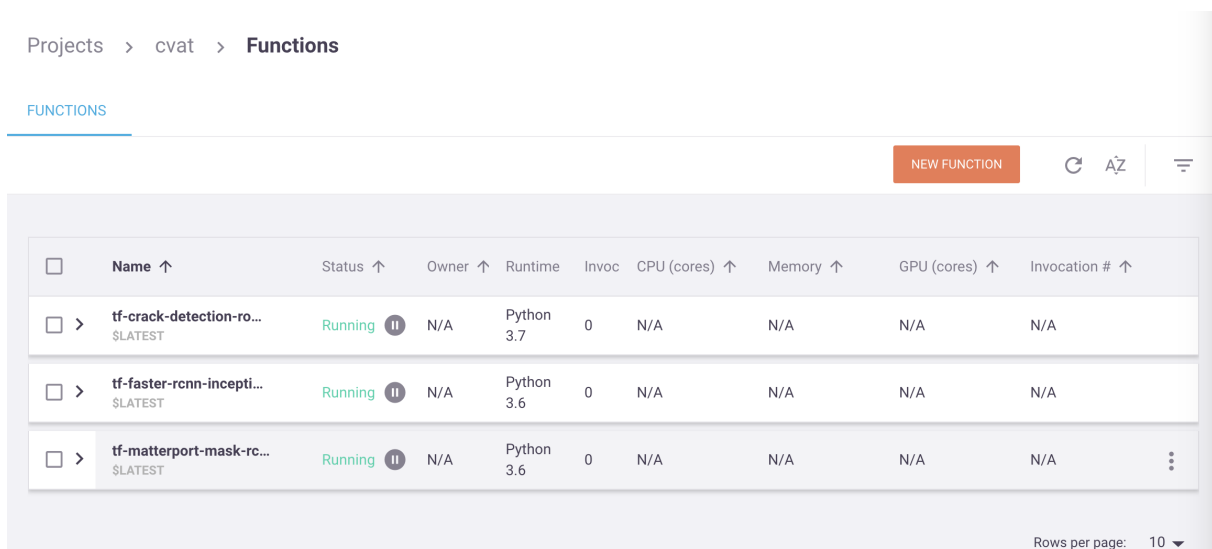
3.2.2.4 Deployment

For the deployment we use Docker [78] containers because they provide a highly portable and efficient solution for running applications across different platforms and environments. Docker allows us to package all the dependencies and configurations needed to run the application into a group of containers, making it easier to deploy and scale the tool. With Docker, we can ensure consistency and reproducibility across different environments, which allows us to deploy the DAL-ART tool wherever we want. Additionally, Docker provides a lightweight and isolated environment that minimizes the risk of conflicts and interference with other applications running on the same machine. Using this, the tool is deployed on an UGent server and can be accessed via the following link: <https://dal4art.ugent.be/>. By utilizing this web address, users can easily access the tool and take advantage of its functionalities.

3.2.3 Integration

The used software stack lays a robust foundation for the DAL-ART tool. As shown by the links in Figure 3.3, we have four integrations that need to work seamlessly together to guarantee a practically usable DAL-ART tool with a smooth user experience. The connection between the REST backend and the database is facilitated by the Django ORM system. This ensures a reliable and robust link between the backend and the database, handling data storage and retrieval. Secondly, is the integration between the models and the Nuclio API handled by Nuclio itself, eliminating the need for further explanation in these two integrations. Our focus however, is on the integration between the UI, backend, and model training API. To allow deep active learning of models, we must establish communication channels between the UI and the model training API through the Django backend. These are all API's communicating with each other and will be clarified with the flow of training a model, obtaining predictions from a model and adding a model.

Model Training The training of a model will be started on the UI in the Models screen (see Section 3.3.5). This will make a synchronous call to the backend which starts preparing the dataset. The dataset is then exported to the disk in a folder which is mounted to the model's Docker container. This way the data can easily be loaded by the DAL model handler. If any errors occur in these steps, they will be displayed on the screen for the user to see, as shown



The screenshot shows the Nuclio API interface for managing functions. The breadcrumb navigation is 'Projects > cvat > Functions'. Below the navigation, there is a 'FUNCTIONS' header and a 'NEW FUNCTION' button. The main content is a table with the following columns: Name, Status, Owner, Runtime, Invoc, CPU (cores), Memory, GPU (cores), and Invocation #. Three functions are listed, all in a 'Running' state.

<input type="checkbox"/>	Name ↑	Status ↑	Owner ↑	Runtime	Invoc	CPU (cores) ↑	Memory ↑	GPU (cores) ↑	Invocation # ↑
<input type="checkbox"/>	tf-crack-detection-ro... SLATEST	Running	N/A	Python 3.7	0	N/A	N/A	N/A	N/A
<input type="checkbox"/>	tf-faster-rcnn-incepti... SLATEST	Running	N/A	Python 3.6	0	N/A	N/A	N/A	N/A
<input type="checkbox"/>	tf-matterport-mask-rc... SLATEST	Running	N/A	Python 3.6	0	N/A	N/A	N/A	N/A

At the bottom right of the table, it says 'Rows per page: 10'.

Figure 3.3: Nuclio API to add a model.

in Figure 3.13. After, the preparation for a training session is done, a call to the Nuclio API is made. This call is set-up asynchronously since the training of a model can take quite some time, and a synchronous call will block the user’s experience.

Model Prediction The call to get predictions from a model is started from the Annotation screen (see Section 3.3.4). Since we need an immediate response back from the model, all these calls are set-up synchronously. Firstly, the UI makes a call to the backend which gathers the required modalities for the prediction. Then, the call is forwarded to the Nuclio API which loads the latest model and gets a prediction in the form of a binary list. On the way back, this prediction is translated to be visualised in the UI.

Model Incorporation Before training a model and obtaining predictions, it is of course necessary to add a model to the DAL-ART tool. This is done with the help of the Nuclio API. To deploa a model, Nuclio requires two files to be written. A function.yml file specifying what the requirements for the Docker container are and a file with the name main a function handler making the predictions.

The .yml file specifies the programming language that needs to be used and which packages should be part of the Docker container. Next to that, one should also specify the path where the models need to be stored and some other configurations for the predictions [77].

The code file can contain code from any programming language. However, due to Python’s widespread adoption as the primary framework for ML models, it was the logical choice for implementing our crack detection use case. Consequently will the following paragraphs solely focus on an integrating with Python. It’s worth mentioning that with minimal additional effort,

any programming language can be utilized.

To ease the development process, we built an interface for AI engineers to follow and easily add new models. This interface can be seen in Listing A.2. Any new model should implement a way for itself to load and save a model from/to a given path. Next to that, it should also implement the three ways to actively train the model and a way to make it predict the features of interest. At last nuclio requires a handler function to be written, which is abstracted away and is now the boiler plate code shown in Listing A.1. All the other tasks are taken care of by the DAL model handler.

The first responsibility of the DAL model handler is versioning. The implemented versioning system adopts a format consisting of two numbers separated by a dot. The first number signifies a completely new model, while the second number denotes a fine-tuned model. For instance, when retraining a model, a brand new model is created, resulting in an increment of the first version number. Consider the scenario where versions 1.0, 1.1, and 2.0 already exist, and model 1.0 is retrained. In such a case, a new model with version number 3.0 is generated. Subsequently, if transfer learning or continuous learning is applied to model 3.0, a new model is created with version number 3.1.

The second responsibility it handles is data loading. All the training functions receive a DataSet object as input, as demonstrated in Listing A.2. This DataSet is a list of samples, each containing modalities and their corresponding gold data. By abstracting away the data loading process and re-using it for a lot of models, the AI engineer can achieve a more efficient. If necessary, a custom data loading method can be implemented to cater to specific model requirements or dataloaders. Lastly, it manages the mundane tasks such as converting predictions to the CVAT format and selecting the appropriate function to invoke based on the incoming event.

There are two distinct ways to add a model: using the Nuclio UI or deploying it with Docker on the server.

The first option is to utilize the user interface provided by Nuclio, which offers a convenient graphical interface for adding models. This interface, shown in Figure 2, simplifies the process and allows users to easily upload and test their desired models in the DAL-ART tool. It provides a user-friendly experience and even includes pre-configured examples that specify the correct Python version and framework, enabling users to focus solely on their Python code. However, there are a few drawbacks to consider. Firstly, the DAL model handler needs to be copied in, instead of imported. This way of coding is not really optimal. Secondly, this approach does not support reproducibility on different servers.

Alternatively, users can opt for the more technical and robust approach by adding models through the backend using Nuclio commands. This method provides greater flexibility and control over the model integration process. By leveraging Nuclio and Docker commands, the users can precisely manage the deployment and configuration of models within the DAL-ART tool. The following DAL-ART command is then used to deploy a new model on the server:


```
nuctl deploy --project-name cvat \  
  --path serverless/tensorflow/deep_crack_detection/nuclio \  
  --volume 'pwd' / serverless/common/dal:/opt/nuclio/common \  
  --volume 'pwd' / ../ data:/opt/nuclio/data \  
  --platform local
```

The path option should contain both the function.yml and the main.py code for the model. Next to that we mount the DAL model handler so it can be imported from common, as shown in Listing A.1. At last we mount the path where the datasets are exported. This way multiple models can be trained from the same project's dataset without duplicating that data.

3.3 DAL-ART Characteristics

To review the DAL-ART tool, we start where any user starts: in the login screen, which is shown in Figure 3.4. After, a user can create a project to actively train a model in Figure 3.5. To train a supervised learning model, we of course need data which can be uploaded in the Data screen, depicted in Figure 3.7. Here data can be uploaded in different tasks. To annotate the data, the user needs to go to the Task screen in Figure 3.8 and select on which task they want to add annotations. After, the annotations can be added in the Annotation screen, shown in Figure 3.9. At last, we look how a model can be added to the DAL-ART tool and how a training process can be started through the UI (Figure 3.11). In the DAL flow, a user would then go back and annotate more data using the previously trained model.

3.3.1 Login Screen

Given the high value of annotated data and trained ML models, it is essential to protect them from unauthorized access. One way to do this is by implementing a login screen, as shown in Figure 3.4. This screen requires users to provide login credentials to access the data and trained models. If no account is created yet, one can of course create a new one.

When working in teams is required, an organization can be created to streamline collaboration and ensure that everyone has access to the same projects, data and models. To invite users to an organization, an invitation email needs to be sent.

3.3.2 Project Screen

Once we are logged in and joined the right organization, we go automatically to the projects screen shown in Figure 3.5. This screen can also be reached by clicking in the top bar on the

projects button. Here we have an overview of which projects are currently being developed. Logically we would have a 1-to-n relationship to the existing models where we start one project per feature of interest that we want to detect. Starting a new project can be done with the blue plus in the right top corner. In Figure 3.6 we would create a new project, give the project a name and specify the labels for the features of interest that we want to extract. Here the color can be set for the labels to easily differentiate the drawn masks from the original image. Additionally, the project screen allows you to create new projects, adjust their settings, and add team members to help with the annotation process.

Another very useful feature of this screen is the ability to import previously annotated datasets, or export datasets for training elsewhere in different formats. Some commonly known formats which can be used are: Segmentation mask, ImageNet, YOLO and COCO.

GAIM

DAL-ART Painting Analysis

New user? [Create an account](#)

[Forgot password?](#)

Sign in

Email or username

Next

Figure 3.4: Login screen of the DAL-ART tool.

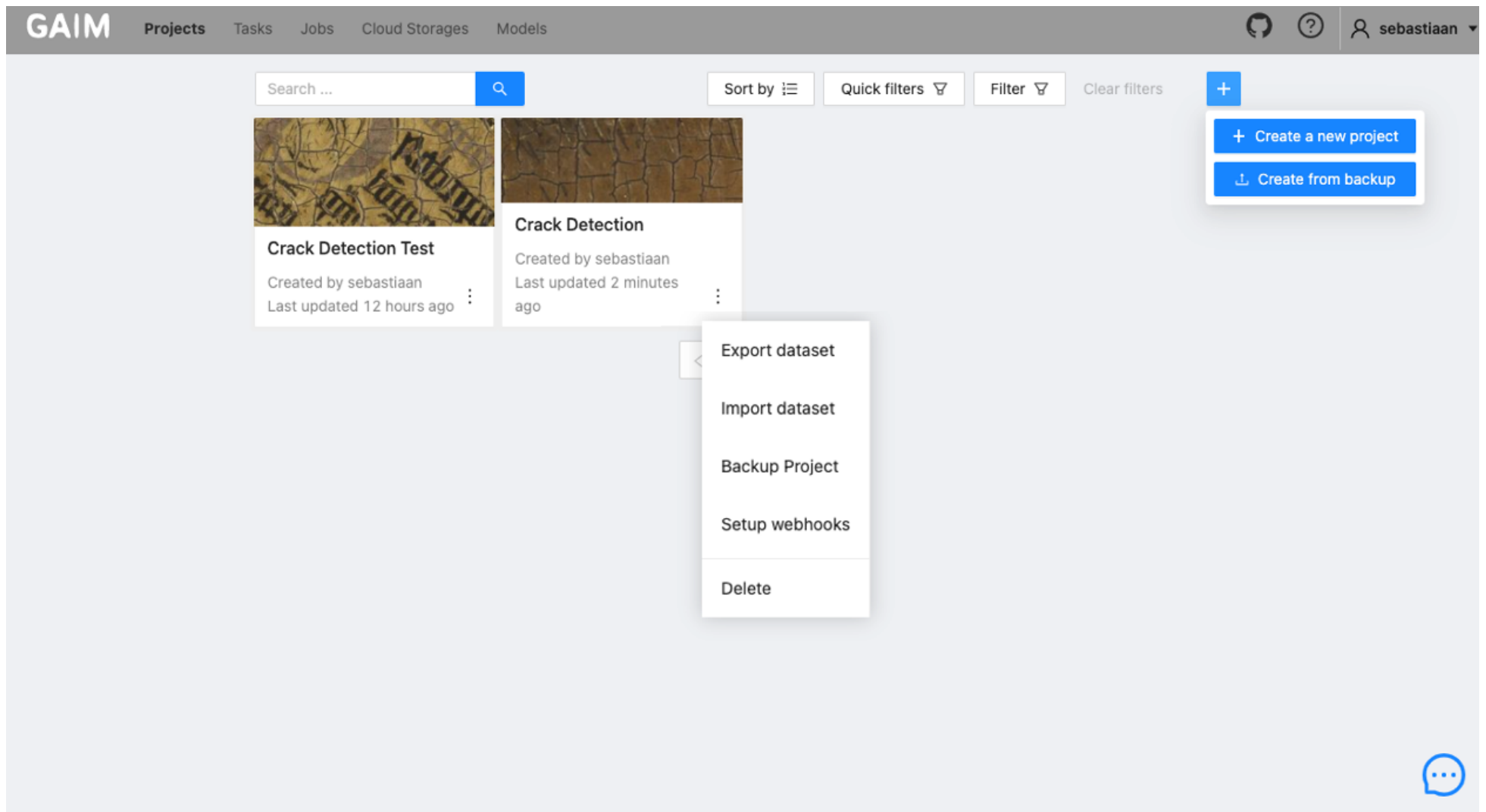


Figure 3.5: Project screen of the DAL-ART tool.

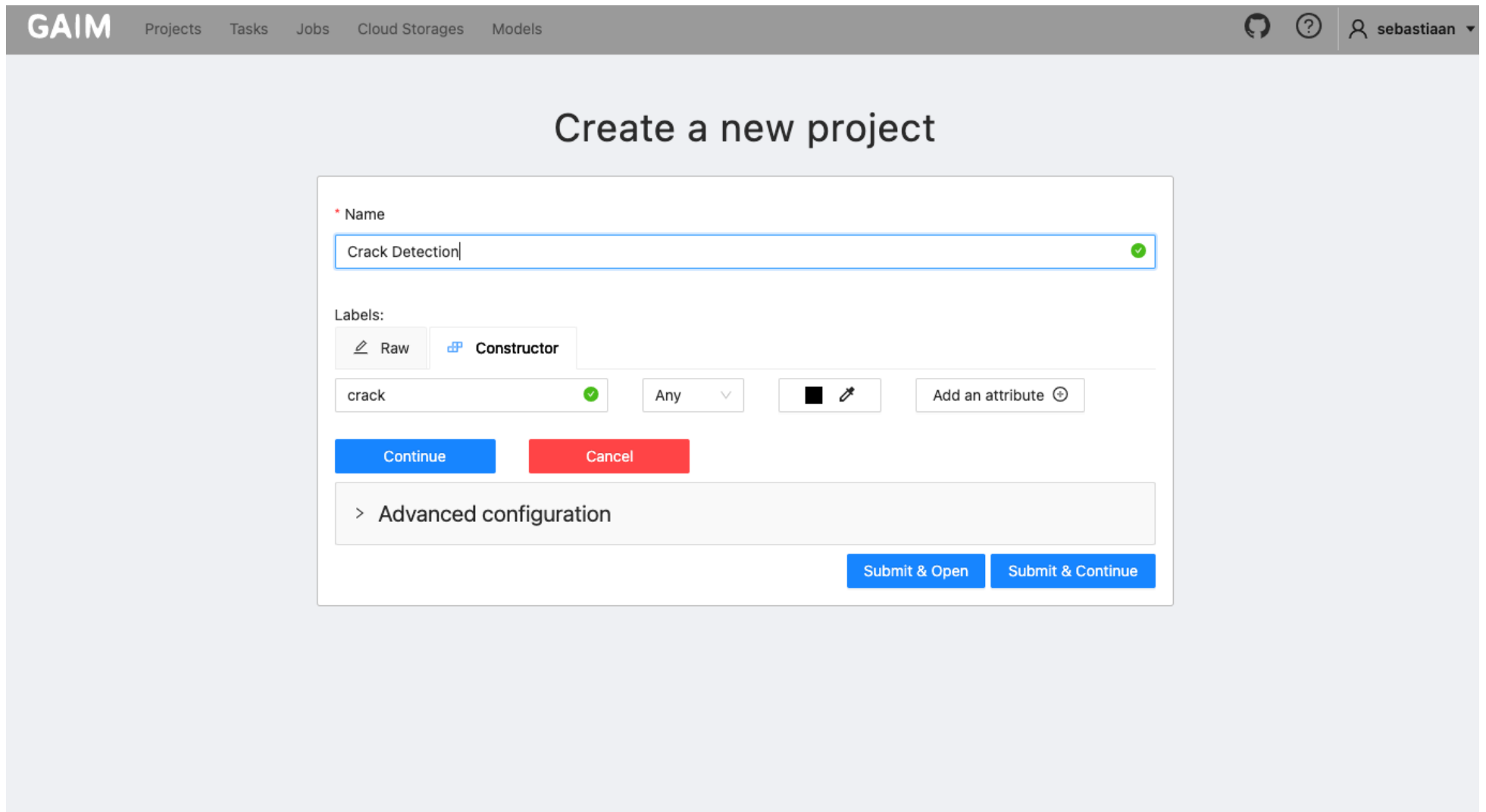






Figure 3.6: Screen to start a new project in the DAL-ART tool.

GAIM Projects Tasks Jobs Cloud Storages Models    **sebastiaan** ▾

Basic configuration


* Name
JoosVijd1438 

Project
Crack Detection

Subset
Train

Labels
Project labels will be used

* Select files
[My computer](#) [Connected file share](#) [Remote sources](#) [Cloud Storage](#)



Click or drag files to this area
You can upload an archive with images, a video, or multiple images




-  JoosVijd1438_2009_1VIS_registered.tif
-  JoosVijd1438_2009_IRR.tif
-  JoosVijd1438_2009_XRR_enhanced_registered.tif

Figure 3.7: Screen to add new data to the project in the DAL-ART tool.

3.3.3 Task Screen

The task screen serves as the platform for adding and organizing the different annotation tasks required for each project. When accessing this screen, all the tasks for all the projects are displayed, making it easier to manage and oversee different annotation processes. However, if you want to focus on the tasks associated with a specific project, you can simply click on that project's name in the project screen. By doing so, you will be directed to a new screen that only displays the tasks linked to that particular project.

To add data to the project and create a new task, click the blue plus button located in the top right corner of the screen which takes you to Figure 3.7. When adding new data, it is crucial that the appropriate modalities that the model needs for training are uploaded in the right order; otherwise, we could see inconsistencies and unexpected behaviour during training and inference time. The expected order for the crack detection model is first the RGB image, then the infra-red image and lastly the x-ray image. If too many modalities are present, the model will only use the first ones uploaded.

Files can be uploaded from local sources, cloud sources, or remote sources such as GitHub URLs. If desired, a partially annotated dataset can also be uploaded, with the original files and the corresponding annotations.

3.3.4 Annotation Screen

When you select a task and its corresponding job, you will be taken to the annotation screen - which can be seen in Figure 3.9 - where you can create a mask for a patch of the digital painting. To make sure you have the clearest view, you can use the arrows on top of the screen to scroll through the different modalities. Once you have selected the appropriate modality, you can start drawing a mask with the pencil tool located on the left-hand side of the screen. Clicking on the pencil tool will prompt a pop-up where you can select the size of the pencil. You can also select the gum tool to correct any errors that have been made by you or the model.

For areas that are more difficult to annotate, you can zoom in using the mouse-pad or scroll-wheel. If you want a clearer view of a certain part, you can adjust the image's saturation, contrast, or brightness. This can be done through the panel that pops-up at the bottom of the screen.

On the right-hand side of the screen, there is a panel that allows you to adjust the mask's opacity. This feature comes in very handy when trying to distinguish between different features of interest such as cracks and the original picture.

During training, the different masks of one patch are added together using an OR operator. Therefore, it is best to annotate areas where the features of interest, such as cracks, are clearest to annotate when creating the mask. For instance, when detecting cracks, the infra-red image

may show the straight lines more clearly, while the RGB image provides a cleaner overview of the corners where the cracks intersect. At last, there is an x-ray image, showing variations in height, highlighting the valleys formed by the cracks. If you have initiated the annotation process on the RGB modality, but wish to continue annotating on the X-ray image for example, you can click the propagate button located on the right-hand side of the screen, as illustrated in Figure 3.10. This action will trigger a pop-up window, allowing you to select the modality to which you want to propagate the selected mask. The available options are depicted in Figure 3.12. First, you can specify the number of modalities to which you wish to propagate the mask, followed by selecting the desired target modality. It is crucial to delete the previous mask when making corrections, as we utilize an OR operator on all annotated masks at prediction time. Failing to delete the old mask could result in the older mask from a different modality overwriting the correction made on the newer mask.

In Figure 3.10 the last and maybe most powerful feature of the annotation screen is visualised. Here, the patch is automatically annotated with a CNN model in the first iteration of its active learning process. To use this feature, you should click on the wand on the left-hand side of the screen. Once clicked, a pop-up will appear allowing you to select a model to use for automatic annotations. By doing this, the annotation task is reduced from full annotations to cleaning of the automatically generated mask.




3.3.5 Model Screen






To access the different models available, click on the Models button located on the top bar of the screen. Once there, you will be able to see a list of available models, along with a description, supported labels and their corresponding framework. This is shown in Figure 3.11.





At the end of the list, there is an actions drop-down that provides the option to start an active training session for the model. As outlined in Section 2.2, three methods of active training are available, including re-learning, continuous learning and transfer learning.

If you select a training option, a modal will appear prompting you to select the version that requires retraining and from which project the training data needs to be taken, which can be seen in Figure 3.13. If no project or version is selected before starting a training session, the error prompt will appear on top of the screen, asking you to choose a project and version.

For older models that do not support active learning, it is not possible to initiate a training session. In such cases, the text "No Retractable Versions" will appear when selecting a version, indicating that the model is not compatible with active learning.

GAIM Projects **Tasks** Jobs Cloud Storages Models    **sebastiaan** ▾

Search ...  Sort by  Quick filters  Filter  Clear filters 

	#22: original_frame Created by sebastiaan on May 22nd 2023 Last updated a few seconds ago	 ● Pending 0 of 1 jobs	Open Actions ⋮
	#17: joosvijd Created by sebastiaan on May 22nd 2023 Last updated 3 hours ago	 ● Pending 0 of 1 jobs	Open Actions ⋮


< 1 > 

Figure 3.8: Screen to have an overview of the tasks.

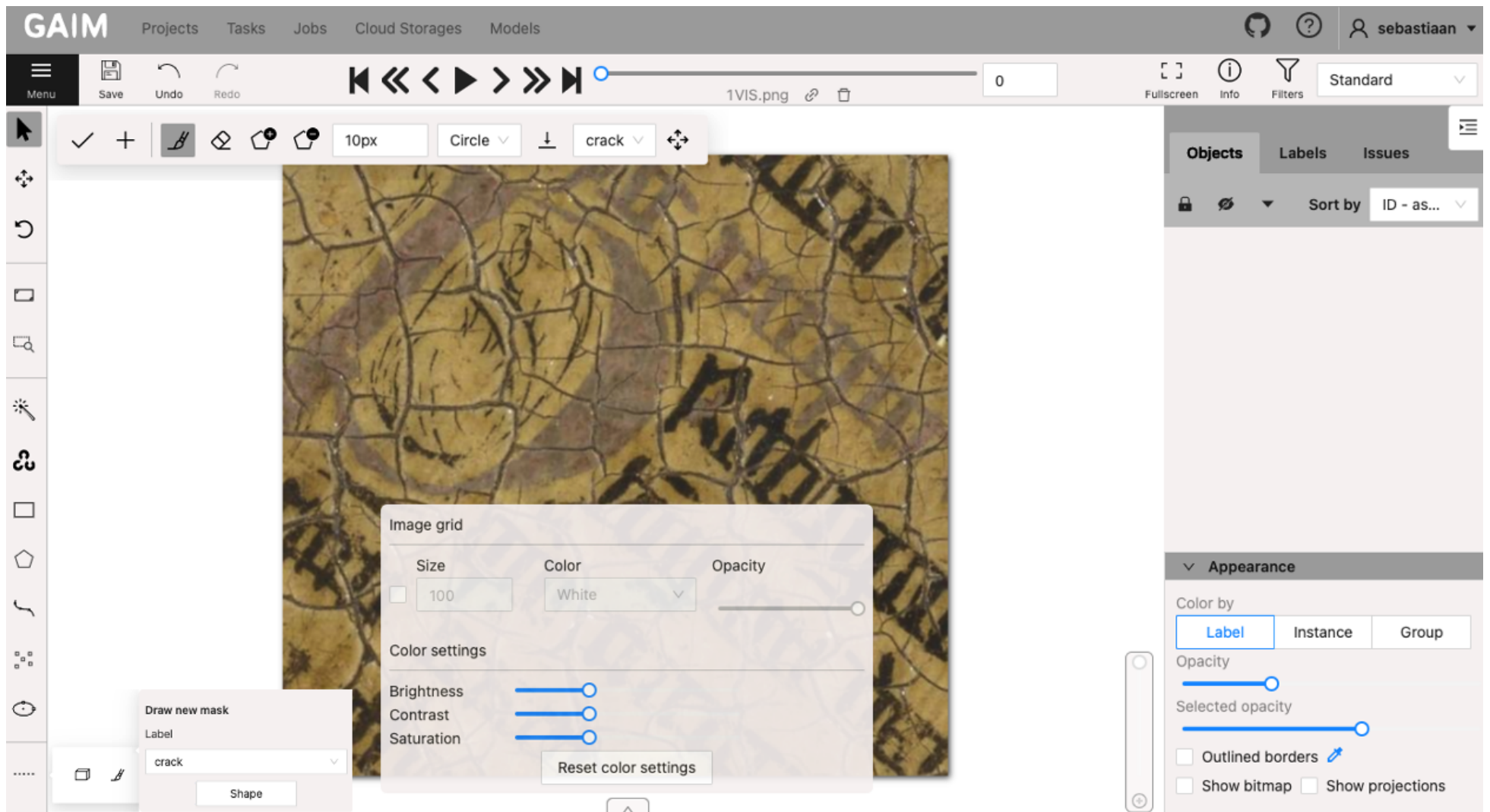


Figure 3.9: Screen to annotate new data in the DAL-ART tool.

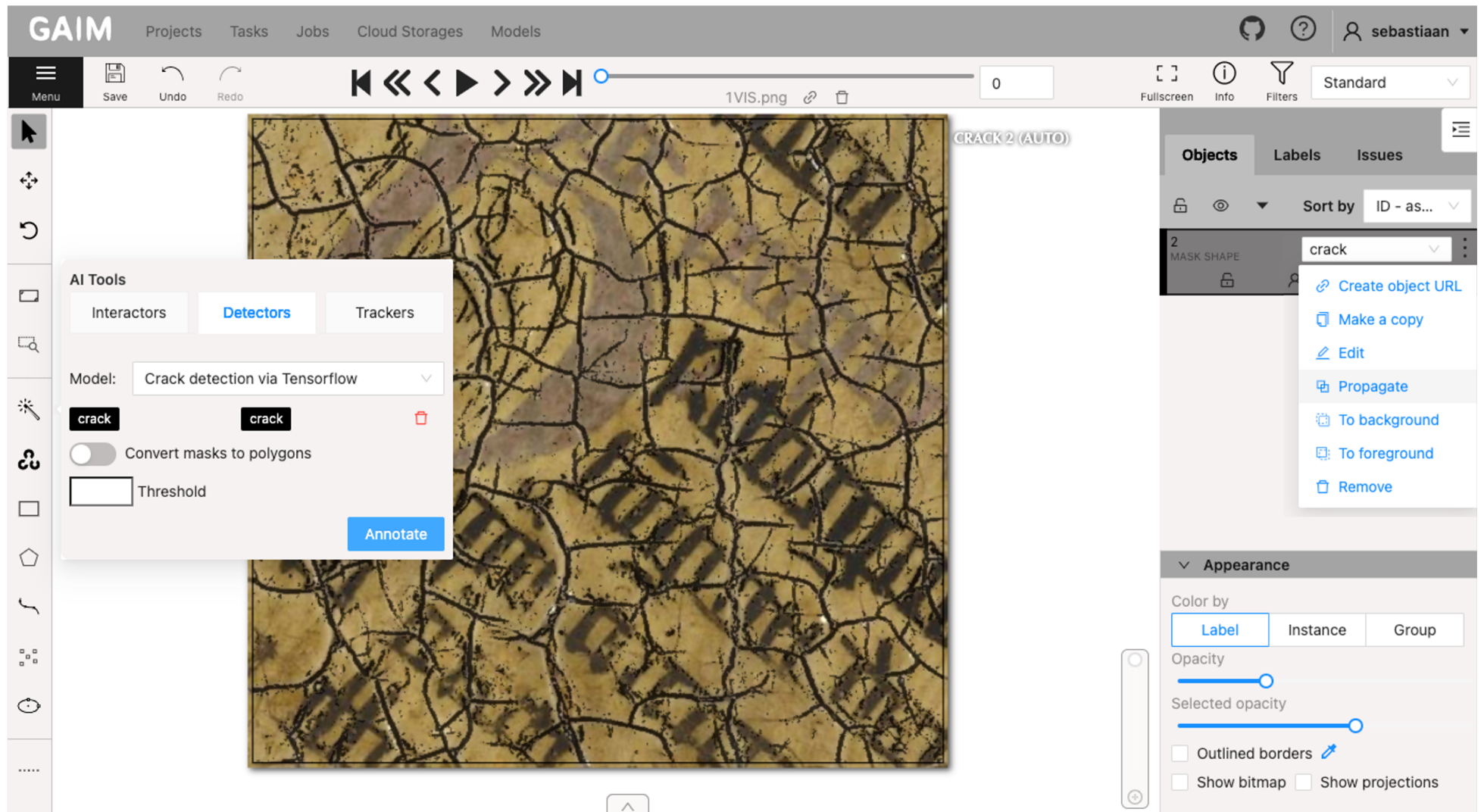





Figure 3.10: Annotate new data with the previously trained model.

GAIM Projects Tasks Jobs Cloud Storages **Models**    **sebastian** ▾

Framework	Name	Type	Description	Labels	Actions
tensorflow	Crack detectio...	detector	DAL implementation of crack detection on Python 3, Keras, and TensorFlow.	Supported labels ▾	Actions Re-learn Continuously learn Transfer learn
tensorflow	Faster RCNN vi...	detector	Faster RCNN from Tensorflow Object Detection API	Supported labels ▾	
tensorflow	Mask RCNN via...	detector	An implementation of Mask RCNN on Python 3, Keras, and TensorFlow.	Supported labels ▾	Actions




Figure 3.11: Choose how to actively train a model in the DAL-ART tool.

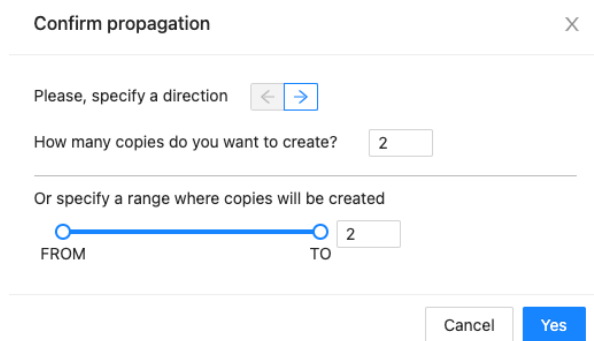


Figure 3.12: Pop-up for propagating a mask to the following page.

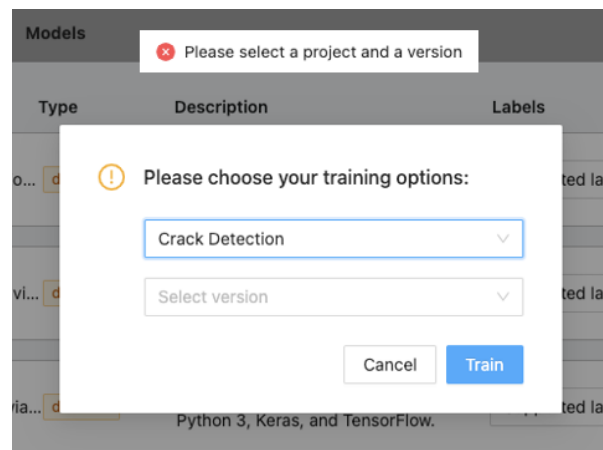


Figure 3.13: Error handling in the case of an invalid configuration.

4

EXPERIMENTAL RESULTS AND DISCUSSION

This chapter focuses on the execution and experiments with the previously described models, techniques, and DAL-ART tool, aiming to present how to best use it in practice. Building upon the features presented in Chapter 3, we start by annotating some new data with the tool in Section 4.2. Then, in Section 4.3, we use different active training techniques on the CNN model developed by Sizyakin et al. [7] to improve it on the parts where it needs active learning. We end this chapter with a conclusion and some discussions about our experiments and the DAL-ART tool in Section 4.4.

4.1 Experimental Setup

When following the active learning flow previously shown in Figure 3.1, our iterative process begins with manual annotation of new data. Subsequently, we actively train a model and repeat this cycle. In our experiments, we will follow the same flow for one of these iterations. Initially, we start from new data along with a previous checkpoint from a crack detection model developed by Sizyakin et al. [7]. Next, we will employ the tool to actively annotate a portion of this data and assess the process and its outcomes. After, we deploy three active learning techniques that have been identified and developed, evaluating their performance in terms of active learning, generalization, and catastrophic forgetting. For the experiments we used data taken from the Book, Joos Vijd and Singing Angels panels.

As a consequence of certain technicalities with UGent domain names, the DAL-ART tool was not publicly accessible yet during the experiments. This limited our ability to conduct an very in-depth experiment with real users on how the annotation and active learning procedure would change with its usage. Nonetheless did we evaluate it to the fullest extent possible.

4.2 Actively Annotating

As discussed in Chapter 2, the current SOTA models for crack detection show impressive capabilities. However, they face challenges such as limited data availability and a lack of full generalization when encountering data drift or the need for predictions on new panels. For that reason it is very important that we have a straightforward method for augmenting our dataset. To assess its performance, we compare the usability, annotation time, and visual quality of annotations across various annotation methods. To conduct this evaluation, we performed a small comparative analysis by annotating three patches using an old annotation method, the DAL-ART method without active learning, and the DAL-ART method with active learning assistance. Previously, annotations were done using tools like GNU Image Manipulation Program (GIMP) [79]. GIMP is a powerful, free, and open-source software widely used for editing and manipulating digital images. It offers a variety of tools and features for image retouching, resizing, and advanced editing, making it a popular choice when manipulating images.

Unfortunately, we do not have precise time measurements for the annotation process using GIMP, as it was never measured at that time. However, based on estimations, it took approximately 10-20 minutes to annotate a 256×256 patch, which gives an average time of 15 minutes. The other measurements are visualised in Table 4.1. The average time required for annotating with the DAL-ART tool is close to 20 minutes, as shown in the average time column. This is slightly higher than the GIMP annotations. On average, we observe a 49% improvement in annotation time compared to the GIMP annotation process, while achieving a 58% gain compared to the manual annotation process when using the DAL-ART tool.

One possible reason for the longer annotation time of the DAL-ART annotation without active annotations is that the annotations created with GIMP are often intentionally incomplete. Figure 4.1 displays the different annotations clearly, indicating that some cracks are missing or are too small due to the challenging nature of crack annotations. Fortunately is the CNN model, when combined with proper pre-processing techniques, relatively resilient to these incomplete annotations and can still learn from such data. However, providing more accurate data will only benefit the model further. For more recent models the research state that precise and accurate annotations are crucial for achieving good predictions, especially when using a U-Net model [8].

	Annotation time 1	Annotation time 2	Annotation time 3	Average time
GIMP	10-20 min	10-20 min	10-20 min	15min
Manual DAL-ART	16min 20s	18min 31s	20min 15s	18min 22s
Active DAL-ART	6min 41s	8min 55s	7min 9s	7min 35s

Table 4.1: Measured annotation times for three patches of the Ghent Altarpiece.



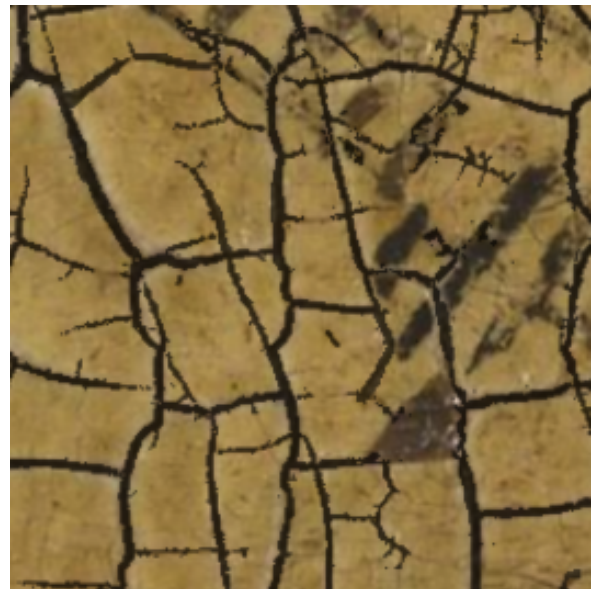
(a) Original image of patch.



(b) GIMP annotated patch.



(c) Manual DAL-ART annotated patch.



(d) Active DAL-ART annotated patch.

Figure 4.1: Comparison of annotated patch 3 of Book 16 taken from the Ghent Altarpiece.

If we compare the different annotations visually in Figure 4.1, are the annotations generated through active learning arguably the most precise and finely detailed. It captured the finest lines because the used model tends to over predict a little bit.

When comparing DAL-ART and GIMP for usability, both tools offer the same basic image editing features like brushes, zooming, and visibility adjustments. However, GIMP provides

more advanced image editing features. The main advantage of the DAL-ART tool over GIMP is the decision to have a web-based tool. This allows multiple annotators to collaborate on the same project, which is beneficial when working with larger datasets. Additionally enables the DAL-ART's setup anyone with any hardware setup and an internet connection to perform annotation work. In contrast can setting up GIMP on different operating systems be more challenging and require extra work.

4.3 Actively Learning

In Section 2.2, we discussed how we can actively train a model in three different ways: re-learning, continuously learning, and transfer learning. We will now test the CNN model created by Sizyakin et al. [7] for these 3 active learning strategies with the DAL-ART tool. To compare how much information the model can retain and how well it can handle catastrophic forgetting, we will measure its performance on the patch shown in Figure 3.10. As explained in the research paper and shown in the picture, the predictions can be very accurate on data close to the training set. However, for some patches, active learning is necessary. One of those patches we identified is shown in Figure 4.2a, where there are clearly some mistakes in the predictions.

4.3.1 Re-learning

When re-learning, use the newly annotated patch to re-train our model from scratch, along with the old dataset. Then we will evaluate how this affects the performance of our model on three different types of patches: the original patch that we used to train the model initially, the new patch that we added through active learning, and a generalization patch that is similar to the new patch but has not been seen by the model before.

The results of our experiment in Figure 4.2b, Figure 4.3b, and Figure 4.4b. Each figure shows the predictions of our model on the corresponding patch of data. We can observe that adding the new patch through active learning improves the performance of our model significantly on both the new patch and the generalization patch. This means that our model can learn from the new data and generalize to unseen data that has similar characteristics. However, we also notice that our model suffers from catastrophic forgetting, which means that it forgets some of the information that it learned from the old dataset. This is mainly because the old dataset has changed over time and does not contain the original training sample anymore. This is a common challenge in ML problems, especially when the data distribution changes over time. Therefore, we do not recommend re-learning for the CNN model, as it may lose some of its previous knowledge and accuracy.



(a) Prediction with old CNN model.



(b) Prediction with re-learned model.



(c) Prediction with continuously learned model.



(d) Prediction with transfer learned model.

Figure 4.2: Different predictions after active learning on the JoosVijd panel patch 1438 taken from the Ghent Altarpiece.

4.3.2 Continuously Learning

We can now use continuously learning to fine-tune our model. However, instead of training the model from scratch, we start from the weights of the old model that we trained on the old dataset. We want to see how this affects the performance of our model on the same types of



(a) Prediction with old CNN snapshot.



(b) Prediction with re-learned model.



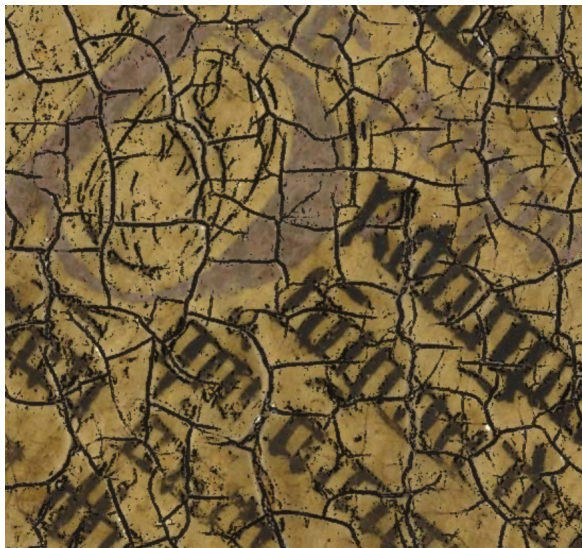
(c) Prediction with continuously learned model.



(d) Prediction with transfer learned model.

Figure 4.3: Different predictions after active learning on the JoosVijd panel patch 1518 taken from the Ghent Altarpiece.

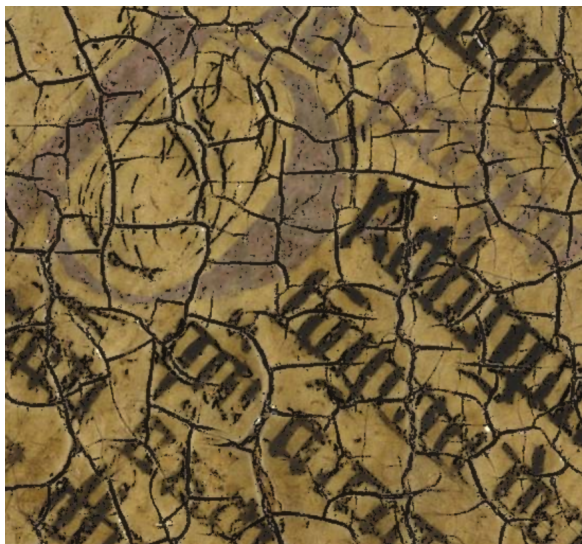
patches as before. The results of our experiment can be seen in Figure 4.2c, Figure 4.3c, and Figure 4.4c. We can see that fine-tuning our model on the new patch through continuously learning has similar results on both the new patch and the generalization patch as re-learning. This means that our model can still learn from this and generalize. Moreover, we can also see that our model does not suffer from catastrophic forgetting, which means that it remembers



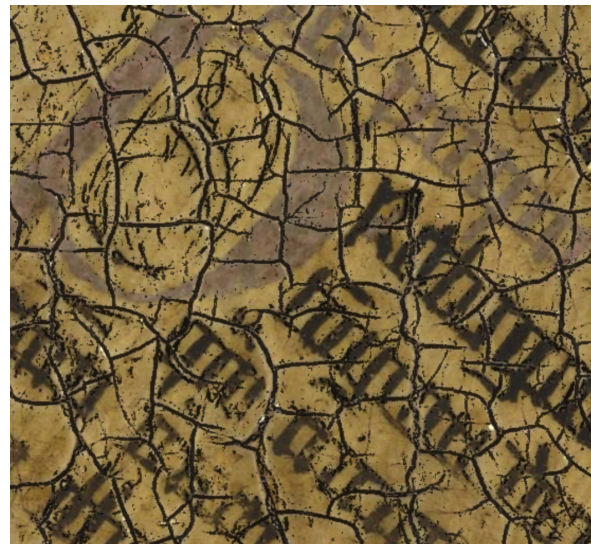
(a) Prediction with original snapshot.



(b) Prediction with re-learned model.



(c) Prediction with continuously learned model.



(d) Prediction with transfer learned model.

Figure 4.4: Different predictions after active learning on an old training patch taken from the Ghent Altarpiece.

most of the information that it learned from the old dataset.

4.3.3 Transfer Learning

Transfer learning is another technique that allows us to use a pre-trained model and fine-tune it to a new task. In our case, we use transfer learning to modify the CNN model that we trained

on the old dataset. However, instead of fine-tuning the full model, we only fine-tune the last 4 layers which is the FCNN part, which is the classifier that makes the final predictions. The CNN part, which is the feature extractor that extracts the relevant features from the data, is frozen and not updated. We again want to see how this affects the performance of our model on the same types of patches as before.

The results of this experiment can be seen in Figure 4.2d, Figure 4.3d, and Figure 4.4d. We can see that transfer learning our model on the new patch does not improve the performance of our model a lot on any of the patches. This means that our model cannot adapt well to the new data by only changing the FCNN part. So, we froze too many layers and did not allow enough flexibility for the model to learn new features. Since continuously learning already gives us good very good results without almost any catastrophic forgetting, it is not worth to explore transfer learning further for this model as it would not give us any benefit.

4.4 Discussions

Our experiments show that the DAL-ART tool and the active learning of the CNN model can be very effective. The tool also makes it easy and fast to conduct further experiments, as it offers a smooth user experience. For the case of actively annotating, the DAL-ART tool shows, particularly when combined with active learning assistance, improved annotation efficiency and precision compared to traditional methods like GIMP. Additionally is its usability slightly better because of the DAL-ART's web-based nature. It is important to note that these findings are specific to the CNN model used in this evaluation. Different models employed for other digital painting analysis tasks or in earlier iterations of active learning process may yield different results, potentially with slightly lower performance gain.

When actively training on this data, we can see that the CNN benefits from that while retaining most information on previously learned data. Next to that it shows quite good generalisation capabilities when actively learning. The best method based on this one patch example was clearly continuously learning as the new data set is not exactly the same as the original one and transfer learning would not add a lot of benefit since the model does not suffer from catastrophic forgetting too much. It has some drawbacks however where active learning does not seem to resolve the problems with the noisy predictions. Further research is needed to see if a bigger model or newer model like U-Net [8] can reduce the noise in the predictions while actively learning. Another drawback of the model is that the optimal threshold is different after every training process. This makes evaluating the model quite difficult.

At last we have a look at the robustness of the DAL-ART tool and the CNN model. Ensuring the robustness of software research is a critical factor when transitioning from the research phase to a practical implementation, particularly for a UI designed to be actively used by diverse

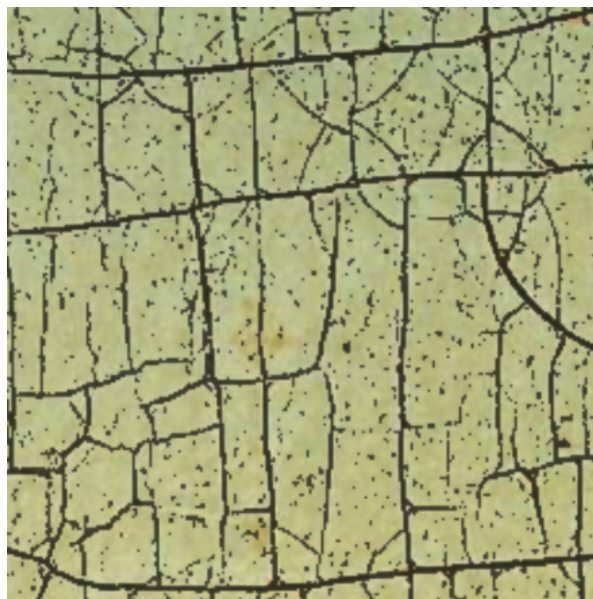
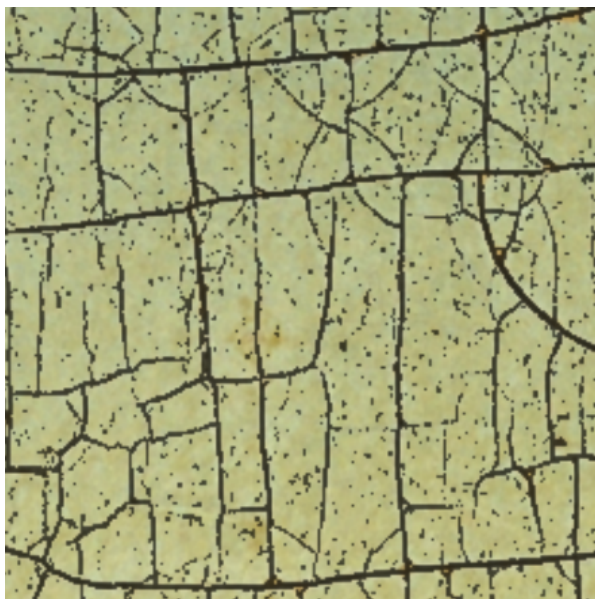


Figure 4.5: Predictions based on 3 modalities. Figure 4.6: Predictions based on 1 modality.

groups with varying backgrounds. Throughout the course of our experiments, we did a thorough assessment of the code’s robustness with the outcome of a good user experience without any identified bugs.

Moreover, we specifically focused on testing the robustness of the CNN implementation within the DAL-ART tool. Our attention turned to addressing a common challenge encountered in real-world scenarios, where certain digital panels may lack specific modalities. To accommodate this, we extended the implementation to allow the model to make accurate predictions solely based on the visual modality.

Figure 4.5 visually demonstrates the predictions based on the integration of three modalities, while Figure 4.6 showcases the predictions based on a single modality. Upon observation, the differences between the two are barely visible to the human eye. There is a slight change of only 15 pixels, which translates to a negligible loss of 0.02% in accuracy.

During the training phase, we can encounter the same problem. Unfortunately would allowing the model to be trained with less modalities require an entirely new model, which would undermine the purpose of utilizing the existing model. To overcome this, we implemented a solution wherein the model logs the patches that lack certain modalities during training. So these patches can be skipped during the training process, allowing the training to proceed smoothly. This adaptive approach ensures that the model continues to learn effectively despite the absence of certain modalities in specific patches.

By conducting these tests and thoroughly evaluating the robustness of the DAL-ART tool and the CNN implementation, we have confirmed its reliability and effectiveness in generating accurate predictions, even when dealing with incomplete modalities in new panels. This ensures the tool's practicality and its potential for successful other real-world applications on various feature detection tasks in digital paintings.

5

CONCLUSION AND FUTURE WORK

Researchers have been investigating automatic crack detection on the Ghent Altarpiece for the last years to replace the burden of manually identifying these cracks. They realized that using automated techniques could help identify and analyze cracks on this famous artwork, which opened up exciting possibilities for art restoration and conservation. To wrap up this thesis, we reflect on the significant impact of this work in the field and the valuable contributions it has made to preserving and understanding the Ghent Altarpiece, as well as the wider field of art conservation.

This master's thesis has successfully achieved its primary objective of creating a practical DAL framework with the DAL-ART tool. The framework has been designed and implemented to seamlessly integrate an existing deep learning model into an active learning environment. By enabling users to view and correct the model's predictions through annotations, a model can actively improve over time. Essential features, such as zooming into images, scrolling through multi-modal inputs, adjusting transparency settings for annotated masks, and active annotating have been incorporated to facilitate effective annotation. At last makes the web-based nature of the tool it usable for everyone without any setup cost.

Upon the completion of the tool, we successfully integrated a SOTA crack detection model and actively trained it on patches in a remarkably efficient manner. This demonstrated the robustness of the tool and how it can be used by other researchers for any multi-modal digital painting analysis task. Next to that we showed that the most effective approach to enhance the model's performance was through continuous learning, showing generalization capabilities on certain patches while avoiding catastrophic forgetting.

The successful completion of this thesis sets the stage for further advancements in the fields of active learning and crack detection research. The developed DAL-ART framework not only accomplishes the integration of deep learning models into interactive environments but also establishes a foundation for more effective collaboration between AI developers and end-users

allowing for efficient future work. To that extent we now propose several directions for future work and research to elevate crack detection research and the tool to new heights. We present two research-focused directions, followed by one pragmatic improvement aimed at enhancing the usability and efficiency of the tool:

- **Expanding CNN Experiments:** This thesis already conducted several experiments using the SOTA CNNs for crack detection, showing very promising results. There is a scope for further exploration however as the experiments were done on a limited dataset. It is recommended that future work includes a more comprehensive exploration of CNN-based approaches, enabling a deeper understanding of their potential and further refining the models used in the DAL-ART tool.
- **Exploring Newer Model Experiments:** Due to the substantial time investment in designing and building a production-ready DAL-ART tool, there was no opportunity to conduct experiments on the latest SOTA U-Net models. However, the literature highlights the high promise of these newer models. Exploring and experimenting with these advanced U-Net architectures can provide valuable insights into their performance and potential enhancements in crack detection with DAL.
- **Enhancing Model Training:** While the current model training process is functional, there is room for improvement to optimize its effectiveness. Currently, the training logs are simply written to a log file, but a more favorable approach would be to integrate the training process with a dedicated training API, such as W&B (Weights & Biases). By leveraging a training API, the model training process can be elevated to a more efficient and streamlined workflow, empowering researchers to achieve better results and insights.

BIBLIOGRAPHY

- [1] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, *et al.*, “Deep learning for computer vision: A brief review,” *Computational intelligence and neuroscience*, vol. 2018, 2018.
- [2] B. Cornelis, A. Doms, J. Cornelis, F. Leen, and P. Schelkens, “Digital painting analysis, at the cross section of engineering, mathematics and culture,” in *2011 19th European Signal Processing Conference*, IEEE, 2011, pp. 1254–1258.
- [3] “The most stolen work of art.” (), [Online]. Available: <https://www.britannica.com/story/the-most-stolen-work-of-art>. (accessed: 22.04.2023).
- [4] B. Cornelis, T. Ružić, E. Gezels, A. Doms, A. Pižurica, L. Platiša, J. Cornelis, M. Martens, M. De Mey, and I. Daubechies, “Crack detection and inpainting for virtual restoration of paintings: The case of the ghent altarpiece,” *Signal Processing*, vol. 93, no. 3, pp. 605–619, 2013.
- [5] A. Pizurica, L. Platisa, T. Ruzic, B. Cornelis, A. Doms, M. Martens, M. De Mey, and I. Daubechies, “Virtual restoration and mathematical analysis of pearls in the adoration of the mystic lamb,” in *Het Lam Gods Series of Lectures*, 2014.
- [6] A. Pizurica, L. Platisa, T. Ruzic, B. Cornelis, A. Doms, M. Martens, H. Dubois, B. Devolder, M. De Mey, and I. Daubechies, “Digital image processing of the ghent altarpiece: Supporting the painting’s study and conservation treatment,” *IEEE Signal Processing Magazine*, vol. 32, no. 4, pp. 112–122, 2015.
- [7] R. Sizyakin, B. Cornelis, L. Meeus, H. Dubois, M. Martens, V. Voronin, *et al.*, “Crack detection in paintings using convolutional neural networks,” *IEEE Access*, vol. 8, pp. 74 535–74 552, 2020.
- [8] R. Sizyakin, V. Voronin, and A. Pižurica, “Virtual restoration of paintings based on deep learning,” in *Fourteenth International Conference on Machine Vision (ICMV 2021)*, SPIE, vol. 12084, 2022, pp. 422–432.
- [9] R. Sizyakin, V. Voronin, A. Zelensky, and A. Pižurica, “Virtual restoration of paintings using adaptive adversarial neural network,” *Journal of Electronic Imaging*, vol. 31, no. 4, p. 043 025, 2022.

- [10] T. Ruzic, B. Cornelis, L. Platisa, A. Pizurica, A. Doods, M. Martens, M. De Mey, and I. Daubechies, “Craquelure inpainting in art work,” eng, in *Vision and material : interaction between art and science in Jan van Eyck’s time, Abstracts*, Brussels, Belgium, 2010.
- [11] L. Platisa, B. Cornelis, T. Ruzic, A. Pizurica, A. Doods, M. Martens, M. De Mey, and I. Daubechies, “Spatio-gram features to characterize pearls and beads and other small ball-shaped objects in paintings,” eng, in *Vision and material : interaction between art and science in Jan Van Eyck’s time*, ser. Speciale Uitgaven, M. De Mey, M. Martens, and C. Stroo, Eds., vol. 6, KVAB PRESS, 2012, pp. 315–329.
- [12] T. Ruzic and A. Pizurica, “Context-aware image inpainting with application to virtual restoration of old paintings,” eng, in *IEICE Information and Communication Technology Forum, Proceedings*, Sarajevo, Bosnia&Herzegovina, 2013.
- [13] S. Huang, W. Liao, H. Zhang, and A. Pizurica, *Paint loss detection in old paintings by sparse representation classification*, eng, 2016.
- [14] L. Meeus, S. Huang, N. Zizakic, X. Xie, B. Devolder, H. Dubois, M. Martens, and A. Pizurica, “Assisting classical paintings restoration: Efficient paint loss detection and descriptor-based inpainting using shared pretraining,” eng, in *Optics, Photonics and Digital Technologies for Imaging Applications VI*, P. Schelkens and T. Kozacki, Eds., vol. 11353, France, 2020, p. 12.
- [15] S. Huang, B. Cornelis, B. Devolder, M. Martens, and A. Pizurica, “Multimodal target detection by sparse coding: Application to paint loss detection in paintings,” eng, *IEEE Transactions on Image Processing*, vol. 29, pp. 7681–7696, 2020.
- [16] Sizyakin, Roman, “Deep learning methods for crack detection and image restoration with application to digitized paintings,” eng, Ph.D. dissertation, Ghent University, 2022, XXIV, 136.
- [17] “The ghent altarpiece by jan van eyck.” (), [Online]. Available: <https://www.worldhistory.org/image/12908/the-ghent-altarpiece-by-jan-van-eyck/>. (accessed: 22.04.2023).
- [18] B. Settles, “Active learning,” *Synthesis lectures on artificial intelligence and machine learning*, vol. 6, no. 1, pp. 1–114, 2012.
- [19] X. Zhan, Q. Wang, K.-h. Huang, H. Xiong, D. Dou, and A. B. Chan, “A comparative survey of deep active learning,” *arXiv preprint arXiv:2203.13450*, 2022.
- [20] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, “Deep learning for generic object detection: A survey,” *International journal of computer vision*, vol. 128, pp. 261–318, 2020.
- [21] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, “Image segmentation using deep learning: A survey,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 7, pp. 3523–3542, 2021.

- [22] J. Huyan, W. Li, S. Tighe, Z. Xu, and J. Zhai, “Cracku-net: A novel deep convolutional neural network for pixelwise pavement crack detection,” *Structural Control and Health Monitoring*, vol. 27, no. 8, e2551, 2020.
- [23] “Computer vision tutorial: A step-by-step introduction to image segmentation techniques (part 1).” (), [Online]. Available: <https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/>. (accessed: 09.04.2023).
- [24] T. M. Mitchell *et al.*, *Machine learning*. McGraw-hill New York, 2007, vol. 1.
- [25] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*. John Wiley & Sons, 2021.
- [26] D. G. Kleinbaum, K. Dietz, M. Gail, M. Klein, and M. Klein, *Logistic regression*. Springer, 2002.
- [27] S. R. Gunn *et al.*, “Support vector machines for classification and regression,” *ISIS technical report*, vol. 14, no. 1, pp. 5–16, 1998.
- [28] Z. Zhang and M. Sabuncu, “Generalized cross entropy loss for training deep neural networks with noisy labels,” *Advances in neural information processing systems*, vol. 31, 2018.
- [29] R. Sizyakin, V. Voronin, N. Gapon, A. Zelensky, and A. Pizurica, “A deep learning-based approach for defect detection and removing on archival photos,” in *2020 IS&T International Symposium on Electronic Imaging (EI 2020)*, 2020, pp. 029–1.
- [30] OpenAI, *Gpt-4 technical report*, 2023.
- [31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [32] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural networks*, vol. 61, pp. 85–117, 2015.
- [33] J. Janssens, “Demystifying convolutional neural networks using sparse coding: Application to medical image segmentation,” 2021.
- [34] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” *arXiv preprint arXiv:1710.05941*, 2017.
- [35] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1998.
- [36] S. K. Kumar, “On weight initialization in deep neural networks,” *arXiv preprint arXiv:1704.08863*, 2017.
- [37] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.
- [38] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in *Proceedings of COMPSTAT’2010: 19th International Conference on Computational Statistics Paris France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, Springer, 2010, pp. 177–186.

- [39] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [40] V. H. Phung and E. J. Rhee, “A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets,” *Applied Sciences*, vol. 9, no. 21, 2019.
- [41] M. A. Nielsen, *Neural networks and deep learning*. Determination press San Francisco, CA, USA, 2015, vol. 25.
- [42] “Image segmentation with monte carlo dropout unet and keras.” (), [Online]. Available: https://nchlis.github.io/2019_10_30/page.html. (accessed: 20.04.2023).
- [43] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, “Imagenet large scale visual recognition challenge,” *International journal of computer vision*, vol. 115, pp. 211–252, 2015.
- [44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.
- [45] M. Chhabra and R. Kumar, “An advanced vgg16 architecture-based deep learning model to detect pneumonia from medical images,” in *Emergent Converging Technologies and Biomedical Systems: Select Proceedings of ETBS 2021*, Springer, 2022, pp. 457–471.
- [46] E. Rezende, G. Ruppert, T. Carvalho, A. Theophilo, F. Ramos, and P. d. Geus, “Malicious software classification using vgg16 deep neural network’s bottleneck features,” in *Information Technology-New Generations: 15th International Conference on Information Technology*, Springer, 2018, pp. 51–59.
- [47] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Medical Image Computing and Computer-Assisted Intervention—MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, Springer, 2015, pp. 234–241.
- [48] H. Cao, Y. Wang, J. Chen, D. Jiang, X. Zhang, Q. Tian, and M. Wang, “Swin-unet: Unet-like pure transformer for medical image segmentation,” in *Computer Vision—ECCV 2022 Workshops: Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part III*, Springer, 2023, pp. 205–218.
- [49] X. Li, H. Chen, X. Qi, Q. Dou, C.-W. Fu, and P.-A. Heng, “H-denseunet: Hybrid densely connected unet for liver and tumor segmentation from ct volumes,” *IEEE transactions on medical imaging*, vol. 37, no. 12, pp. 2663–2674, 2018.
- [50] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, “Deep learning applications and challenges in big data analytics,” *Journal of big data*, vol. 2, no. 1, pp. 1–21, 2015.
- [51] “Active learning in machine learning [guide & examples].” (), [Online]. Available: <https://www.v7labs.com/blog/active-learning-guide>. (accessed: 24.10.2022).

- [52] Y. Yang, Z. Ma, F. Nie, X. Chang, and A. G. Hauptmann, “Multi-class active learning by uncertainty sampling with diversity maximization,” *International Journal of Computer Vision*, vol. 113, pp. 113–127, 2015.
- [53] Q. Jin, M. Yuan, Q. Qiao, and Z. Song, “One-shot active learning for image segmentation via contrastive learning and diversity-based sampling,” *Knowledge-Based Systems*, vol. 241, p. 108 278, 2022.
- [54] R. Burbidge, J. J. Rowland, and R. D. King, “Active learning for regression based on query by committee,” *Lecture Notes in Computer Science*, vol. 4881, pp. 209–218, 2007.
- [55] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, B. B. Gupta, X. Chen, and X. Wang, “A survey of deep active learning,” *ACM computing surveys (CSUR)*, vol. 54, no. 9, pp. 1–40, 2021.
- [56] C. Shui, F. Zhou, C. Gagné, and B. Wang, “Deep active learning: Unified and principled method for query and training,” in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 1308–1318.
- [57] J. Folmsbee, X. Liu, M. Brandwein-Weber, and S. Doyle, “Active deep learning: Improved training efficiency of convolutional neural networks for tissue classification in oral cavity cancer,” in *2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018)*, IEEE, 2018, pp. 770–773.
- [58] C. Schröder and A. Niekler, “A survey of active learning for text classification using deep neural networks,” *arXiv preprint arXiv:2008.07267*, 2020.
- [59] P. Liu, H. Zhang, and K. B. Eom, “Active deep learning for classification of hyperspectral images,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 2, pp. 712–724, 2016.
- [60] S. Budd, E. C. Robinson, and B. Kainz, “A survey on active learning and human-in-the-loop deep learning for medical image analysis,” *Medical Image Analysis*, vol. 71, p. 102 062, 2021.
- [61] M. Bloodgood and K. Vijay-Shanker, “A method for stopping active learning based on stabilizing predictions and the need for user-adjustable stopping,” *arXiv preprint arXiv:1409.5165*, 2014.
- [62] B. Curtis, H. Krasner, and N. Iscoe, “A field study of the software design process for large systems,” *Communications of the ACM*, vol. 31, no. 11, pp. 1268–1287, 1988.
- [63] J. Fonseca, G. Douzas, and F. Bacao, “Increasing the effectiveness of active learning: Introducing artificial data generation in active learning for land use/land cover classification,” *Remote Sensing*, vol. 13, no. 13, p. 2619, 2021.
- [64] “Deep active learning with pytorch.” (), [Online]. Available: <https://github.com/cure-lab/deep-active-learning/>. (accessed: 24.10.2022).

- [65] “Deepal: Deep active learning in python.” (), [Online]. Available: <https://github.com/ej0c16/deep-active-learning>. (accessed: 24.10.2022).
- [66] Facebook. “Introduction to the react docs.” (), [Online]. Available: <https://react.dev/>. (accessed: 10.11.2022).
- [67] “Introduction to the angular docs.” (), [Online]. Available: <https://angular.io/docs>. (accessed: 10.11.2022).
- [68] “Introduction to the vue docs.” (), [Online]. Available: <https://vuejs.org/guide/introduction.html>. (accessed: 10.11.2022).
- [69] “Welcome to labelme, the open annotation tool.” (), [Online]. Available: <http://labelme.csail.mit.edu/Release3.0/>. (accessed: 10.11.2022).
- [70] “Coco annotator, made with vue.js.” (), [Online]. Available: <https://madewithvuejs.com/coco-annotator>. (accessed: 10.11.2022).
- [71] “Computer vision annotation tool (cvat).” (), [Online]. Available: <https://github.com/opencv/cvat>. (accessed: 15.12.2022).
- [72] “Django makes it easier to build better web apps more quickly and with less code.” (), [Online]. Available: <https://flask.palletsprojects.com/en/2.2.x/>. (accessed: 25.04.2023).
- [73] “Django makes it easier to build better web apps more quickly and with less code.” (), [Online]. Available: <https://www.djangoproject.com/>. (accessed: 25.04.2023).
- [74] “11 most in-demand programming languages.” (), [Online]. Available: <https://bootcamp.berkeley.edu/blog/most-in-demand-programming-languages/>. (accessed: 25.04.2023).
- [75] “Sqlite.” (), [Online]. Available: <https://sqlite.org/index.html>. (accessed: 25.04.2023).
- [76] “Postgresql: The world’s most advanced open source relational database.” (), [Online]. Available: <https://www.postgresql.org/>. (accessed: 25.04.2023).
- [77] “Nuclio: Serverless platform for automated data science.” (), [Online]. Available: <https://nuclio.io/>. (accessed: 22.03.2023).
- [78] “Develop faster. run anywhere.” (), [Online]. Available: <https://www.docker.com/>. (accessed: 25.04.2023).
- [79] “The free & open source image editor.” (), [Online]. Available: <https://www.gimp.org/>. (accessed: 24.05.2023).

A

APPENDIX

Listing A.1: Nuclio handler in python

```
import json
from common.dal_model_handler import ModelHandler
from nuclio_sdk.context import Context
from nuclio_sdk.event import Event

def handler(context: Context, event: Event):
    model_handler = CrackModelHandler(event)

    result = model_handler(event, context)

    return context.Response(body=json.dumps(result), headers={},
        content_type='application/json', status_code=200)

class CrackModelHandler(ModelHandler):
    ....
```


Listing A.2: Interface for the DAL ModelHandler in python

```
import os
from typing import Any, TypeVar
import numpy as np
from PIL import Image
from nuclio_sdk.context import Context
from nuclio_sdk.event import Event

Model = TypeVar('Model')
DataSet = list[tuple[tuple[Image, ...], dict[str, Image]]]

class ModelHandler:

    model_path: str = "opt/nulcio/"
    ...
    def load(self, path: str):
        """ Load model from given path """
        raise NotImplementedError

    def save(self, path: str, model: Model):
        """ Save model on given path """
        raise NotImplementedError

    def predict(self, event: Event, context: Context) -> list:
        """ Add a prediction method for your model here """
        raise NotImplementedError

    def transfer_learn(self,
        context: Context,
        dataset: DataSet
    ) -> Model:
        """ Transfer learn your model here """
        raise NotImplementedError

    def continiously_learn(self,
        context: Context,
        dataset: DataSet
    ) -> Model:
        """ Continiously learn your model here """
        raise NotImplementedError

    def re_learn(self, context: Context, dataset: DataSet) -> Model:
        """ Re-learn your model here """
        raise NotImplementedError
```


DAL-ART: Deep Active Learning Tool for Multimodal Image Analysis Applied to Crack Detection in Paintings

Sebastiaan Verplancke

Student number: 01604354

Supervisors: Prof. dr. ir. Aleksandra Pizurica, Prof. dr. Maximiliaan Martens
Counsellors: Yoann Arhant, Srdan Lazendic

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2022-2023