COMPUTATION TOOLS 2022

The Thirteenth International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking

ISBN: 978-1-61208-954-6

April 24 - 28, 2022

Barcelona, Spain

COMPUTATION TOOLS 2022 Editors

Claus-Peter Rückemann, Leibniz Universität Hannover / Westfälische Wilhelms-Universität Münster / North-German Supercomputing Alliance (HLRN), Germany

COMPUTATION TOOLS 2022

Forward

The Thirteenth International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking (COMPUTATION TOOLS 2022), held between April 24 - 28, 2022, continued a series of events dealing with logics, algebras, advanced computation techniques, specialized programming languages, and tools for distributed computation. Mainly, the event targeted those aspects supporting context-oriented systems, adaptive systems, service computing, patterns and content-oriented features, temporal and ubiquitous aspects, and many facets of computational benchmarking.

Similar to the previous edition, this event attracted excellent contributions and active participation from all over the world. We were very pleased to receive top quality contributions.

We take here the opportunity to warmly thank all the members of the COMPUTATION TOOLS 2022 technical program committee, as well as the numerous reviewers. The creation of quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and effort to contribute to COMPUTATION TOOLS 2022. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations and sponsors. We also gratefully thank the members of the COMPUTATION TOOLS 2022 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope COMPUTATION TOOLS 2022 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in the area of computational logics, algebras, programming, tools, and benchmarking. We also hope that Barcelona provided a pleasant environment during the conference and everyone saved some time to enjoy the historic charm of the city

COMPUTATION TOOLS 2022 Steering Committee

Cornel Klein, Siemens AG, Germany
Claus-Peter Rückemann, Westfälische Wilhelms-Universität Münster (WWU) / DIMF / Leibniz Universität Hannover, Germany

COMPUTATIONAL TOOLS 2022 Publicity Chairs

Lorena Parra, Universitat Politecnica de Valencia, Spain
Javier Rocher, Universitat Politècnica de València, Spain

COMPUTATION TOOLS 2022

Committee

COMPUTATION TOOLS 2022 Steering Committee

Cornel Klein, Siemens AG, Germany
Claus-Peter Rückemann, Westfälische Wilhelms-Universität Münster (WWU) / DIMF / Leibniz Universität Hannover, Germany

COMPUTATIONAL TOOLS 2022 Publicity Chairs

Lorena Parra, Universitat Politecnica de Valencia, Spain
Javier Rocher, Universitat Politècnica de València, Spain

COMPUTATION TOOLS 2022 Technical Program Committee

Lorenzo Bettini, Università di Firenze, Italy
Ateet Bhalla, Independent Consultant, India
Narhimene Boustia, University Saad Dahlab, Blida 1, Algeria
Azahara Camacho, Opinno, Spain
Angelo Ciaramella, University of Naples Parthenope, Italy
Cornel Klein, Siemens AG, Germany
Emanuele Covino, Universita' di Bari, Italy
Marcos Cramer, TU Dresden, Germany
Santiago Escobar, VRAIN - Universitat Politècnica de València, Spain
Andreas Fischer, Deggendorf Institute of Technology, Germany
Roderick Melnik, Wilfrid Laurier University, Canada
Corrado Mencar, Università degli Studi di Bari Aldo Moro, Italy
Ralph Müller-Pfefferkorn, Technische Universität Dresden, Germany
Keiko Nakata, SAP SE - Potsdam, Germany
Adam Naumowicz, University of Bialystok, Poland
Cecilia E. Nugraheni, Parahyangan Catholic University, Indonesia
Alberto Policriti, University of Udine, Italy
James Tan, Singapore University of Social Sciences, Singapore
Hans Tompits, Technische Universität Wien, Austria
Miroslav Velev, Aries Design Automation, USA
Kristin Yvonne Rozier, Iowa State University, USA

Table of Contents

Efficient Formal Verification with Confidence Intervals

Naif Alasmari
Department of Computer Science
University of York
York, U.K.
email: nnma500@york.ac.uk

Radu Calinescu
Department of Computer Science
University of York
York, U.K.
email: radu.calinescu@york.ac.uk

***Abstract*—Formal verification with confidence intervals is a model checking technique that computes confidence intervals for parametric Markov model properties when observations of the unknown transition probabilities of these models are available. However, the high computational costs of the technique limit its scalability severely. To address this limitation, we introduce efficient formal verification with confidence intervals (eFACT), a model checking tool that enables the efficient analysis of parametric discrete-time Markov chains. eFACT supports the verification of reliability, performance, and other non-functional requirements for larger systems than currently possible. To that end, eFACT integrates recent advances in parametric model checking into a previous tool for formal verification with confidence intervals, and employs an efficient binary search technique to further speed up the determination of the highest confidence level at which a non-functional requirement can be deemed violated or satisfied.**

***Keywords*—confidence intervals; formal verification; non-functional software requirements; probabilistic model checking.**

I. Introduction

Over the years, quantitative verification has been a powerful means for analysing the performance, reliability, and other non-functional properties of systems. However, the analysed system should be modelled carefully and accurately as a Markov model in order to obtain a precise verification result. Building a Markov model for the system is a time-consuming task because it requires determining the system's states and the transitions between them, as well as their probabilities. Establishing the precise probabilities of transitions is challenging [1] since the probabilities can only be estimated, with error margins, from run-time observations of the system, system logs, or based on input obtained from domain experts. Therefore, the error values of probabilities estimation could be accumulated by quantitative verification and can produce inaccurate outcomes due to the non-linearity of Markovian models. The formal verification with confidence intervals (FACT) [1] resolves this limitation by providing an interval for the verification result rather than a single value [2].

FACT is a probabilistic model checker that calculates confidence intervals for properties of parametric Markov chains that have observations for unknown transition probabilities. The current FACT version invokes PRISM [3] to get the algebraic expression for the targeted property of a parametric Discrete-Time Markov Chain (pDTMC) model. However, when the algebraic expression is too large to analyse (i.e., it exceeds the memory or computational resources available), or the behaviour of the parametric model is complex (e.g., has continual change), the capability of FACT becomes limited. Thus, FACT does not scale well to large pDTMCs.

To extend the capability of FACT, our paper introduces $e$fficient Formal verific$A$tion with Confidence in$T$ervals (eFACT), a new model checker that is able to calculate such confidence intervals for larger pDTMC models. eFACT exploits efficient parametric model checking (ePMC), which uses domain-specific modelling patterns [4] in order to produce sets of closed-form subexpressions of the analysed properties, and uses these subexpressions as terms in the main formula for the analysis of the whole pDTMC model. The ePMC derives an abstraction model from the original pDTMC. The abstraction model consists of fragments, and each fragment represents a single state in the abstraction model and a subset of states in the original model. The main formula is the abstraction model's algebraic expression, and the component formula is a formula related to the fragment. In this way, eFACT computes the confidence intervals for each closed-form expression, and then uses the obtained results to calculate the confidence interval for the main formula. Furthermore, eFACT exploits a binary search technique to enable engineers to obtain the highest confidence level at which a non-functional requirement of a system can be confirmed as violated or satisfied—an important feature unavailable in the FACT tool. Furthermore, since we are concerned about the safety and business-critical systems, it is vital to consider a high confidence level before deploying the system.

The paper is organised as follows. Section II explains how eFACT computes the confidence intervals for the properties of large pDTMC models. Next, Section III demonstrates the use of binary search to efficiently find the required confidence level. Section IV describes the case studies used to evaluate eFACT. Next, Section V discusses the experimental results, and Section VI compares our solution to related work. Finally, Section VII briefly summarises this work, and highlights directions for future work.

II. Confidence intervals for large pDTMCs

For a given pDTMC model and a property of this model encoded in Probabilistic Computation Tree Logic (PCTL), FACT obtains an algebraic expression from PRISM to compute confidence intervals. However, FACT cannot produce confidence intervals when the model is large (as explained in the previous section). eFACT aims to analyse large pDTMC models with at least one unknown transition probability, provided that observations of the unknown transition exist. To achieve this purpose, we exploit a recent advance in probabilistic model checking and the model checker ePMC [4][5] that produces closed-form expressions (i.e., component formula) for the property being analysed, and then combines them into one main formula. eFACT analyses each expression separately to produce its confidence intervals for the provided confidence levels. The confidence intervals of the expression then substitute into the main formula. Therefore, the outcomes of all expressions contribute to calculating the confidence intervals for the analysed property of a given large pDTMC.

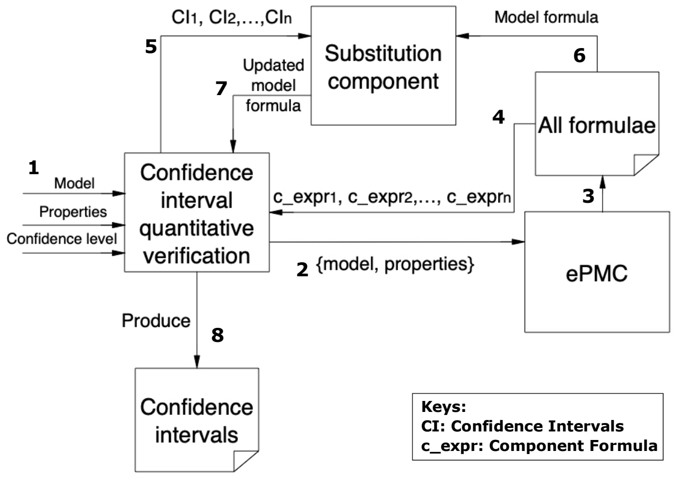

Figure 1. eFACT structure.

At a given confidence level α, computing confidence intervals for a large pDTMC model consists of three main steps: confidence interval quantitative verification, ePMC, and substitution. The confidence interval quantitative verification is used to receive the inputs and compute the confidence intervals for closed-form expression. ePMC is employed to produce the closed-form expressions and the main formula of the property. The substitution component is used to substitute the outcome of all closed-form expressions in the main formula of the property. The substitution component handles two equations representing the main formula: one to substitute all lower intervals of each expression into the main formula and the other is used to substitute the upper level of expressions in the main formula.

Figure 1 illustrates the steps followed to compute the confidence intervals for a large pDTMC in detail. First, the confidence interval quantitative verification will receive the pDTMC model, property, and a range of confidence intervals as inputs. The model and property are then sent to ePMC to obtain all possible closed-form expressions formulae (Steps 2 and 3 in Figure 1). There are two kinds of produced formulae: the component formula (closed-form expression) and the model formula used to analyse the system model. The component formula could be a part of the model formula in the latter formula. In general, the formula represents an algebraic expression related to the analysed property of the model. The component formulae (denoted as c_expr_1,c_expr_2,...,c_expr_n in the figure) are then sent for confidence interval quantitative verification to compute their confidence intervals sequentially (Step 4). The results of analysing the component formulae are sent to the substitution component to substitute their results into the model formula (as shown in Steps 5 and 6). Finally, the model formula is sent to the confidence interval quantitative verification unit to calculate the final confidence intervals that will appear to the end-user.

III. Finding the highest confidence level

When engineers use eFACT to compute confidence intervals for a PCTL-encoded pDTMC property, they are often interested in comparing these intervals with a bound that the property must satisfy per the analysed system's non-functional requirement. Furthermore, they are particularly interested in finding the *highest confidence level* α_{MAX} at which the requirement can be shown as violated or satisfied, given the available set of observations of the unknown pDTMC transitions. For confidence levels $\alpha > \alpha_{MAX}$, the observations available are insufficient for deciding whether the requirement is satisfied. Finding the value of α_{MAX} (or a close approximation of it) enables important decision-making. For instance, if a requirement can be shown to be satisfied at the highest confidence level $\alpha_{MAX} = 0.99$, the system can be confidently deployed (based on the requirement being met). In contrast, if a requirement can only be shown as satisfied at the highest confidence level $\alpha_{MAX} = 0.75$, the decision of whether to deploy the system cannot be made. Further observations should be obtained, for example, by testing the relevant system components.

eFACT can compute a potentially very large number of confidence intervals at different confidence levels α to find a close approximation of α_{MAX}. eFACT is highly inefficient in achieving this, given the overheads of formal verification with confidence intervals. Therefore, we developed an efficient method (implemented in eFACT) for computing this close approximation. This method employs a binary search to efficiently approximate the highest confidence level α_{MAX}. Therefore, instead of slowly performing verification for each confidence level to determine where the requirement is satisfied or violated, the binary search technique will speed up the process of achieving this. When the user inserts the model (non-functional requirement and range of confidence levels), eFACT starts its work by verifying the first inserted confidence level and computing its confidence intervals. Following this, it moves to the last inserted confidence level to calculate its confidence intervals. Now, there are two confidence levels with their intervals, enabling eFACT to check whether the analysed property requirement is located inside those intervals. If the requirement is located inside all intervals, the process will terminate with a message stating that the requirement is undecidable for the given range of confidence levels. Otherwise, eFACT moves to the middle confidence level (e.g., if the range of confidence levels is between 89 and 99, then the middle level is 94) and computes the confidence intervals. eFACT then checks the requirement's position over the current

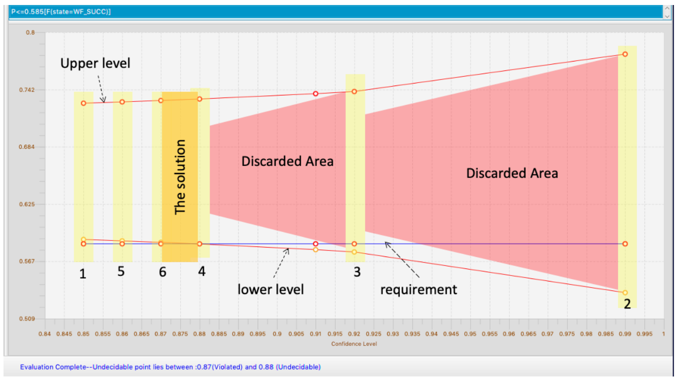

Figure 2. An example of using binary search in eFACT.

confidence intervals and compares it with the obtained ones over the confidence intervals from the first and last levels.

The confidence levels that lie between the middle confidence level and the other confidence level, in which the requirement's position matches its place in the middle level, will be discarded. Again, eFACT moves to the middle of the remaining confidence levels and repeats the same procedure until it determines the highest confidence level α_{MAX}.

Figure 2 shows the verification result, where eFACT is looking for the confidence level at which the requirement is violated or satisfied. The test was conducted between confidence levels 0.85 and 0.99, where the increment step was 0.01. Instead of completing 15 verification tests to determine the required confidence level, we performed six verification tests until the required result was found. The discarded area (red area) has a list of confidence levels with intervals containing the requirement; therefore, performing additional tests in this area is useless. The solution area (green area) is where the requirement's position moves from outside the confidence intervals to be inside the next intervals.

IV. Case Studies

A. *Service-based systems*

Service-based systems (SBSs) are applications that provide services that are dependent on or connected to each other [6]. SBSs comprise internal system components and possible independent third-party components implemented as services. There are different ways in which services can conduct operations similar to those of SBSs but with different probabilities in their execution time (t_1,...,t_n), costs (c_1,...,c_n) and successes (p_1,...,p_n). The patterns are adopted from [4]: sequential execution (SEQ), sequential execution with a retry (SEQ-R), sequential execution with retry1 (SEQ-R1), probabilistic execution (PROB), probabilistic execution with a retry (PROB-R), probabilistic execution with retry1 (PROB-R1), parallel execution (PAR), and parallel execution with a retry (PAR-R). They are used to implement the SBS operations with n services equivalent to those operations described below:

1) SEQ $(p_1, t_1, c_1, ..., p_n, t_n, c_n)$: There are n services invoked in order, terminated after the last service or upon a first successful request.
2) SEQ-R $(p_1, t_1, c_1, ..., p_n, t_n, c_n, r)$: This is similar to SEQ. However, if all service invocations fail, the operation is re-executed from the first service with probability r or it fails with probability 1-r.
3) SEQ-R1 $(p_1, t_1, c_1, r_1, ..., p_n, t_n, c_n, r_n)$: This is similar to SEQ. However, service i will be re-invoked with probability r_i if the invocation of this service fails.
4) PAR$(p_1, t_1, c_1, ..., p_n, t_n, c_n)$: There are n services invoked simultaneously. The operation will use the output of the first successful invocation.
5) PAR-R $(p_1, t_1, c_1, ..., p_n, t_n, c_n, r)$: This is similar to PAR. However, if all service invocations fail, the operation is re-executed with probability r or it fails with probability 1-r.
6) PROB $(x_1, p_1, t_1, c_1, ..., x_n, p_n, t_n, c_n)$: There is a single service to request. The probability that indicates the service i is x_i, where $\Sigma_{i=1}^{n}$ x_i =1.
7) PROB-R$(x_1, p_1, t_1, c_1, ..., x_n, p_n, t_n, c_n, r)$: This is similar to PROB. However, if the service invocations fail, the operation is re-executed with probability r or it fails with probability 1-r.
8) PROB-R1$(x_1, p_1, t_1, c_1, r_1, ..., x_n, p_n, t_n, c_n, r_n)$: This is similar to PROB. However, if the service invocations fail, the service is re-invoked with the probability of r or it fails with probability 1-r.
9) Combination: This is a combination of the above patterns.

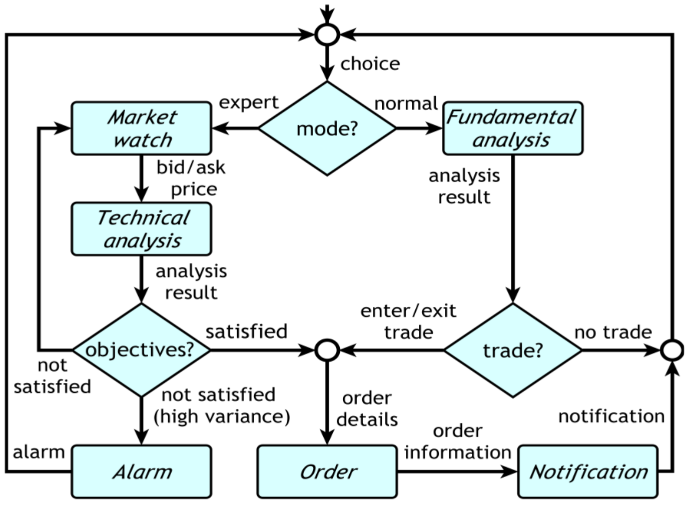

Figure 3. Foreign Exchange System Workflow, from [7].

To evaluate eFACT, a foreign exchange system (FX system) from the SBS area that aims to assists the trader is adopted from [7]. As shown in the Figure 3, the FX system offers the trader two operational modes: expert or normal. The expert mode executes the trade automatically when the transaction meets the customer's objectives. It begins with the market-watch component to obtain the current price of the chosen currency, then it uses the technical-analysis component to assess the market and estimate the price movement. The analysed outputs could be one of three options:

1) The transaction can be performed because the objectives that the traders set up are satisfied;
2) The market watch component is re-invoked since the objectives were not met; and
3) The objectives are incorrect, and the Alarm unit will be triggered to warn the trader.

Conversely, the FX system utilises the fundamental-analysis component in its normal mode to determine whether to conduct a transaction, retry the analysis or end the session.

eFACT aims to analyse the following properties of the pDTMC model of the FX system with multiple services (from 1 to 6), and under different patterns:

1) $P1$: The possibility of completing a transaction successfully, written in the PCTL format as $P=?[F(state = WF\_SUCC)]$;
2) $P2$: The estimated time to execute the transaction, written in the PCTL format as $R\{"time"\}=?[F((state = WF\_SUCC)|(state = WF\_FAIL))]$; and
3) $P3$: The estimated cost of running the transaction successfully, written in the PCTL format as $R\{"cost"\} = ?[F((state = WF\_SUCC)|(state = WF\_FAIL))]$.

B. Three-tier software architectures

The three-tier server [8], as shown in Figure 4, provides three services: web, database and application services. The services are hosted on four different physical servers (A, B, C and D) and operate on different virtual machines (VMs). The system can be scaled-up to include more servers, VMs and service instances. This case study presents the following three patterns:

- Basic (B): Several tier instances are running on a server. If the server crashes, the running tier instances are lost.
- Virtualised (V): There are a number of tier instances, and each one is running on its own virtual machine on a server.
- Virtulised-M (VM): This is similar to the virtualised pattern. However, when the server crashes, a monitoring component can detect a crash before it occurs. Therefore, the virtual machine can be migrated to other running servers. For example, consider a server with several components, including processors, disks, and memory chips, that are now working but are prone to crash over time. If a substantial quantity of components crashes, the detection monitoring component discovers the crashes and begins migrating all the VMs in this server to another server.

If the engineers intend to evaluate the probability of deploying options for the three-tier software on different servers, they could evaluate the following properties:

1) $P1$: This measures the likelihood of the system failing within a determined time due to all tier instances failing. It can be written in PCTL as $P=?[F\ done\ \&\ fail]$; and
2) $P2$: This assesses the possibility of a single failure point during the analysis. The PCTL encoded for this property is $P=?[F\ done\ \&\ spf]$.

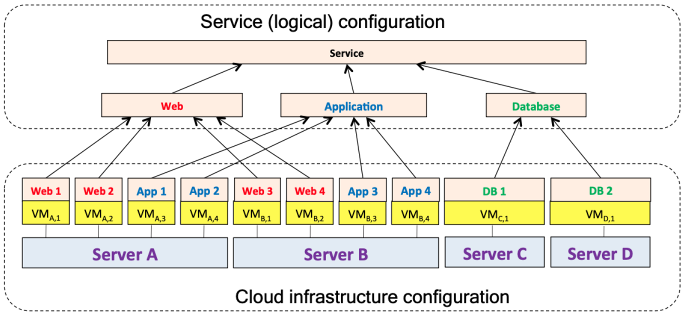

Figure 4. Three-tier architecture deployed on a Cloud, from [8].

V. Evaluation

We performed a set of experiments to compare eFACT and FACT using two different case studies from different areas. Those case studies are described in Section IV. All experiments were conducted on an OSX 10.14.6 MacBook Pro laptop with an 8 GB 1600 MHz DDR3 RAM and CPU 2.5 GHz Intel Core i5 processor.

A. Experimental environment

eFACT was developed using JAVA and required installing the following tools and applications:

1) PRISM/Storm [9] are model checker tools used to analyse properties and produce algebraic expressions. eFACT tested using PRISM v4.4 and Storm v1.5.1.
2) MATLAB is used for computing confidence intervals, and the used version is R2019a.
3) YALIMP [10][11] is a MATLAB-based modelling language that was developed by Johan Lofberg and contains several free and commercial solvers. It is used to model and formulate both convex and non-convex optimisation problems. It is invoked in the background by eFACT/FACT to solve the convex optimisation problem. Our work was applied using version 20210331.
4) Gurobi [12] is an optimisation solver that YALIMP can invoke to solve the optimisation problem.
5) ePMC repository defines the model's patterns and contains the expressions related to the properties of the model.

B. Results

For the first case study that SBS explained in Section IV-A, we carried out several experiments to analyse three properties ($P1$, $P2$, $P3$), produce the confidence intervals from α=0.90 to $\alpha = 0.99$, and record the execution time in seconds. The analysis was carried out under different patterns and with different services. Table I summarises the results and shows the execution time (in seconds) taken to analyse each property using eFACT and FACT. The table contains the following symbols:

- (T) denotes the time out, which means that the execution time exceeds the predefined time of 1800 seconds without completing the analysis.

TABLE I. THE RESULTS OF FX SYSTEM, (THE EXECUTION TIME IS IN SECONDS).

Pattern	Services	eFACT			FACT		
		P1	P2	P3	P1	P2	P3
SEQ	1	141.405	175.357	188.243	98.817	105.328	100.098
	2	147.276	194.089	194.503	T	T	T
	3	188.993	225.028	227.659	-	-	-
	4	286.416	354.003	353.514	-	-	-
SEQ-R	2	197.213	284.967	282.978	T	T	T
	3	253.189	348.103	355.679	-	-	-
	4	1507.257	1314.996	1285.241	-	-	-
SEQ-R1	2	187.587	270.305	272.994	T*	T	T
	3	222.844	322.971	322.634	-	-	-
	4	423.71	501.281	510.492	-	-	-
PAR	2	141.299	189.637	185.0	T	T	T
	3	186.318	269.309	221.306	-	-	-
	4	290.874	384.679	296.634	-	-	-
PAR-R	2	199.079	299.229	280.785	T	T	T
	3	248.545	380.596	337.698	-	-	-
	4	1487.141	1787.865	1352.024	-	-	-
PROB	2	138.287	183.473	182.499	182.595	1130.434	1025.849
	3	143.724	187.631	192.325	-	-	-
	4	148.537	197.401	200.723	-	-	-
PROB-R	2	197.09	262.869	263.637	T	T	T
	3	220.979	294.346	295.803	-	-	-
	4	238.253	339.933	334.826	-	-	-
PROB-R1	2	186.974	262.863	260.842	T	T	T
	3	216.312	293.574	294.891	-	-	-
	4	230.583	337.127	345.044	-	-	-
Combination	Min	155.351	199.017	197.774	T	T	T
	Max	292.297	290.975	286.386	-	-	-
	Mean	177.582	231.562	222.059	-	-	-
	Stdev	30.139	24.23	24.952	-	-	-

- (T*) means the tool is failed to produce an algebraic expression for the property being analysed during the predefined time.
- (-) indicates that we skipped this experiment since the previous model is smaller than the current one, and it failed to compute the confidence intervals in the determined time frame.

As shown in Table I, the execution time recorded for eFACT is better than FACT's execution time, except for the first row, where the model has a single service (SEQ pattern with one service). Further, to analyse the model in the first row, eFACT requires more time to compute confidence intervals for the component expressions (more than one expression) before substituting their results into the model formula. Moreover, we notice that the difference is not so significant. The table shows that eFACT takes less time than FACT for the analysis of other patterns and services.

For the second case study mentioned in Section IV-B, several experiments were performed to evaluate their two properties ($P1, P2$) and calculate the confidence intervals from α=0.90 to $\alpha = 0.99$. Table II illustrates the results for four models of four servers with different patterns. FACT takes less execution time to analyse the model of deployment D1, which is found in the first row. The model is simple and produces a small expression that FACT can handle. In deployment D2, the model has some complexity (loop), and the expressions for both properties are too large. Therefore, FACT failed to analyse them before the time was out. eFACT can handle this model since it deals with small component expressions to first analyse them and then exploit their outcomes in analysing the main formula. The third row is for deployment D3, which is a loop-free model. We note that FACT can analyse this model but is higher than eFACT analysis time. The last row shows the superiority of eFACT, where FACT cannot analyse the properties of this model since the algebraic expression failed to be produced in 1800 seconds.

VI. RELATED WORK

Software engineers can exploit the probabilistic model checking to analyse and assess the reliability, correctness, potential performance and other key attributes of systems with probabilistic behaviour. However, the model can be affected by the unquantified estimation errors of transition probabilities, leading to uncertainty. Specifically, the probabilities of transitions from one state to another in DTMC could be unrealistic since statistical experiments calculate them. Multiple studies have been conducted to diminish the uncertainty that arises in DTMC models. The studies accomplished by [13][14] have sought to capture this kind of problem. Kozine and Utkin [13] supposed that the probability value should be included between two bounds (upper and lower) instead of being a specific value. They exploited the theory of interval-valued coherent prevision to generalise discrete Markov chains and introduce interval-valued, discrete-time Markov chains (IDTMCs). Škulj [14] attempted to refine the IDTMCs and develop consecutive steps to make the IDTMCs suitable for the models with generic convex sets of probabilities. The work in [15] applied upper and lower bounds on the complexity

TABLE II. THE RESULTS OF THE MULTI-TIER SYSTEM.

Deployment	Number of instances	Server type				eFACT		FACT	
		Server A	Server B	Server C	Server D	P1	P2	P1	P
D1	6	V	V	B	B	203.266	214.387	94.088	86.317
D2	6	VM	VM	B	B	331.342	359.419	T	T
D3	10	V	V	V	V	337.281	365.966	687.554	730.999
D4	10	VM	VM	VM	VM	824.153	873.638	T*	T*

of calculating values for undetermined probabilities in the model checking of an interval Markov chains that increased the likelihood of satisfying ω-regular specification. FACT [1] computes confidence intervals for analysing the properties of Markov chains instead of giving a single value to resolve uncertainty. However, FACT fails to compute the confidence intervals when the produced algebraic expression is too large or not produced. eFACT employs the ePMC approach to compute the confidence intervals when FACT cannot.

VII. CONCLUSION AND FUTURE WORK

In this paper, we introduced eFACT, a new model checker with confidence intervals that significantly improves the scalability of existing solutions for the analysis of pDTMCs. In addition, eFACT can benefit engineers who want to establish the analysed pDTMC model's confidence level in the satisfaction or violation of a given non-functional requirement. Our experimental results show that eFACT has better execution times than the model checker FACT that it builds on, outperforming FACT in most cases. One of our work's limitation is that the model requires a repository of components' equations and an abstract model that require a domain expert. However, this limitation can be resolved using a recently introduced generic method for efficient parametric model checking [16]. Integrating this new method into eFACT and further evaluating the scalability of the tool represent areas of future work for our project. As another future work direction, the efficiency of eFACT can be further increased by analysing the component expressions generated by ePMC in parallel.

REFERENCES

[1] R. Calinescu, K. Johnson, and C. Paterson, "FACT: A probabilistic model checker for formal verification with confidence intervals," in International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2016, pp. 540–546.

[2] N. Alasmari, R. Calinescu, C. Paterson, and R. Mirandola, "Quantitative verification with adaptive uncertainty reduction," Journal of Systems and Software, 2022, in press. Pre-print available at https://www.sciencedirect.com/science/article/abs/pii/S016412122200036X.

[3] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic symbolic model checker," in International Conference on Modelling Techniques and Tools for Computer Performance Evaluation. Springer, 2002, pp. 200–204.

[4] R. Calinescu, C. Paterson, and K. Johnson, "Efficient parametric model checking using domain knowledge," IEEE Transactions on Software Engineering, vol. 47, no. 6, pp. 1114–1133, 2019.

[5] R. Calinescu, K. Johnson, and C. Paterson, "Efficient parametric model checking using domain-specific modelling patterns," in 2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER). IEEE, 2018, pp. 61–64.

[6] M. Deubler, J. Grünbauer, J. Jürjens, and G. Wimmel, "Sound development of secure service-based systems," in Proceedings of the 2nd International Conference on Service Oriented Computing, 2004, pp. 115–124.

[7] S. Gerasimou, G. Tamburrelli, and R. Calinescu, "Search-based synthesis of probabilistic models for quality-of-service software engineering," in 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2015, pp. 319–330.

[8] R. Calinescu, S. Kikuchi, and K. Johnson, "Compositional reverification of probabilistic safety properties for large-scale complex IT systems," in Monterey Workshop. Springer, 2012, pp. 303–329.

[9] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A storm is coming: A modern probabilistic model checker," in International Conference on Computer Aided Verification. Springer, 2017, pp. 592–600.

[10] J. Lofberg, "YALMIP : a toolbox for modeling and optimization in MATLAB," in 2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508), 2004, pp. 284–289.

[11] J. Löfberg, "Modeling and solving uncertain optimization problems in yalmip," IFAC Proceedings Volumes, vol. 41, no. 2, pp. 1337–1341, 2008.

[12] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2021. [Online]. Available: https://www.gurobi.com

[13] I. O. Kozine and L. V. Utkin, "Interval-valued finite Markov chains," Reliable computing, vol. 8, no. 2, pp. 97–113, 2002.

[14] D. Škulj, "Discrete time Markov chains with interval probabilities," International Journal of Approximate Reasoning, vol. 50, no. 8, pp. 1314–1329, 2009.

[15] M. Benedikt, R. Lenhardt, and J. Worrell, "LTL model checking of interval Markov chains," in International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2013, pp. 32–46.

[16] X. Fang, R. Calinescu, S. Gerasimou, and F. Alhwikem, "Fast parametric model checking through model fragmentation," in 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021, pp. 835–846.

High-pass Filters Preprocessing in Image Tracing with Convolutional Autoencoders

Andreas Fischer and Zineddine Bettouche
Deggendorf Institute of Technology
Dieter-Görlitz-Platz 1
94469 Deggendorf
E-Mail: andreas.fischer@th-deg.de, zineddine.bettouche@th-deg.de

***Abstract*—Image tracing describes the task of converting a raster image into a vector format. This paper investigates different processing pipelines that can extract an abstract representation of an image by means of high-pass filtering, autoencoding, and vectorization. Results indicate that reconstructing an image using Autoencoders, then filtering it with high-pass filters, and finally vectorizing it, can represent the image more abstractly while improving the efficiency of the vectorization process.**

***Index Terms*—image quality, vector graphics, neural networks, autoencoders, high-pass filters, vectorization, complexity theory, information technology.**

I. Introduction

Graphical information can be represented digitally in two ways: raster-oriented or vector-oriented. The visual information in raster images is encoded as a 2D pixel array (or a bitmap). This is the case in the well-known Portable Network Graphics (PNG) or JPEG File Interchange Format (JFIF, commonly known just as JPEG). However, vector graphics represent data in a set of mathematical primitives such as lines and circles. A widely used format for vector graphics is Scalable Vector Graphics (SVG).

There are many use-cases for conversion between the two ways of representation. Vector images have to be rasterized in order to display on a raster-oriented monitor, and raster images have to be vectorized, if one wants to obtain a scale-free representation of the image. However, vectorization is not as accurate as rasterization. The main obstacle of converting a raster image into a vector graphic is the identification of mathematical primitives in a way that is appropriately fitting. On the one hand, a literal representation produces too much noise and fails to capture the relationships between image areas. On the other hand, the mapped image should not vary much from its original version.

This paper investigates different processing pipelines that can extract an abstract representation of an image by means of high-pass filtering, autoencoding, and vectorization. It extends previous work by Fischer and Amesberger [1], investigating the interplay between autoencoders and high-pass filters in the vectorization process. The cat dataset by Zhang et al. [2] is used to demonstrate the applicability of our approach.

The remainder of this paper is structured as follows: In Section II a brief introduction is given on image tracing, autoencoders and high-pass filters. Section IV introduces the methodology of this paper. This includes the evaluation methods used and the reason why they have been chosen. Section V presents the experiments and their results. This is the part that attempts to eliminate inefficient processing algorithms, so that only a few pipelines that score closely are put forward to further evaluation. Section VI includes the evaluation of the different processing pipelines built, and closes with a summarizing interpretation. Section III discusses related work. Finally, Section VII concludes the paper and discusses future work.

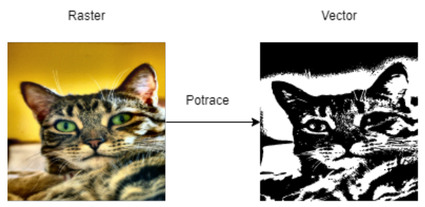

Figure 1. Potrace vectorization

II. Background

In this section, the three main techniques for image processing used in this paper are discussed. Image tracing is used to produce vector graphics from a raster image. Autoencoders and high-pass filters are used as pre-processing steps to reduce the complexity in the image with the goal of achieving an abstract representation of the image in its vector format.

A. Image Tracing

Image tracing is the process of vectorization raster images. It works by using edge detection mechanisms to identify areas in a raster image to be represented as mathematical objects such as polygons. The vectorization program used in this work is Potrace by Peter Selinger [3]. It traces the images by first converting them into black-and-white and then extracting Bézier-bounded polygons (cf. Figure 1).

B. Autoencoder

Autoencoders are artificial neural networks that can be trained to reconstruct the input by passing it through an

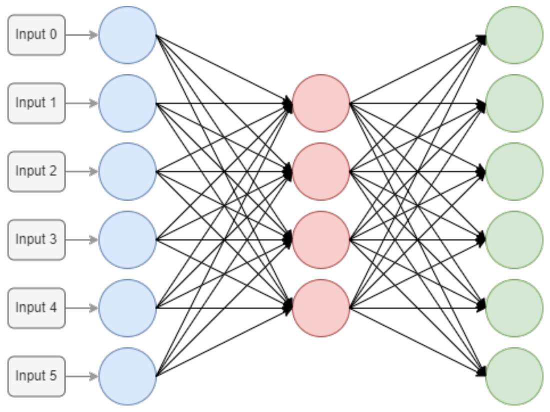

Figure 2. Example structure of an autoencoding network

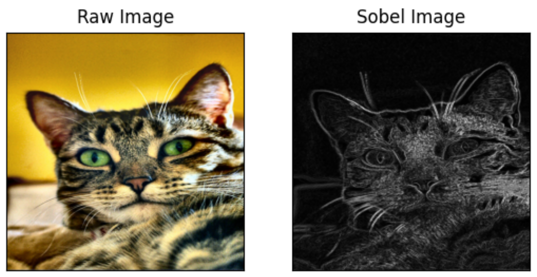

Figure 3. Applying Sobel derivatives on a random image

information bottleneck. The network learns how to preserve the input by an efficient reconstruction. This preservation of information can reduce the input complexity.

A basic autoencoder architecture is shown in Figure 2. The input layer is the layer that is the farthest on the left. The number of neurons becomes smaller the closer the layer is to the center, and this builds the information bottleneck. When the information reaches the center layer, it is at its highest density. By mirroring the architecture to the right, the encoded data from the center layer gets decoded again to its original size. The training process encourages the output layer to match the information that was passed to the input layer.

C. High-pass Filters

A high-pass filter can be used to make an image appear sharper. These filters (e.g., Sobel [4] and Canny [5]) emphasize fine details in the image. High-pass filtering works with the change in intensity. If one pixel is brighter than its immediate neighbors, it gets boosted. Figure 3 shows the result of applying a high-pass filter (Sobel) on a random image.

III. Related Work

In a paper done in MIT, Solomon and Bessmeltsev [6] explored the use of frame fields. The general idea of their method is to find a smooth frame field on the image plane, where at least one direction is aligned with nearby contours of the drawing. Around X- or T-shaped junctions, the two directions of the field will be aligned with the two intersecting contours. Then, the topology of the drawing is extracted by tracing the frame field and grouping traced curves into strokes. Finally, they created with the extracted topology a vectorization aligned with the frame field.

Lacroix [7] analyzed some problems of R2V conversion, and a strategy has been proposed involving a preprocessing stage generating a mask, from which edges are removed and lines are kept. A clustering is then performed while considering only the pixels of the mask. A new algorithm, the medianshift, has been proposed in this context. Then comes the labeling process which should also take the pixel type into account. The last step involves a regularization procedure. The importance of the pre-processing ignoring edge pixels while keeping lines has been shown on some examples. Tests also showed the superiority of the median-shift over the mean-shift, and over the clustering method used by Vector-Magic. This paper also showed that a better line vectorization can be obtained from enabling the extraction of dark lines, which can support the use of high-pass filters as a preprocessing stage to put further emphasis on those dark lines.

Xie et al. [8] designed a novel approach, which achieves a performance comparable to traditional linear sparse coding algorithm on the simple task of denoising additive white Gaussian noise. They use autoencoders to reduce image noise in the area of repairing damaged images.

Gong et al. [9] presented an algorithm that successfully completes the automatic extraction and vectorization of the road network. The main obstacles in road extraction in remote sensing images are: first, different scales and strong connectivity; second, complex backgrounds and occlusions; and third, high resolution and a small proportion of roads in the image. The process of road vectorization in this paper is mainly divided into road network extraction and vectorization preservation. This work also shows the advantages of using dense dilation convolution, which points to the possibility of using autoencoding models for vectorization preservation.

Fischer and Amesberger [1] showed that preprocessing the raster image with an autoencoder neural network, can reduce complexity by over 70% while keeping reasonable image quality. They proved that autoencoders perform significantly better compared to PCA in this task. We base our our work on this previous work, having a closer look at the effect of high-pass filters on autoencoding in an image vectorization pipeline.

IV. Methodology

This section describes the methodology of this paper. The programming implementation is first introduced along with the autoencoder structure. Then, the common grounds of processing pipelines are demonstrated. In addition, the evaluation methods are presented with the rationale behind their selection.

A. Autoencoder Structure & Software Implementation

The test/evaluation framework was implemented in Python. The autoencoder was implemented with TensorFlow [10] and Keras [11]. The convolutional neural network was built with convolution and pooling layers in three steps to a 32×32

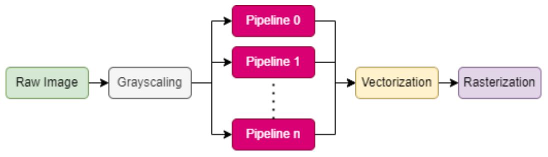

Figure 4. General processing approach

bottleneck. The decoder mirrors this structure with three steps of transposed convolutional layers and batch normalization layers. The autoencoder input is set to a 255x255 image (gray-scaled). The high-pass filters used in this paper are the standard implementations in OpenCV [12].

B. General Approach of Processing

Regardless of the path an image takes in any pipeline that will be built, the first processing stage is always going to be converting the image into gray-scale. The focus of this work is around single-channel images; however, it can be extended in the future for multi-channel (RGB) processing. Therefore, when a pipeline is demonstrated visually, the initial version of the image displayed is going to be gray-scale, but this is implying that the raw RGB images were all grayscaled, which will be a common branch for all the pipelines built in this work.

After an image is grayscaled, it will be put through a certain cascade of processing stages. In this paper, the concerned stages are: High-pass Filtering, Autoencoding, and Vectorization. The experiments of this work are going to tune the different parameters that these stages can take. More importantly, the outputs of all pipelines possible are going to be in a vector format; due to the fact that we are attempting to enhance the vectorization process, while aiming for an abstract representation of the image. Therefore, a rasterization stage is going to always be placed at the end of every pipeline. Converting images back into their raster format is mandatory to perform a comparison between the grayscaled image that was initially fed to a pipeline, and its resulting vector format. Hence, we rasterize the vector output to be able to evaluate the efficiency of the pipeline. A general processing approach for the different pipelines is shown in Figure 4.

C. Evaluation Methods

The case at hand deals with both vector and raster images. Therefore, for a comparison to take place, a comparison method for each format needs to be selected.

- **Vector:** Various methods can be used to measure the level of the complexity in a vector image. One can be the size of the file, which can be used to infer the length of the entire path entries in the file. Furthermore, investigating the reduction of complexity can be done through analyzing the longest path tags. The number of path tags can be taken as a characteristic value of the complexity. In this paper, it is assumed that the number of SVG path entries is directly related to the complexity.
- **Raster:** There are mainly two common ways of comparing raster images. The first one is comparing images based on the Mean Squared Error (MSE) [13]. The MSE value denotes the average difference of the pixels all over the image. A higher MSE value designates a greater difference between the original image and processed image. Nonetheless, it is indispensable to be extremely careful with the edges. A major problem with the MSE is that large differences between the pixel values do not necessarily mean large differences in content in the images. The Structural Similarity Index (SSIM) [14] is used to account for changes in the structure of the image rather than just the perceived change in pixel values across the entire image. The implementation of the SSIM used is contained in the Python library Scikit-image [15]. The SSIM method is significantly more complex and computationally intensive than the MSE method, but essentially the SSIM tries to model the perceived change in the structural information of the image, while the MSE actually estimates the perceived errors.

In the experiments conducted for this paper, the results of MSE and SSIM drive to the same conclusion. Therefore, in order to avoid redundancy, only the SSIM graphs are displayed in this paper.

V. EXPERIMENTS

Experiments in this paper are essential to tune the different parameters a pipeline can have. In this section, some of the important experiments undergone are going to be presented.

Nevertheless, other unmentioned experiments also helped in cutting down the number of possible pipelines for evaluation. For instance, at the beginning of the work, three high-pass filters were selected for their commonality: Gaussian, Sobel, and Canny. However, the Gaussian filter introduced noise onto the images, which was not negligible. An experiment was done that attempted to reduce such noise by applying the Grain-extract or the Difference filters, but the results were not acceptable. Therefore, the Gaussian filter was removed from the set of filters. Another experiment was conducted to compare the scores of Sobel and Canny filtering. Both SSIM and MSE indicated that the two filters were so close to each other that there was not a decisive choice between them.

The two experiments mentioned below deal with the effect of features' color on the autoencoding and vectorization stages. The need for experimentation concerning such effect has risen when it was found that the conventional implementation of filters in the programming libraries resulted in images having a dark background with light features (lines and shapes), which was perceived as counter-intuitive.

The filters that were succeeded with the word **-direct** are the ones that follow the conventional implementation (dark background with white lines). On the opposite side, filters with **-inverse** as suffix refer to the ones that are constituted of white background with black lines.

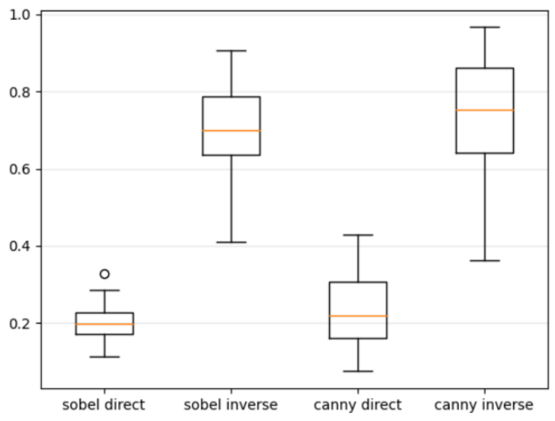

Figure 5. Similarity of autoencoded images in relation to the color of their features

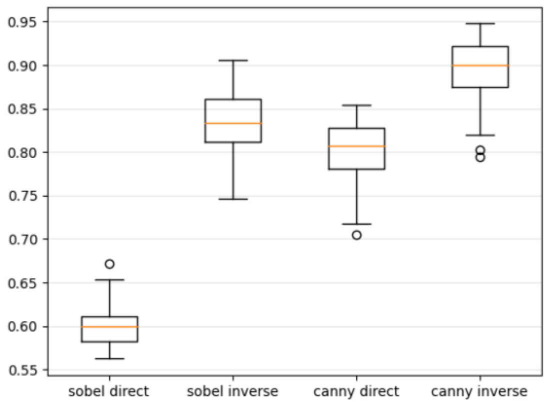

Figure 6. Similarity of vectorized images in relation to the color of their features

A. Features' Color Effect on Autoencoding

This experiment was done to obtain the difference between training an autoencoder with images whose lines are drawn in black on white background, and training it with the same images but inverted. Therefore, four models of autoencoders were trained with 5000 epochs. 50 images were reconstructed, in order to make the measurement more generalized.

From the plotted results in Figure 5 we conclude that autoencoders respond better when the training images have darker features.

B. Features' Color Effect on Vectorization

This experiment aims to display the effect of high-pass filters on reconstructed-images vectorization. We took 50 random images, reconstructed them and filtered each image with Canny and Sobel filters (direct and inverse: 4 versions per image). Finally, all of the images were vectorized, and then compared (after rasterization) with their versions pre-vectorization.

From the box-plots in Figure 6, we conclude that the filters brought more definition to the lines in the images, which made the shapes appear clearer, and this has lead to a better vectorization. Therefore, white images with black lines get vectorized better when compared to the darker images.

C. Results of Experimentation

Concerning autoencoding, for the sobel-direct, the mean and standard-deviation values were 0.202 and 0.044, respectively. Whereas their inverse scored 0.699 and 0.124, respectively. For the canny-direct, the mean and standard-deviation values were 0.234 and 0.090, respectively. Whereas the inverse scored 0.741 and 0.150, respectively. These values further states a better learning rate for the autoencoder when the most important features of an image are darker than its other contained data. Therefore, we can conclude that when training an autoencoder, the semi-supervised neural network responds better when the training images have darker lines in their important features.

Following the same pattern, the box-plots show a better fitness of white images with black lines when compared to the darker images in vectorization.

VI. Evaluation

Evaluation is concerned with how abstract the resulting images are. As there are two pre-processing blocks (filtering and autoencoding), four different pipelines can be built: autoencoding, filtering, autoencoding-filtering, and filtering-autoencoding. After one of these selections is fed the images, a vectorization process is always cascaded at the end.

First, all of the resulting images are going to be evaluated based on their path count (size) and similarity to the input images. Then, a summary of evaluation is going to be introduced for each of the pipelines individually.

Before engaging in the evaluation, it is good to elaborate on the column naming of the upcoming plots:

- default: the default image.
- sobel, canny: the filtered version of the image by the respective filter.
- dec: the decoded version.
- vect: the vectorized version.
- A combination of two or more indicates the case of cascaded stages. A default-dec-sobel label represents the following: the default image is reconstructed with the autoencoder then filtered with the sobel filter.

A. Evaluating the size of the produced images

To evaluate the size of the image, we count the number of path objects generated in the SVG file. From Figure 7 (note that the graph is in logarithmic scale) we see that the autoencoder (*-dec-*) significantly reduced the size of images, as it keeps only the most important features. The reconstructed filtered images (canny-dec, sobel-dec) had a similar path count. Although it was much smaller than the ones that did not go through that step, it was still above the default images that were reconstructed and vectorized without any filtering. Finally, when filters were applied onto the default images that were put through an autoencoding stage (default-dec-sobel,

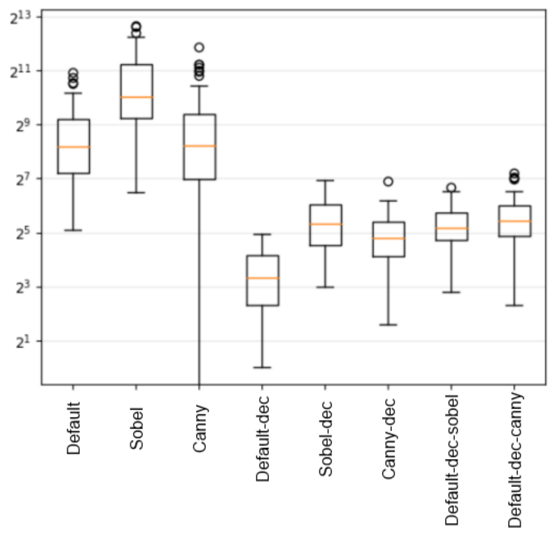

Figure 7. Path count of the resulted groups of vector images

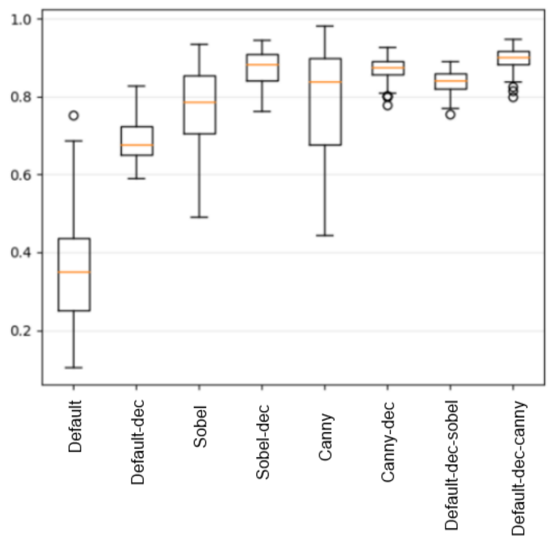

Figure 8. Vectorization accuracy of different pipelines

default-dec-canny), these images scored in size calculations very similarly to the filtered images when only reconstructed (canny-dec, sobel-dec).

B. Evaluating the quality of the produced images

A more accurate way of examining the efficiency of the vectorization process of each pipeline, is to compare the images and their vector versions (Figure 8). The pipeline of autoencoding-filtering-vectorization (two last groups on the most-right) seems to experience the highest SSIM, which indicates its fitness in vectorization. It made more sense for the autoencoder to reconstruct the images and then for the filters to come afterwards, putting emphasis on the important features of each image.

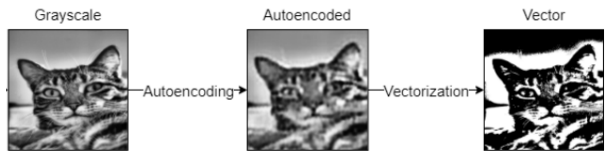

Figure 9. Autoencoding-vectorization pipeline

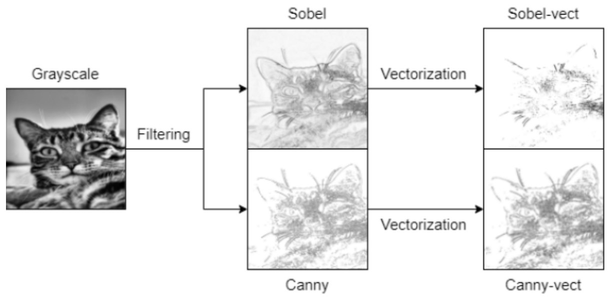

Figure 10. Filtering-vectorization pipeline

C. Implemented Pipelines: an evaluation summary

This is a summary of results evaluation for each of the pipelines individually.

- **Autoencoding-Vectorization:** This pipeline was based on the work of Fischer and Amesberger [1]. However, the implementation was different, and the evaluation was about the abstractness of the results. The quality of the vectorization is acceptable only in terms of general similarity. However, an abstract representation of the image is not achieved (Figure 9).
- **Filtering-Vectorization:** In this pipeline (Figure 10), the vectorization algorithm finds a difficulty in vectorizing the filtered images. This is due to the noises caused by the applied filters. Although the experiments showed that the quality of the vectorization increased when the images where taken as a light background with dark features, the noise involved created an obstacle for Potrace to convert thoroughly the images into a vector format, which resulted into losing data.
- **Filtering-Autoencoding-Vectorization:** This pipeline was built as an attempt to enhance the *Autoencoding-Vectorization* pipeline. Although the autoencoding stage was efficient in reducing the size of the images, it did not result in an abstract view of the image features. Therefore, a filtering stage was placed prior the autoencoding process. Unfortunately, this pipeline does not achieve the result intended. The autoencoding stage was supposed to reconstruct the filtered images in a lower complexity; but the case at hand is that, the autoencoding model is attempting to smooth the images, canceling the effect of the high-pass filters. This has resulted in a significant drop in the quality of the vector images, which is seen in Figure 11.
- **Autoencoding-Filtering-Vectorization:** Due to the results in the *Filtering-Autoencoding-Vectorization*

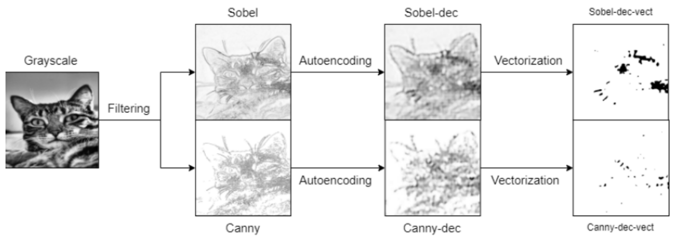

Figure 11. Filtering-autoencoding-vectorization pipeline

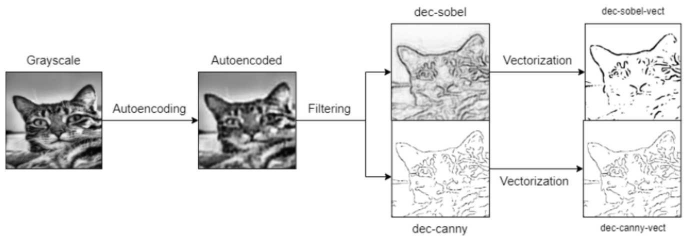

Figure 12. Autoencoding-filtering-vectorization pipeline

pipeline, it was clear that the filtering stage would act in a more proper way if it succeeded the autocoding process, rather than preceding it. This was concluded when the autoencoding model was seen to reduce the complexity of the images while introducing a smoothing effect. The filters were placed after the reconstruction stage to preserve the important features of the reduced-complexity image. This cascade shows an acceptable vectorization quality while resulting in the intended abstract representation of the images as shown in Figure 12.

As for providing more visualizations of the results that can be obtained with this pipeline, Figure 13 shows some random images that were fed to the Autoenconding-filtering-vectorization pipeline along with their respective output images. As can be seen, the features of the cats are extracted very clearly in all examples.

VII. Conclusion

This paper overall discussed the use of high-pass filters in vectorization pipelines, along with the autoencoding stage. It is concluded in this chapter that high-pass filters can enhance the training of an autoencoder, which in return make the vectorization process more efficient by preserving the important features of an image.

After evaluating the efficiency of the vectorization algorithm in every pipeline, it was clear that the images that went through the cascade of autoencoding-filtering, scored the highest in similarity, and the lowest in error. This points to the fact that the images that were reconstructed, preserved the most important features, which were brought up even more by the filtering part succeeding the reconstruction, which leads to not only a better vectorization but also a more abstract representation of the image.

This cascade of autoencoding-filtering gave decent results that matched the initial expectations; however, further work must be put into the structure of the models built, and their training dataset.

Concerning this future work, experiments showed that dark features on a light background in images can improve both the training of autoencoder models and the process of vectorization, which can be a good candidate for further investigation. From another side, the work in this paper deals with single-channel images (gray-scale); however, the vectorization of multi-channel images can be put through future experimentation.

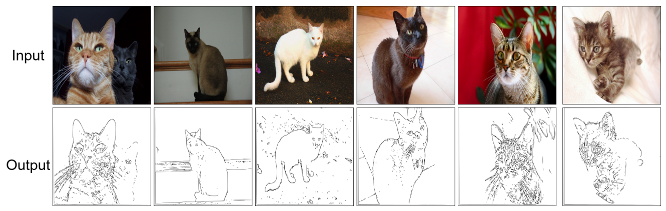

Figure 13. Some of the output images along with their input images of the pipeline Autoenconding-filtering-vectorization

References

[1] A. Fischer and M. Amesberger, "Improving image tracing with artificial intelligence," in *2021 11th International Conference on Advanced Computer Information Technologies (ACIT)*, 2021, pp. 714–717.

[2] W. Zhang, J. Sun, and X. Tang, "Cat head detection - how to effectively exploit shape and texture features," in *ECCV*, 2008.

[3] P. Selinger, "Potrace : a polygon-based tracing algorithm," 2003.

[4] N. Kanopoulos, N. Vasanthavada, and R. L. Baker, "Design of an image edge detection filter using the sobel operator," *IEEE Journal of solid-state circuits*, vol. 23, no. 2, pp. 358–367, 1988.

[5] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[6] M. Bessmeltsev and J. Solomon, "Vectorization of line drawings via polyvector fields," 2018.

[7] V. Lacroix, "Raster-to-vector conversion: Problems and tools towards a solution a map segmentation application," 03 2009, pp. 318 – 321.

[8] J. Xie, L. Xu, and E. Chen, "Image denoising and inpainting with deep neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012.

[9] Z. Gong, L. Xu, Z. Tian, J. Bao, and D. Ming, "Road network extraction and vectorization of remote sensing images based on deep learning," in *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, 2020, pp. 303–307.

[10] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[11] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[12] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[13] C. Sammut and G. I. Webb, "Mean squared error," *Encyclopedia of Machine Learning*, no. 4, pp. 653–653, 2010.

[14] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[15] S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu, "scikit-image: image processing in python," *PeerJ*, vol. 2, p. e453, 2014.

ISBN: 978-1-61208-954-6