Emilien Chevallier

emilien.chevallier@phelma.grenoble-inp.fr

A Master thesis presented for the degree of
Micro-Nanotechnologies for Integrated Systems
2019
STMicroelectronics
850 Rue Jean Monnet, 38920 Crolles, France

Physics-based machine learning-aided method for optical mask generation in the Lithography Process

From October 15th 2019 to March 13th 2020

Confidentiality: yes

Under the supervision of:

- **Company supervisor: Pascal URARD,** *pascal.urard@st.com*

Present at the defence: yes (Skype)

- **Phelma Tutor: Youla MORFOULI,** *Panagiota.morfouli@phelma.grenoble-inp.fr*

Grenoble INP - Phelma ; Politecnico di Torino ; EPFL
France - Italy - Switzerland
March 2020

Acknowledgements

I would like to thank all the STMicroelectronics colleagues for their pleasant welcoming, especially in my department where people never hesitated to offer me their help. I really enjoyed this 5 month placement as I was ideally welcomed. Internship students are considered as real members of the company, I have never felt excluded from the team as I was taking part in weekly meetings, participated in external activities such as skiing, and the company even paid part of my monthly travel plan.

I am grateful for all the team members who gave me the opportunity to work in a pleasant environment with highly skilled and experienced engineers. I hope to have the opportunity to work in the same kind of professional environment in the future.

I would like to thank more specially my supervisor Pascal Urard who led my project during this internship, as he was always available to provide me with all the explanations necessary to the achievement of the project. He also devoted a considerable time giving me advice not only useful for my internship but also for my career perspectives. I would also like to thank Charlotte Beylier from the OPC department as she took time to teach me about her work on photomasks and Clément Bonnafoux, a mathematical PhD student who helped me with the maths when it became a bit too abstract.

Finally I would like to thank my tutor Youla Morfouli, who took the time to come within the company to check up on me and to reply to my emails and numerous reports as to make sure I could make the most of what I have been doing during this internship.

Abstracts

English

Photolithography is a major process used extensively as a manufacturing process, for instance in the fabrication of integrated circuits. For such a process, one needs to develop photomasks, with a certain robustness to parasitic effects like diffraction, being a consequence of the ever-shrinking race in the semiconductor industry. With the applications of such circuits being always more complex and numerous, the customers are becoming more and more adamant concerning the quality of the products, whereas the technologies become more and more difficult to control. Such limitations need to be addressed to improve the reliability by using different approaches.

This thesis presents a process flow using a physics-based approach to model the generation of photolithographic masks using Machine Learning algorithms.

First, an introduction chapter presents the context of the project, then a second chapter exposes the theory and the important parameters and techniques behind the two concepts of photolithography and Machine Learning. In the third chapter we propose and detail a modelling and its implementation. Finally, in the fourth chapter, the realisation of the process flow is detailed with the results of the developed scripts, be it in the preparation of the EDA data, in the development of the main algorithm, as well as the implementation of the physics-based model we proposed earlier ; improvements to come are then highlighted as to improve the overall results.

Keywords: Photolithography, EDA, GDSII, Inverse Lithography, Optics, OPC, Defocus, Machine Learning, Gradient Descent, Neural Networks, Convolutional Neural Networks, VAE, GAN, TensorFlow, Python.

Français

La photolithographie est un procédé majeur largement utilisé dans la fabrication de circuits intégrés. Pour un tel procédé, il faut développer des photomasques, avec une certaine robustesse aux effets parasites comme la diffraction, étant une conséquence de la course sans cesse à la miniaturisation dans l'industrie des semi-conducteurs. Les applications de tels circuits étant toujours plus complexes et nombreuses, les clients deviennent de plus en plus exigeants quant à la qualité des produits, tandis que les technologies deviennent de plus en plus difficiles à contrôler. Ces limitations doivent être adressées pour améliorer la fiabilité en utilisant différentes approches.

Ce rapport présente un processus utilisant une approche basée sur la physique pour modéliser la génération de masques photolithographiques à l'aide d'algorithmes d'apprentissage automatique.

D'abord, un chapitre d'introduction présente le contexte du projet, puis un deuxième chapitre expose la théorie, les paramètres et techniques importantes derrière les deux concepts de photolithographie et d'apprentissage automatique. Dans le troisième chapitre, nous proposons et détaillons une modélisation et sa mise en œuvre. Enfin, dans le quatrième chapitre, la réalisation du processus est détaillée avec les résultats des scripts développés, que ce soit dans la préparation des données de design, dans le développement de l'algorithme principal, ainsi que la mise en œuvre du modèle basé sur les équations d'optique que nous avons proposé plus tôt; les améliorations à venir sont ensuite mises en évidence afin d'améliorer les résultats globaux.

Mots-clés: Photolithographie, EDA, GDSII, Lithographie inversée, Optique, OPC, Defocus, Machine Learning, Gradient Descent, Neural Networks, Réseaux de neurones convolutifs, VAE, GAN, TensorFlow, Python.

Italiano

La fotolitografia è un processo importante ampiamente utilizzato nella fabbricazione di circuiti integrati. Per un tale processo, è necessario sviluppare fotomik, con una certa solidità agli effetti parassiti come la diffrazione, essendo una conseguenza della corsa sempre più ridotta nel settore dei semiconduttori. Con le applicazioni di tali circuiti sempre più complesse e numerose, i clienti stanno diventando sempre più irremovibili riguardo alla qualità dei prodotti, mentre le tecnologie diventano sempre più difficili da controllare. Tali limitazioni devono essere affrontate per migliorare l'affidabilità utilizzando approcci diversi.

Questa tesi presenta un flusso di processo che utilizza un approccio basato sulla fisica per modellare la generazione di maschere fotolitografiche utilizzando algoritmi di Machine Learning.

Innanzitutto, un capitolo introduttivo presenta il contesto del progetto, quindi un secondo capitolo espone la teoria e gli importanti parametri e tecniche alla base dei due concetti di fotolitografia e Machine Learning. Nel terzo capitolo proponiamo e dettagliamo una modellizzazione e la sua implementazione. Infine, nel quarto capitolo, la realizzazione del flusso di processo è dettagliata con i risultati degli script sviluppati, sia nella preparazione dei dati EDA, nello sviluppo dell'algoritmo principale, sia nell'implementazione della fisica- modello basato che abbiamo proposto in precedenza; i miglioramenti a venire verranno quindi evidenziati per migliorare i risultati complessivi.

Parole chiave: Fotolitografia, EDA, GDSII, Litografia inversa, Ottica, OPC, Defocus, Apprendimento automatico, Discesa a gradiente, Reti neurali, Reti neurali convoluzionali, VAE, GAN, TensorFlow, Python.

Contents

Introduction

Company

My master thesis was carried out over a period of 22 weeks in the Research and Development department of STMicroelectronics. During this placement I was supervised by Pascal Urard, a System Design Director.

STMicroelectronics is a French-Italian multinational electronics and semiconductor manufacturer. It belongs to the world's largest semiconductor companies and is a leading Integrated Device Manufacturer delivering solutions that are key to Smart Driving and the Internet of Things. Around 7,800 people are working in R&D and product design, but the company also owns 18,500 patents, 9,600 patent families and around 590 new patent filings as of 2019. STMicroelectronics' creation dates back to 1987 by the merger of two government-owned semiconductor companies: SGS Microelettronica (Società Generale Semiconduttori) of Italy and Thomson Semiconducteurs, the semiconductor arm of France's Thomson.

Unlike many other semiconductor companies, STMicroelectronics owns and operates its own semiconductor wafer fabs maintains a strong commitment to innovation, and draws on a rich pool of chip-manufacturing technologies, including advanced FD-SOI (Fully Depleted Silicon-on-Insulator) CMOS (Complementary Metal Oxide Semiconductor), differentiated Imaging technologies, RF-SOI (RF Silicon-On-Insulator), Bi-CMOS, BCD (Bipolar, CMOS, DMOS), Silicon Carbide, VIPower, and MEMS technologies. ST is a multinational company, the revenue of which is approximately 10 Billion USD in 2019, and its principal wafer fabs are located in Agrate Brianza and Catania (Italy), Crolles, Rousset, and Tours (France), and in Singapore. These are complemented by assembly-and-test facilities located in China, Malaysia, Malta, Morocco, the Philippines, and Singapore.

During this internship I was part of a research team the focus of which was the implementation of Machine Learning for the upcoming technologies and the improvement of the current manufacturing processes.

Motivation

For my master thesis, I wanted to work on a project linked to manufacturing processes as I was always interested in working in this very field, although I developed a rising interest in the field of Machine Learning, as it is becoming a major asset to master on the job market. I was also expecting from this last internship in my education to improve my overall skills in the world of research and on the other hand to develop new skills that could be useful for my future. That is why when Mr. Urard presented this master thesis opportunity by directly contacting me, I knew that this project would completely fulfil my expectations as it deals with a technology that has both fields (manufacturing and AI) intertwined, so that the achievement of the project relies on multidisciplinary skills ranging from understanding Electronics Design through the use of modern design tools to AI design and implementation but also through physics modeling and simulations. Furthermore, I had already heard that STMicroelectronics offers a stimulating work environment where internship students are well supervised and completely part of the team which reinforced my motivation. I believe the fact that my curriculum in the NANOTECH courses would unfold itself to be significant in solving the problem by linking multiple skills such as understanding designers, using computing tools as Python, but also having studied manufacturing processes such as photolithography or etching is something which really made the difference for this internship, especially in my choice and enjoyment when working on it.

State of the art and objectives

Most devices developed by STMicroelectronics use many manufacturing processes when they are fabricated, such as photolithography and those are a crucial step to be improved. For this purpose, the company is investing more and more into research groups working with new modern tools, such as Machine Learning to develop a new approach and optimise such costly processes.

My missions were then to understand the needs within the company using my Nanotech engineering skills, to propose solutions with a flow to improve the lithographic process using Machine Learning, and to implement such a flow. This model will be used to:

- Improve the time required to develop a photomask, thus the yield of the company for future products to be manufactured.

- Develop knowledge of a Physics-based use of Deep Learning algorithms within the company.

My internship was then divided in three steps:

- Learn about Machine Learning techniques and methods to enrich our problem with it.

- Understand the limitations of the current photolithography process within the Optical Proximity Correction department to propose a physics modelling of the process.

- Implement the modelling and get to work with EDA software to end up with a process flow adapted to the company's tools, using both physics equations and Machine Learning on layout designs.

1 Theory

1.1 Photolithography

Here we will describe the photolithographic process, its main parameters and the techniques used to overcome the difficulties.

1.1.1 Generalities

In the semiconductor industry, a major process step in the manufacturing of circuits is the transfer of a design onto a substrate. Such a procedure can be realised using different methods and we will hereby stoop upon a process called photolithography, which is extensively used, sometimes several hundreds of times within a single chip manufacturing process flow.

This process works by illuminating a resist on a substrate through a mask, followed by etching the resist to obtain the desired image on the substrate as shown on the Figure 1.

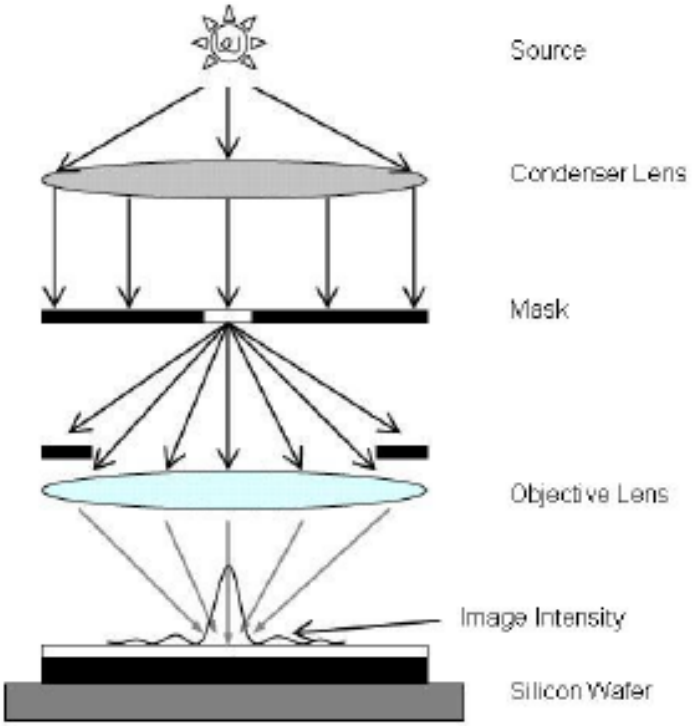


Figure 1: Representation of a typical photolithography system

This process is of critical importance as it conditions the fidelity of the transferred features and thus the proper functioning of the circuit. Light is sent from a source, the main parameter of which we will consider is the wavelength λ.
Light then goes through a condenser lens aiming at focusing the light on the mask. The mask itself is in our case a quartz square of 15cm$\times$15cm produced using electron beam lithography itself, containing the design information. Light is diffracted onto an objective lens or projector and is thereby refocused on the resist lying on the silicon wafer. The important parameters with λ are the numerical aperture $N.A$, the Mask function $\mathbf{M}(x, y)$ and the light intensity on the resist $\mathbf{I}(x, y)$, about which we will give more details later.

1.1.2 Scaling challenges

Due to diffraction effects, interference, processes and scaling of technology, some corrupt patterns can be transferred from the mask to the resist if no correction or tricks are empoloyed to prevent them. If we call d_0 the smallest significant distance of the design's features, also called resolution or pitch, and λ the wavelenght of the source, we can consider to have major problems when :

$$d_0 \approx \frac{\lambda}{2}$$

This is especially true for today's processes which go down to less than 10 nm for d_0 and the wavelength used are of 25 nm. Within the company, the wavelength used for most processes nowadays is 193 nm due to tricks as we will see in this part. Let us explain in a bit of further depth why the resolution appears to be limited.

Resolution limitations

The resolution is mainly limited by three direct parameters as depicted on the Figure 2 : λ, the numerical aperture $N.A$ and the angle of the m$^{\text{th}}$ order of diffraction θ_m. We have the following equation for diffraction :

$$sin(\theta_m) = m\frac{\lambda}{d_0} \tag{1}$$

The Rayleigh equation for the resolution is :

$$R_{pitch} = k_{mpitch}\frac{\lambda}{N.A} \tag{2}$$

with R_{pitch} the smallest "half pitch" that can be printed, k_{mpitch} a constant value which depends on the technology used, typically $k_{mpitch} = 0.28$.

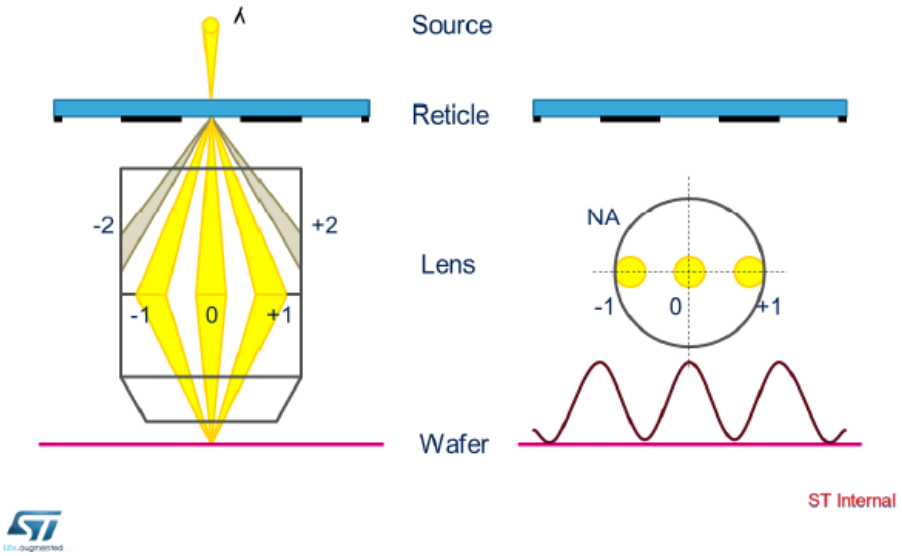


Figure 2: Lithographic process with large features

What happens when a mask has features too small and no modifications were made in the parameters can be represented on the Figure 3 : we lose details due to larger order of diffraction not being included in our Numerical Aperture.

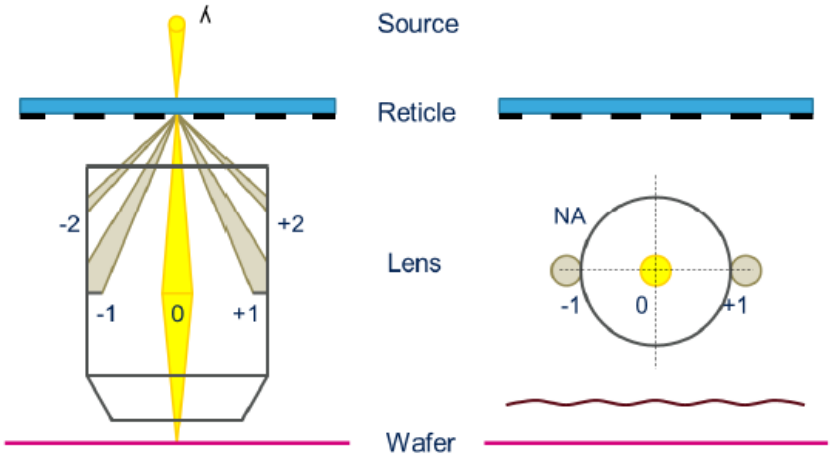


Figure 3: Lithographic process with imperfect parameters

Resolution Enhancement Techniques

In order to prevent and correct such phænomena, different techniques have been developed, to play with the main parameters, such as using light of shorter wavelength, Off-Axis Illumination, PSM (Phase Shift Masking) or OPC (Optical Proximity Correction) and others. These are called *Resolution Enhancement Techniques*, and their goal is to minimise R_{pitch} by including as many diffraction orders as possible in the focused image after the lens.

- Wavelength modification has been used in the past decades until reaching deep ultraviolet (DUV) light from excimer lasers with wavelengths of 248 and 193 nm. The advantage can be depicted as follow on the Figure 4.

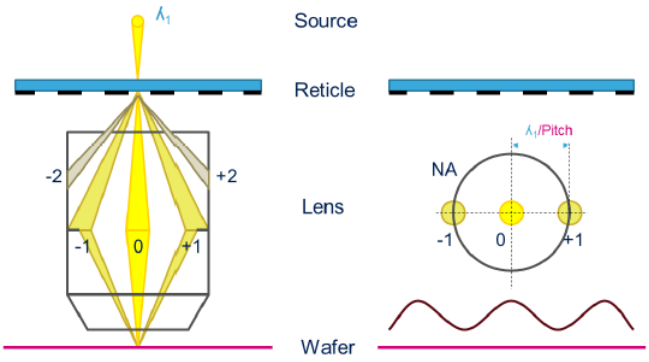


Figure 4: Lithographic process with wavelength shrinking

Decreasing the wavelength allows to reduce the spacing between the diffraction orders, or to shrink θ_m, allowing to have more details about the signal, thus a better resolution within the lens.

- Numerical Aperture modification is also a way to retrieve higher diffraction order as depicted on Figure 5. Increasing N.A allows the process to include more diffraction orders, even though they are widely separated because of the tight patterns on the mask. Therefore more details are printed on the resist, thus a higher resolution.

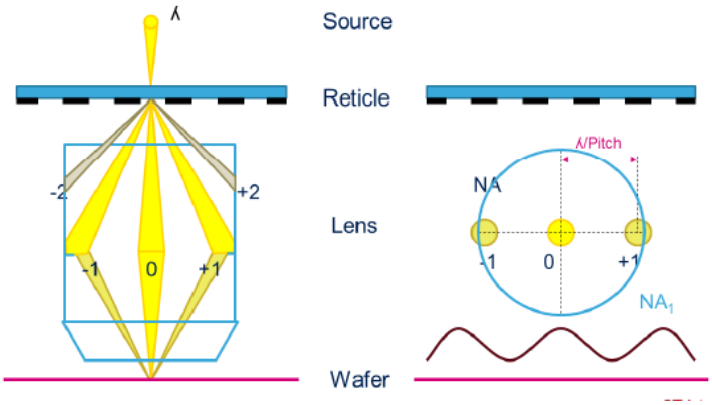


Figure 5: Lithographic process with Numerical Aperture modification

- In order to change the resolution is to directly change θ_m by adding an input angle, a technique also known as off-axis source. The Figure 6 represents this technique.

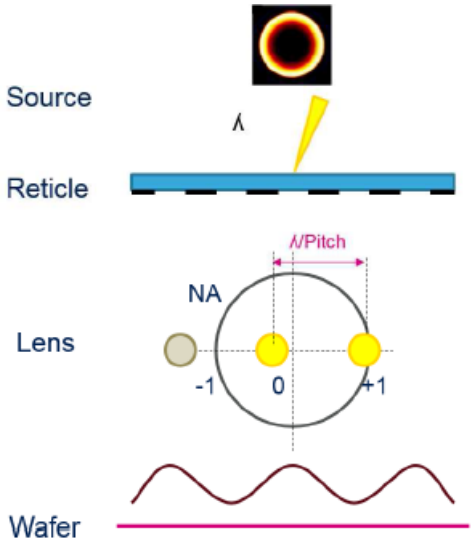


Figure 6: Lithographic process with Numerical Aperture modification

We can see that more orders are included thanks to this technique shifting the location of the original order. The source used to allow such an illumination is a circular source, which leads to the final technique : the choice of the source.

- The choice of the type of source also plays a role in the diffraction, directly having a significant impact on the angle θ_m viewed in 3D. Some examples of sources can be seen on the Figure 7, showing how changes in the sources allow for a better resolution.

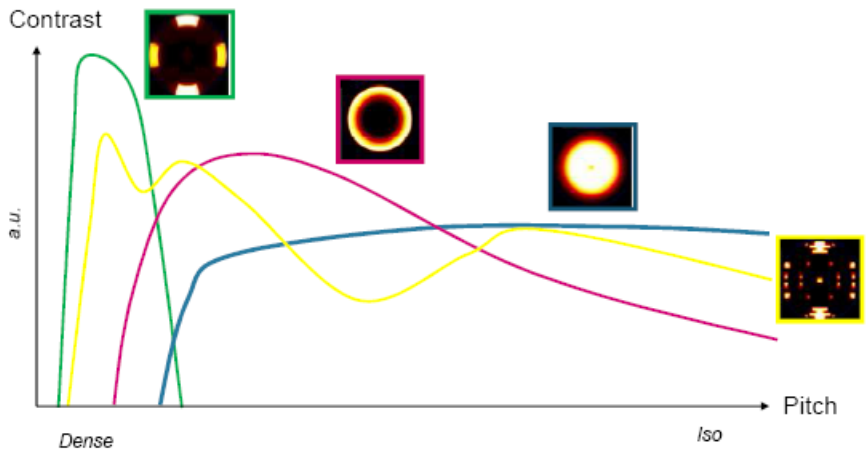


Figure 7: Comparison of the contrast obtained when using different source shapes as a function of R_{pitch}

- Last but not least, the parameter k_{mpitch} linked to process parameters can be played with in order to reduce R_{pitch}. It was a big deal at *STMicroelectronics* in the past years. Reducing it allowed to bring new technology nodes, at the cost of more complex imaging and the increase in proximity effects. OPC, which we will detail a little bit further or phase-shifting masks are examples of innovations leading to a reduction of k_{mpitch}.

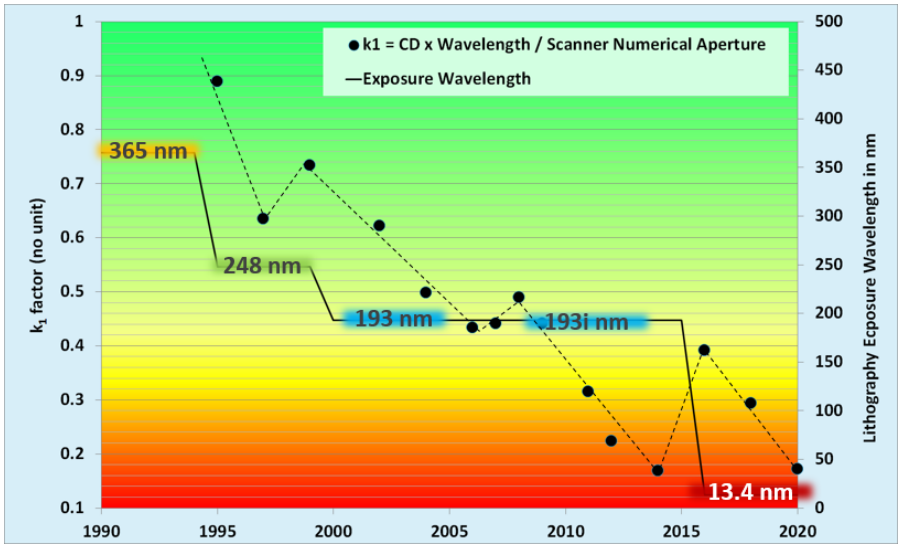


Figure 8: Evolution of R_{pitch} through the years as a the parameter k_{mpitch} evolved

Optical Proximity Correction

Let's focus on OPC as we were working with the OPC department and our project is well-enshrined in improving and optimising their routine.

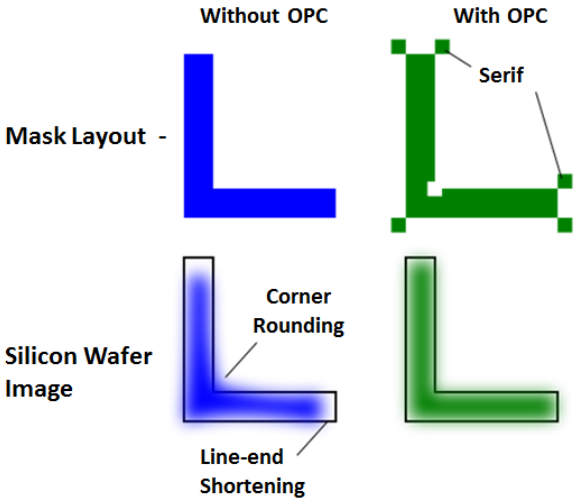


Figure 9: Depiction of the OPC results compared to non-OPC

As depicted on the Figure 9, the principle of OPC is to correct the shape of the design on the mask, so that it prints correctly in spite of the aforementioned parasitic effects. For instance, an approach can be to compensate the second and further orders of diffraction by adding or removing *serifs*, or little squares attached to the patterns, the size of which doesn't allow them to be imprinted in the substrate, but bringing some completion to the imprinted design. Those are directly sticked to the main shape of the design and should not be mistaken with *SRAF*s, or Sub-resolution Assist Features.

These features (SRAF) are actually extra additions of materials which will not be printed, being located away from the patterns and shapes. They most usually take the shape of thin lines outside the core-polygon we wish to imprint. Here is an example of an OPC strategy including SRAF addition on Figure 10.

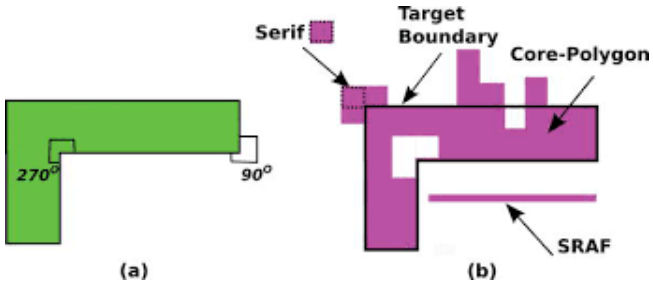


Figure 10: Illustration of SRAF strategy in OPC

1.2 Machine Learning for photolithography

In this section, we will present the notations used in Machine Learning, different methods used and high-end algorithms to generate new data using Deep Learning.

1.2.1 Generalities

A Machine Learning (ML) algorithm aims at learning without being explicitly programmed, by using different kinds of learning methods and steps. Most of what is explained here follows the lecture of A. Ng at Stanford University.

We can define some notations here to illustrate such algorithms: Let us call the X_i, the features or input variables of the algorithm, the number of which is $n \in \mathbb{N}$, and Y the target or output variable.

Let us be a training set

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\},$$

with $m \in \mathbb{N}$ the size of the training set, $x^{(i)}$ the feature vector of example i, and $y^{(i)}$ the target of example i. The number of features is the size of vector $x^{(i)}$ and is therefore n. So we have

$$x^{(i)} = \begin{bmatrix} X_1^{(i)} \\ X_2^{(i)} \\ ... \\ X_n^{(i)} \end{bmatrix}$$

For instance, if we had access to 1500 photomasks, one mask being our $x^{(i)}$, each with 50 features (like the number of shapes being the feature k, the size of the smallest features, or any characteristics) and the associated manufacturability being $y^{(i)}$, a Boolean : 1 if the photomask can be created by the manufacturers or 0 otherwise, we would then have for mask i: $m = 1500$, $n = 50$, $X_k^{(i)} = 28$, and $y^{(i)} = 1$ for a manufacturable mask

The notation $^{(i)}$ is important as it indicates a training example, which is given data and not data we wish to guess, find or compute.

Let us have a function h called hypothesis. This function is the link between the X_i and Y we wish to model, with parameters θ_i. It is only by learning the right θ_i that our function h will be optimal as to find Y given a vector x. For instance if we assume a linear relationship between Y and x, we take our h as a linear regression, we will have

$$h(x) = \sum_{i=1}^{n} \theta_i X_i,$$

and we only need to find the right θ_i to model this correctly. Therefore the θ_i will be the variables our algorithm needs to learn to minimise its own error, and h needs to be chosen wisely.

If we take the example of our 1500 photomasks, and we imagine our features to be the size of all the shapes on the mask, we could imagine an hypothesis being $h(x) = \prod_{i=1}^{n} \mathcal{H}(X_i - \theta_i)$, where $\mathcal{H}$ is the Heaviside step function so that if a feature is smaller than a certain threshold being its intrinsic parameter θ, the mask will not be manufacturable thus $h(x) = 0$.

In order to do so, we try to minimise the cost function with the best θ, being :

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}[h(x^{(i)}) - y^{(i)}]^2,$$

We can also decide to maximise the log-likelihood using the best θ, which is the same for our problem, the log-likelihood being a probabilistic tool with the same gradient as the cost function, for which we assume a probability distribution beforehand giving $P_\theta(y^{(i)}|x^{(i)})$, similarly to our assumption of h. We usually assume the distribution to be Gaussian with a mean of $h(x)$:

$$l(\theta) = log(L(\theta)) = log\left(\prod_{i=1}^{m} P_\theta(y^{(i)}|x^{(i)})\right)$$

To update our θ, we usually use a method called *Gradient Descent* when our cost functions or log-likelihood are differentiable, which consists in having:

$$\theta := \theta + \alpha\nabla_\theta l(\theta),$$

with $\alpha \in \mathbb{R}$ the speed of the descent, or learning rate. For the cost function approach, we just need to replace $l(\theta)$ with $J(\theta)$ We will see later that α can also be a function of the time if we want an adaptive (thus time-dependant) learning rate.

In the case of our 1500 photomasks example and modelling, it would not be wise to use a gradient descent , because the Heaviside step function we used isn't differentiable. Using Dichotomy could be a better approach to find the right θ_i.

Now, let us stoop upon different learning categories.

Learning Categories

Here we will describe three types of learning, which on this day are the most widely used.

- Supervised learning: this consists in giving a learning algorithm training data, composed of both inputs and desired outputs. Therefore, such a method allows the algorithm to learn a function to be used to predict the output associated with inputs one might want to feed it with.

  For instance, giving a huge number of photomasks and their manufacturability as an input to feed an algorithm so that it can classify photomasks in the future would be supervised learning.

- Unsupervised learning: this is a kind of learning in which the algorithm has to learn hidden features between the data based on unlabelled data. It is mainly used when we don't know the features and want to find density distributions of the inputs. Most of our work here is based on unsupervised training, as we don't know the exact output we wish to get in the end and our input data is unlabelled.

  For instance, giving a huge number of photomasks to an algorithm for it to learn their features and be able to generate new photomasks would be unsupervised learning, for we don't give any output to the algorithm.

- Reinforcement learning: this is an approach on how to teach an algorithm using the notion of cumulative reward. It does not need a training set based on paired input and outputs, and is trying to optimise itself by classifying its decisions as good or bad, thus leading to a higher or lower reward in the end, therefore learning a behaviour to adopt.

Supervised regression methods

In order to understand regression based on a simple example, let us explore regression based on supervised learning.

A regression problem, just like ours, is a problem in which we wish to model the relationship between the inputs and the outputs, as opposed to classification where we wish to classify data, that is to say to determine which label is associated to a given input.

Different regression methods exist, and we can here present some of them in supervised learning as examples. Depending on the domain of our output, that is to say $y \in \mathbb{R}$ or $y \in \{0; 1\}$ for instance, we have different regression methods.

For $y \in \mathbb{R}$, we could use a Gaussian modeling and use a least-square method.

For $y \in \{0; 1\}$, we could use a Bernoulli modeling and solve it using a logistic regression (also called sigmoid).

In our example developed in the Generalities section where we wish to classify the manufacturability of 1500 photomasks, we were facing a similar situation with $y \in \{0; 1\}$, in which we decided to use a Heaviside step function instead of a sigmoid. We usually use a sigmoid because of its differentiable properties as opposed to Heaviside.

For $y \in \{1; k\}, k \in \mathbb{N}^*$, we could use a multinomial modeling, and use the so-called softmax regression.

Actually, all of those modeling are part of the same family: they are special cases of the Exponential Family Distributions, and their difference lies in the choice of three parameters.

We will not step into the details of the calculations, however those modellings are usual and well-known, so that it can be useful to try to model a problem using an Exponential Family Distribution.

Now, some models have been developed and can be a bit detailed in here, although the mathematical theory behind them is quite solid and complex: Support vector machines and Neural Networks.

Support Vector Machines and Neural Networks are two Deep Learning methods widely used in the last thirty years for a handful of applications, and they mostly rely on our increased capability at performing complex calculations and handling huge quantities of data. This is why they belong to the Deep Learning sub-category within Machine Learning, for the number of parameters is particularly big, usually around several million parameters to optimise.

They allowed a fundamental shift in our society from using machine learning to solve classic problems to the burst of applications using deep learning for complex modelling. The main difference is the features extraction, which was manual or deterministic in classical Machine Learning, and became part of the learning in Deep learning and became automated within the algorithm.

This implies that we no longer need to find the features in the data before giving them to our algorithm, but it will detect them automatically, and even detect hidden features we could very hardly model by ourselves.

Support Vector Machines

A Support Vector Machine is a model which aims at separating data linearly into categories depending on its features. It constructs hyperplanes in the space of data, aiming at maximising the margin between the data, as to optimise the boundary between the different categories.

Here is a Figure showing the principle of a SVM on a two parameters:

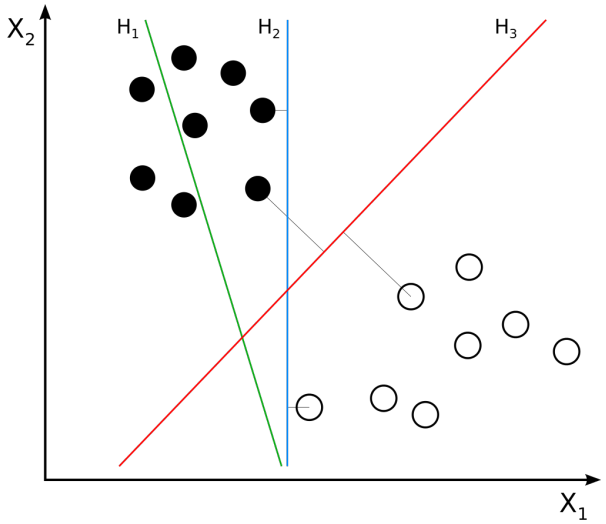


Figure 11: Example of optimal margin classifier on a two features X_1 and X_2 with H_3 being the optimal classifier and H_1 and H_2 being non-optimal solutions

Using a mathematical trick in affine spaces also known as the *kernel trick*, it allows non-linear separation if we decide to increase the dimension of the space we are working in, until it achieves to linearise the boundaries between the data.

This means that data which doesn't seem to be linearly separable in a dimension k can necessarily be in dimension l, with $k < l \leq +\infty$

Therefore once trained, a SVM is able to classify data or to generate new data by taking parameters within the right boundaries, even though its features are not linearly separable. We will not enter the calculations and mathematical details of a SVM, nevertheless the main advantages of such a method over neural networks is that they have a strong founding theory, they reach the global optimum generally better, they have no issue for choosing a proper number of parameters, are less prone to overfitting, as we will see that was an issue with neural networks, they also need less memory to store the predictive model, and they provide more readable results and a geometrical interpretation.

We will now see that neural networks are another method, which provides advantages over SVM, although it mainly depends on the given problem and the dataset at disposal.

Neural Networks

Neural networks are a model based on mathematical entities called neurons. They are basically a simple linear function, with weighted inputs, and single output. They also present an added non-linearity which is called activation function, such as a threshold (sigmoid, Heaviside, tanh...).

When connecting layers of similar neurons, one can model non-linear models by adjusting the weight of these interconnected neurons: this will then model a more complex function which can take a very large number of inputs.

During the training phase of such models, we add a back-propagation function, which aims at adjusting the weight using different methods, also called optimisers. This can be a gradient descent, but other algorithms also exist, such as the Adam optimiser widely used with Google TensorFlow's package for being an extension to stochastic gradient descent specifically designed to train Deep Learning methods.

Deep Learning is often a synonym of neural networks, for the many layers of neurons make the modelling of the learning quite difficult, despite working in many cases to represent models by extracting the features by themselves.

The interconnection between these neurons and the function each of them is designed to perform forms different types of layers. Famous layers include Dense layers, where every neurons are interconnected, or Convolution layers, which perform convolutions on data, as we can see in the Appendix about convolution, but many others exist or can be created for different purposes. Convolution layers have four main parameters, being their dimension or filter, their kernel or stride, the padding being a way to fix edge issues, and the activation function.

Let us give a small example of an activation function we had to use extensively in this very project : maxpooling. The maxpool function aims at reducing the dimension of a dataset, by discretising it in the way of taking the maximum out of a group of data. This is a simple non-linear operation which is widely used as to detect the important features of a dataset when performing hidden features detection.

Neural networks are particularly useful as a brute force algorithm when trying to optimise a model we cannot simply model otherwise, and needs a huge amount of data to be correctly trained.

The pros of such a model are that it provides really good results in terms of modelling, and are more accurate with the number of data used to train them, however they are hard to converge, and require a lot of computational power because of their elevated number of parameters to simultaneously optimise.

Choices

In the end, we decided to use Neural Networks for our problem, due to several factors such as the amount of data we actually possess, the ease of use of such a method with Python's libraries, and the global trend of this new decade.

We used the theory aforementioned to better understand the essence of Machine Learning, as to make the best choices, therefore the use of a VAE-GAN with Convolution Neural Networks, and not a SVM, whereas the regression used is a least-square model as we will see in the Modelling section.

However the question of using SVM versus Neural Networks is fundamental in the way of solving our problem, thus the paragraph on SVM in order to show that our problem could be solved in numerous ways and we had to choose and adapt to what suited the best.

1.2.2 Generative methods

In order to generate images from a model, several tools and algorithms can be used. In our very problematic, two tools seemed to fit with the requirements and the recent trend : VAEs and GANs.

Variational Auto-Encoders

- Generalities

  A Variational Auto-Encoder as defined for the first time in [8], also benempt *VAE*, is a combination of neural networks, the idea of which can be explained by the will of encoding data to a reduced latent space, and decoding it afterwards in order to get the data back.

  It can be seen as a lossy compression of data, in which the encoder and the decoder (both neural networks) learn to minimise the loss of the compression and the accuracy of the decompression by learning the features within the data, and being able to re-generate it.

  Depending on the rules defined in the loss function, a VAE can be trained to modify pre-existing pictures by encoding the original picture, and decoding it by minimising the loss by comparing the decoded picture to a different picture, thus switching from the original data to consciously modified data. A famous example is turning horses on a picture to zebras and vice-versa, or any translation algorithm as explained in [6], both of which are cyclic in their loss. Indeed, the transformation is in fact reversible, and should work both ways. We call the loss a *consistency loss*, for we perform a cyclic loss calculation translating from A to B then from B to A' as to obtain the same A in A'. This type of loss is particularly useful when we don't have a way to directly retrieve A from B or to know if the B we have is correct.

  The specificity of a VAE is the *variational* part, as opposed to a regular auto-encoder. As a matter of fact, instead of encoding or compressing the data by reducing its dimension and learning to map the features to a discrete space, that is to say turning a vector into a single point, a *Variational* Auto-Encoder will map it to a continuous probabilistic space. To come back to the latter analogy, the vector would be turned into two values, being a mean and a standard deviation.

  The decoder just has to take a realisation or sample within the proposed distribution in order to do its job as in a classic Auto-Encoder. When decoding, a Gaussian noise will be then added in order to train the decoder for more robustness to noise and various realisations, therefore achieving greater accuracy in encoding and decoding the data, as well as aiming at more diversity in the generated data.

  Here the given pattern is a handwritten digit we could want to print through the lithography process, and the decoder learns to recreate it from a low dimensional representation created by *realising* events using the means and standard deviations encoded.

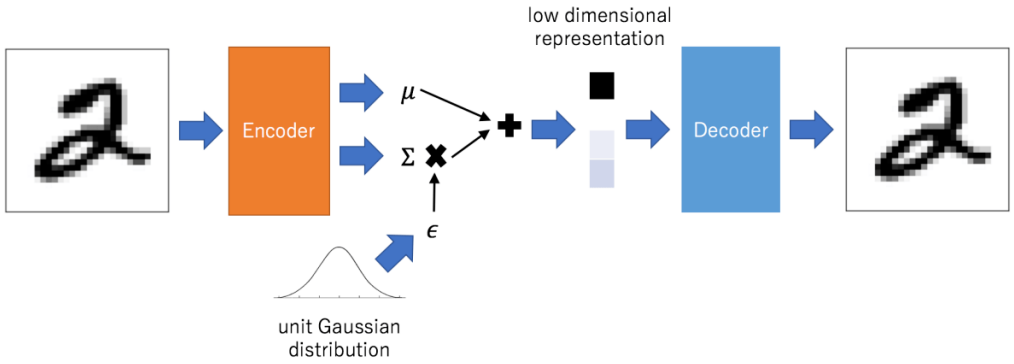


Figure 12: Variational Auto-Encoder's simplified functioning on the MNIST database

- Construction

  A Variational Auto-Encoder is typically made of two convolutional networks, one being the encoder and the other the decoder. The strategy is to reduce the dimensions of the data by using convolution layers on it within the encoder, and rescale it in the decoder using transposed convolutions.

  A convolution layer is a neural network structure which performs convolutions (or cross-correlation to be more accurate) on a dataset with a convolution kernel, whereas a transposed convolution layer performs an opposite operation, which allows for extrapolation of the data according to the activation function we choose.

  In-between convolutional layers, other types of layers can be added, such as Dense layers to perform rotations, scaling, and translations of our data, or Input Layers to prepare the input data into the convolutional layers.

  The more convolutional layers are added, the more parameters there are to evaluate and train, the more the risk of overfitting is high (which means adapting our model to our dataset only and not to any situation), the higher our accuracy in terms of control and the slower the training time.

Generative Adversarial Networks

This structure is made of two distinct blocks, one being a generating block, and the other a discriminating block.

The generative part must generate some data, while the discriminative part must help the generator to generate the right data by classifying the generated outputs as fake or real. By training together, this helps the generator *converge* to accurate results up to the point where one cannot distinguish between inputs from the training set and generated data.

The generative part can be typically made of a VAE, which is trained on real data as to be able to reconstruct data correctly as pre-learning. After a while, we can remove its encoder and only input some low-dimensional noise, which the decoder will interpret as something encoded, and try to decode as faithfully as possible : this leads to the generation of new data according to specific distributions such as the faces shown on the Figure 13, where the networks learned the distribution of parameters making a face on high-resolution pictures.

Figure 13: Samples of high-resolution generated faces with a well-trained GAN by Nvidia [7]

We understand that the discriminator after the training has to be confused, as not able to recognise a generated output from one in the dataset, so basically returning a probability of 0.5 for the two aforementioned categories.

Interestingly, the generator can also be used as a tool to extract features or as a translator, which is mostly the case in our very problematic.

Indeed, not removing the encoder allows to learn how to encode a certain image, so to reduce its dimensions but to keep its main features, and to decode such features in *another* space. If one thinks of a French-English translator application, we could have sentence in French. Reducing its dimensions may allow to get rid of the syntax, but keep the idea behind the sentence, while decoding it could be seen as rewriting the features of the very sentence with English flourishes.

Some Neural Networks applications such as the StyleGAN [7] even allow to distinguish between the different features or *styles*, as to decode an image with the ones we wish to apply to it. It allows the mixing of different *styles* as well, by extracting the features of different images, and choosing between each feature to add when decoding as a single image. It can be seen as an image translation here, but even more like an image dissection and recombination.

As for the loss several methods can be employed. When there is no way to know if the generated data is what we really wanted, as in some cases of *unsupervised learning*, and the goal is to transform an object into another through a reversible process, *for instance turning horses to zebra or zebra to horses* [12] [15], a *consistency loss* can be used as well.

2 Modelling

We will now present the modelling we used to represent Inverse Lithography, the strategy we chose to implement such a modelling, then we will present the flow used for the Assist Features (SRAF) and defocus robustness problematic, and how we decided to implement it.

2.1 Inverse lithography technique

The very first objective is to generate a mask by taking into account the physical equations and parameters as to shortcut the OPC process and improve the yield and timing in the company. This way, we can save resources and get rid of the OPC limitations with another approach. In order to do so, we focused on performing *ILT*, or *Inverse Lithography Technique*. This is a rigorous way of modelling lithography as an inverse optimisation problem, as to determine the mask shapes which deliver the desired on-wafer results. A lot of work has been done in the past in this very field, such as [1], and the method we use here takes advantage of the pre-accomplished work with the calculation power unveiled by today's technology.

2.1.1 Developed model

In order to model our problem, we chose to follow the approach of recent papers, while having major changes in the main algorithm, which we will discuss in the next section. Most recent papers decide to model the lithographic process by splitting it in two problems, the first one being the modeling of the aerial image we produce after the mask, and the second one being the resist model, or behaviour of the resist's chemical alteration. The aerial image model relies on Hopkins' diffraction theory [5], and a lecture from CalTech on Point Spread Functions, so that we can link our three main parameters by stating that

$$\mathbf{I}(x,y) = |\mathbf{h}_{\lambda,N.A}(x,y) \circledast \mathbf{M}(x,y)|^2, \tag{3}$$

with $\mathbf{I}$ being the light intensity of the aerial image at a certain location, $\mathbf{M}$ is the Mask function which is a Matrix of 0s and 1s corresponding to the mask having a hole at a certain location or not. Here, $\circledast$ is the convolution operator and $\mathbf{h}$ is the point-spread function (modelling the diffraction and other effects through the holes of the mask). It is an Airy pattern modelling the influence of each point in space on the centre of the figure. If we consider no obscuration on the source, it is given by:

$$\mathbf{h}(r) = \frac{J_1(2\pi r N.A/\lambda)}{\pi r N.A/\lambda} \tag{4}$$

Here, J_1 is the first-order Bessel function of the first kind, and r is the distance from the centre of $\mathbf{h}$: $r(x,y) = \sqrt{(x-x_0)^2 + (y-y_0)^2}$ and (x_0, y_0) is the location of the center of $\mathbf{h}$, so the centre of the convolution kernel. We can see that the physical parameters λ and $N.A$ are present here in the representation of the diffraction through $\mathbf{h}$.

The resist filtering model is also usually taken as a sigmoid of the light intensity arriving on the photoresist submitted to a threshold as we can see in [13], [2], [10], for the sigmoid is a continuous function and derivable, the gradient of which is more easily calculated than a hard Heaviside function. Indeed, the light arriving is only sufficiently absorbed when it is superior to a certain value, so it is filtered. It is to be noted that we could choose a constant threshold anyway, or a variable one using other functions, but a sigmoid is particularly relevant in the context of

machine learning, it is adjustable, and its derivative simple enough to be calculated quite fast by the algorithm. Therefore we have the final image $\mathbf{Z}(\mathbf{I})$:

$$\mathbf{Z}(\mathbf{I}) = sig(\mathbf{I} - tr) = \frac{1}{1 + exp[-a(\mathbf{I} - tr)]}, \tag{5}$$

with tr being the threshold of the photoresist, so the intensity which is considered as enough for the photoresisted to be imprinted. The parameter a controls the steepness of the sigmoid.

Now, the goal of our algorithm and our project is to find $\mathbf{M}$, knowing that we know which $\mathbf{Z}(\mathbf{I})$ we want : the target layout is called $\tilde{\mathbf{Z}}$. This is an inverse optimisation problem, which we will at first try to solve using an algorithm called *gradient descent*, the theory of which we have enlightened in the previous section. In order to do so, we need to minimise a variable, and the one we choose here is the norm of the difference between $\tilde{\mathbf{Z}}$ and $\mathbf{Z}$:

$$\mathbf{F} = \|\tilde{\mathbf{Z}} - \mathbf{Z}\|_2{}^2, \tag{6}$$

$\|.\|_2$ is called the *l2-norm*, it represents the square root of the sum of the term-by-term components of the matrix of the squared difference between $\tilde{\mathbf{Z}}$ and $\mathbf{Z}$ here, so this seems pretty logical to wish to minimise it, as to generate the desired layout in the end. We take it squared as to remove the square root, and we use this particular norm as to stay in a Hilbertian space where convolutions, Fourier transforms and more tools are well defined. As to find the optimal mask $\mathbf{M}$, we can update it iteratively by using *gradient descent*, which gives :

$$\mathbf{M}^{n+1} = \mathbf{M}^n - \alpha.\nabla\mathbf{F}, \tag{7}$$

with n the step in the algorithm and α the speed of the descent.

We then have to calculate the gradient of $\mathbf{F}$. This calculation is very much difficult, so that we will rather take the results from the aforementioned paper [2], that is to say :

$$\nabla\mathbf{F} = -2a.\mathring{\mathbf{h}} \circledast [(\tilde{\mathbf{Z}} - \mathbf{Z}) \odot (\mathbf{Z}) \odot (\mathbf{1} - \mathbf{Z}) \odot (\mathbf{h} \circledast \mathbf{M})], \tag{8}$$

Here, $\mathring{\mathbf{h}}$ is the rotation of matrix $\mathbf{h}$ by 180° in both directions, $\odot$ is the element-by-element multiplication, and $\mathbf{1}$ is a matrix of 1.

To conclude, here is the computational basis for our algorithm based on a physical model of lithography.

2.1.2 Implementation

As to implement this method, we chose a generative approach using neural networks. This requires several stages as depicted on Figure 14, as well as major customisations of a classic GAN algorithm.

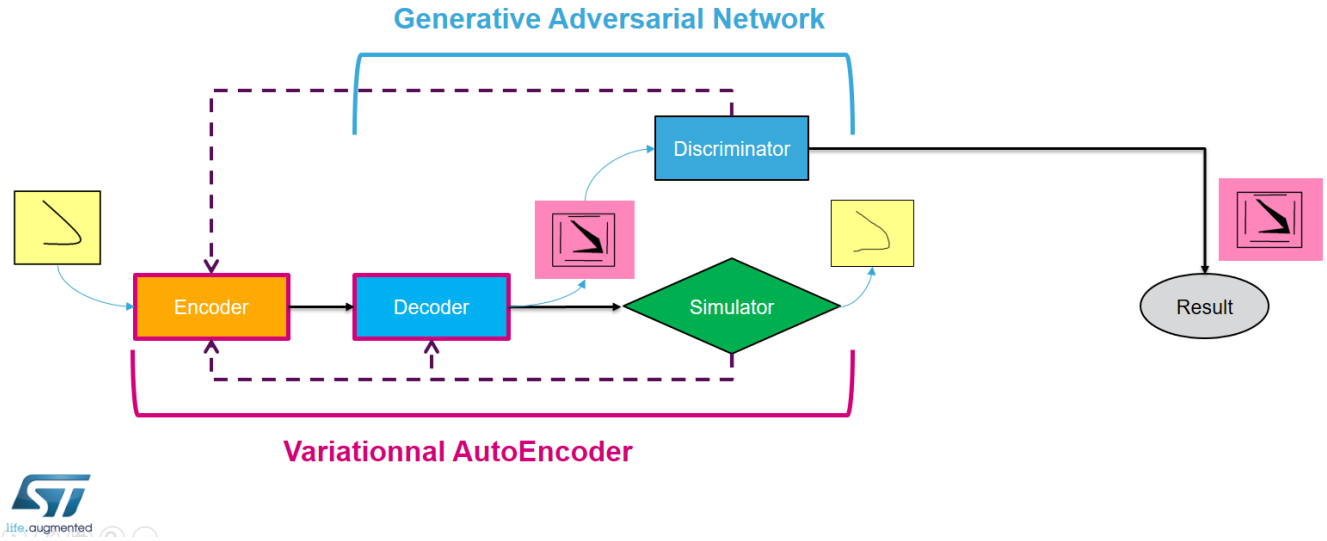


Figure 14: Schematic of the ILT algorithm

Generative networks

We chose to use a Variational Auto-Encoder as in the very papers [8] [14] [13], in order to generate masks from a given design. The input, as in the straight yellow pattern on Figure 14 takes designs of $4\,\mu m^2$ from the layout. It is then encoded as aforementioned, in order to be decoded as a mask.

This is the **major trick** allowing us to generate ILT masks: as the encoder learns the features of what being a "target" is, the decoder learns to map these features as to generate a mask. This could be achieved without the use of a simulator, but the company already had one in order to compare the original target with the target being the result of lithography using the mask our decoder generated. We will see later that their simulator also had many advantages over the loss function from the model.

As of so, several loss functions can be used to ensure the fidelity of the design, such as the one described is our developed model if we forget the simulator, which we could optimise to ensure our algorithm takes the process and physics into account. However simpler loss functions can also be used as a first step, if we manage to calculate $\mathbf{Z}$ in another way.

Simulator

This stage is key to succeed in our endeavour, for it allows us to simulate the lithographic process, thus knowing if the generated mask is of any value. Our loss function would allow not to have a simulator **because it would simulate itself**, however the team decided to use their own tools, as they might be more suitable to their technology. Moreover they were already configured as to check the manufacturability of our masks, which our loss didn't explicitly take into account. Therefore a simpler loss could be used using the result of the simulator. We will nonetheless see in the last section of this report, that the team came back to using the physics-based loss as an improvement over calling an external simulator.

The simulator we addressed is a script using Mentor's *Calibre* software in order to visualise the outcome of our mask. It is a DRC written script (Design Rules Check), which we adapted for several purposes as we will see in Section 3. It allows us to generate lithographic patterns on the resist, as to compare the initial design and the one simulated with the mask.

It can also simulate defocus, and we configured it as to choose which layers we included in the simulation.

An important detail about the simulator is that it allows us to know what the best result on-resist would be from a given design. Therefore we used it as well in the loss-function to generate and compare the *best-case* target, which we will call *the rounded target*, with the simulated design.

Discriminating network

This part is there to ensure the mask we created does not only "work" when simulated, but is also manufacturable. This means that the mask we generated has to respect some criteria, which the discriminator learns to map by training using both regular ILT masks, and some generated by our generative network.

This allows our generative network to try to please the discriminator up to the point when the discriminator cannot recognise which masks are generated, and which are real. It is important to notice that the discriminator is only viable for *STMicroelectronics* owns a huge amount of data of ILT they created for years, therefore making the training feasible.

We can now see that every variable found in equation (8) is determined and the only real unknown is $\mathbf{M}$: $\mathbf{Z}$ is simulated from the generated mask and $\tilde{\mathbf{Z}}$ is given by the company and calculated by the simulator ; $\mathring{\mathbf{h}}$ is calculated from the Bessel function, λ, and $N.A.$ and from $\mathbf{Z}$.

If we were to use the loss from the modelling, we would then only need the layout giving $\tilde{\mathbf{Z}}$, because $\mathbf{Z}$ would be calculated within the loss itself, therefore the simulator would only be used to prepare the data, that is to day calculate $\tilde{\mathbf{Z}}$.

2.2 Defocus Robustness and Assist features addition

After the theorising of ILT, we figured out that such a complete process could be very demanding in terms of realisation time so we kept it to address different and smaller problems. One of the improvements one can think of when they are doing OPC is defocus robustness, which means creating a mask, the target design of which doesn't change much even though the illumination is subject to a defocus. We chose to focus on this very issue, by generating assist features with Machine learning.

2.2.1 Flow for OPC improvements with Machine learning

The goal is then to add *SRAF*s to a pre-optimised mask. Indeed, after the mask is submitted to OPC, the SRAF addition step takes half the time of it, and is not as accurate as one could wish.

Therefore, it would be useful to use the aforementioned method as to add correct SRAF and thus improving the overall quality of the mask. The addition of SRAF is also a way to tackle the defocus problem, as those features are particularly useful in compensating the spatial orders of diffraction on the resist.

The whole concept relies on taking a previously-generated mask by the OPC team, and adding better SRAFs as an extra layer to merge on the layout.

For this a simple modification of the latter algorithm is required as we can see on the Figure 15, a flow schematic developed by the team working on the project, highlighting the main functions we needed to create or optimise and take into account.

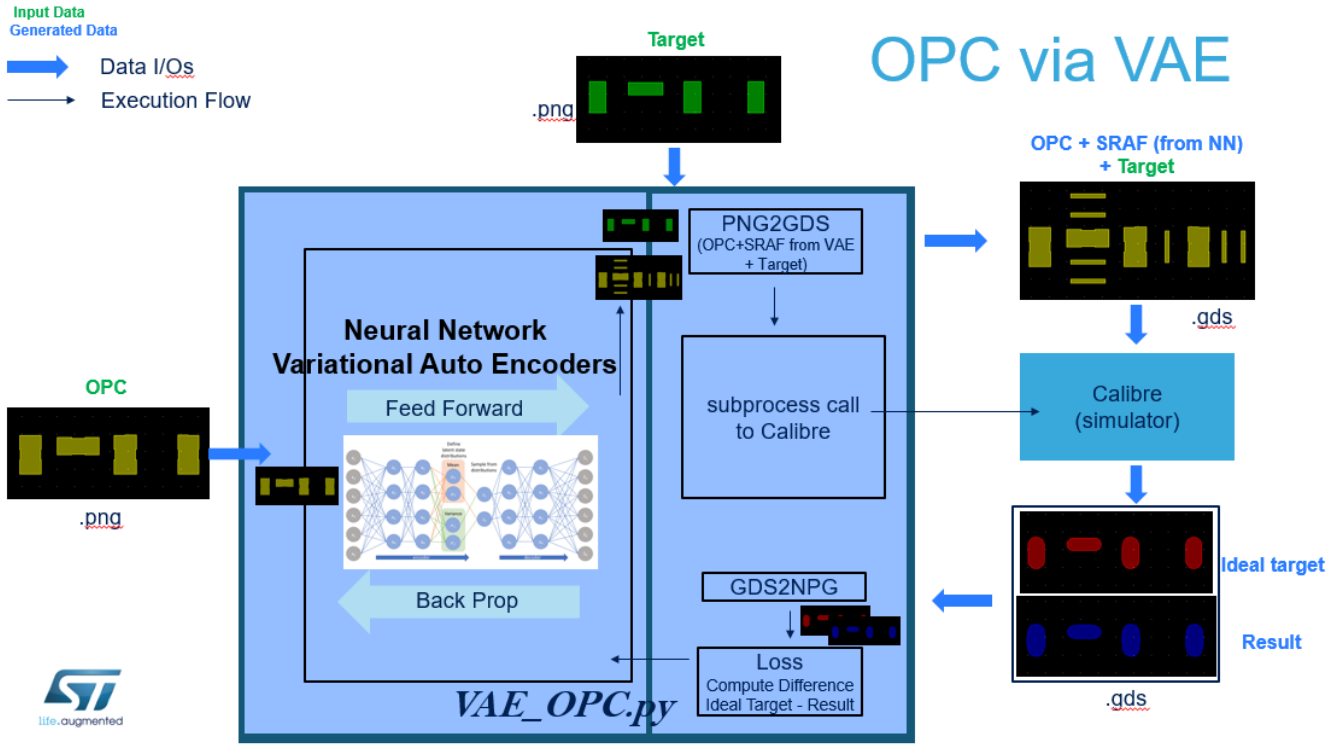


Figure 15: Schematic of the flow for OPC with Machine Learning

This Figure will be better explained later in the Realisation section, however several differences with the previous ILT flow can be highlighted. After taking a pre-processed mask as an input, that is to say a mask for which we already applied OPC algorithms, the Machine learning algorithm learns to add the SRAFs at the right places, and the loss function we use is a simple *l2-norm* without any physical properties, between the desired design (ideal target on the Figure 15) and the simulated one.

It is important to notice that the same loss as before could be used, that is to say the one derived from Hopkins' Diffraction model [5], however the algorithm should then be configured as not to modify the pre-existing mask, for we don't want to touch any pattern but the SRAFs.

For this, it is therefore important to separate the layers and to superimpose them when simulating the photolithography process.

The main project took the path of improving the OPC process with the SRAF generation for defocus robustness, for it was a *quick-win* for the company. Using physics and Machine Learning, this is considered as a first step and a guinea pig to test the efficiency of such methods, hopefully leading to more funding and investments in this very field in the future. This is particularly true as this would at first, if successful, allow to ease the work of the OPC team, improve the yield by reducing the processing time, and convince the company to a shift towards Machine Learning based on smart understandings of the physics processes such as photolithography.

Therefore, instead of jumping into ILT directly, it seemed more convenient to work on the defocus improvement using SRAF generated by model-based Machine Learning.

2.2.2 Implementation

In order to do so, we take the same algorithm as before, although the VAE is trained with mask inputs as mentioned earlier. The desired design is also used as to perform the loss function, but the interesting part is when it comes to the training.

It is important to train the whole algorithm by using different pictures of defocused designs, so that the defocus robustness criterion is respected. Therefore, we developed a DRC script allowing the simulation step to be done with several defocus values ranging from -50 nm to +50 nm, as will be shown on Figure 28 later.

The simulator would then simulate according to a random model it chose amongst the different focus values, thus having our network optimising its generation almost independently from a slight focus variation.

This was crucial to succeed having the algorithm performing what it was meant to, however one has to keep in mind that the defocus values have to stay in a low range as a matter of trust in the company's technology, but also to have efficient masks at a 0 nm defocus value : one does not want the masks to be average everywhere, but to be optimised as much as possible independently from the focus variability.

3 Realisation of the project

Once the theory and the goals were clearly defined, it was time to engage the realisation part, which proved itself to be extremely challenging, due to numerous limitations. In the end, the project involved at least four people working on different parts of the realisation, and the whole algorithm is not finished as of yet. Here we will see three main steps :

- how we prepared the data due to the challenges of working on a manufacturing problem,
- how we constructed our dataset and how we chose to run the whole flow,
- the steps of realisation and testing.

In the latter, we will present how we ran a well-known algorithm, the steps to adapt it for our very problematic, the different results we obtained when changing parameters and their influence, the realisation of the previous modelling part, and the improvements of which one can think for the rest of the project.

3.1 Preparation due to the lithographic process

First of all, the preparation of the algorithm was a huge part, and this is the section upon which we will focus here. We used Python 3.7.4, TensorFlow 2.1, and Mentor's *Calibre* v2019.4_36.18.

3.1.1 Format

Here is the main format issue needing preparation : the file format of the data linked to the photolithography software was the very first limitation we encountered was the format of the data. Indeed, the software used to create the dataset and perform simulations being Mentor's *Calibre*, we needed GDSII conversion to express our data as layout designs and masks. Indeed this software uses the GDSII file format, which is a binary vectorial format describing the layout with keywords to indicate the shapes, coordinates and the layer.

Here is an example of an ASCII-converted GDSII layout file, which we converted in order to better understand how the design layouts were encoded by Mentor's *Calibre*. We can clearly see that the file description uses keywords such as "TEXT" or "LAYER" with coordinates for the software to reconstruct later, which explains why we cannot directly generate the masks in this very format, albeit it would be another problem.

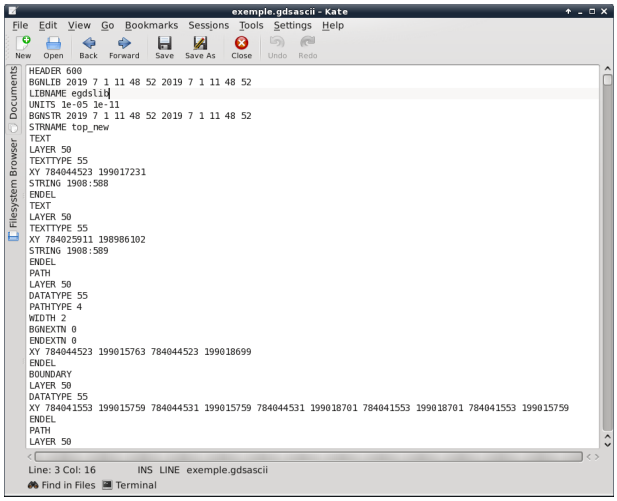


Figure 16: Sample of ASCII-GDSII layout file

However the Machine learning algorithm using convolutions and matrix operations, it was wiser to perform the whole project using images, as an automated generation of GDSII using machine learning would be a different problem. After testing several formats on huge pictures, we retained the PNG format for its ease of use in Python modules such as TensorFlow, its limited final size and its lossless compression ensuring to keep the pixel accuracy we needed, against JPG, BMP and SVG.

In the end, the algorithm we created is format-independent, meaning that it performs the same for any format we would use in the future, by automatically converting any photolithography picture into a binary file loaded into the RAM. As a matter of facts, these very pictures were to be stored in a database, the architecture of which had been developed by previous interns, and the more savings the better for us. It is to be noted that we stored the images locally on a Hard Disk Drive of 350 GB, in order to perform tests and didn't include the final database yet.

Therefore, we also had to develop a python script allowing to convert a GDSII file into a PNG file. For this, we used the library gdsCAD at first from Python, we adapted the source code to run with Python 3, for it only existed for Python 2. Afterwards, we customised some functions to get it return only the layers we was interested in, and save them keeping the accuracy we wanted, in our case being 5 nm per pixel. We can see here on the Figure 17 a direct comparison of the GDSII files and the picture we created.

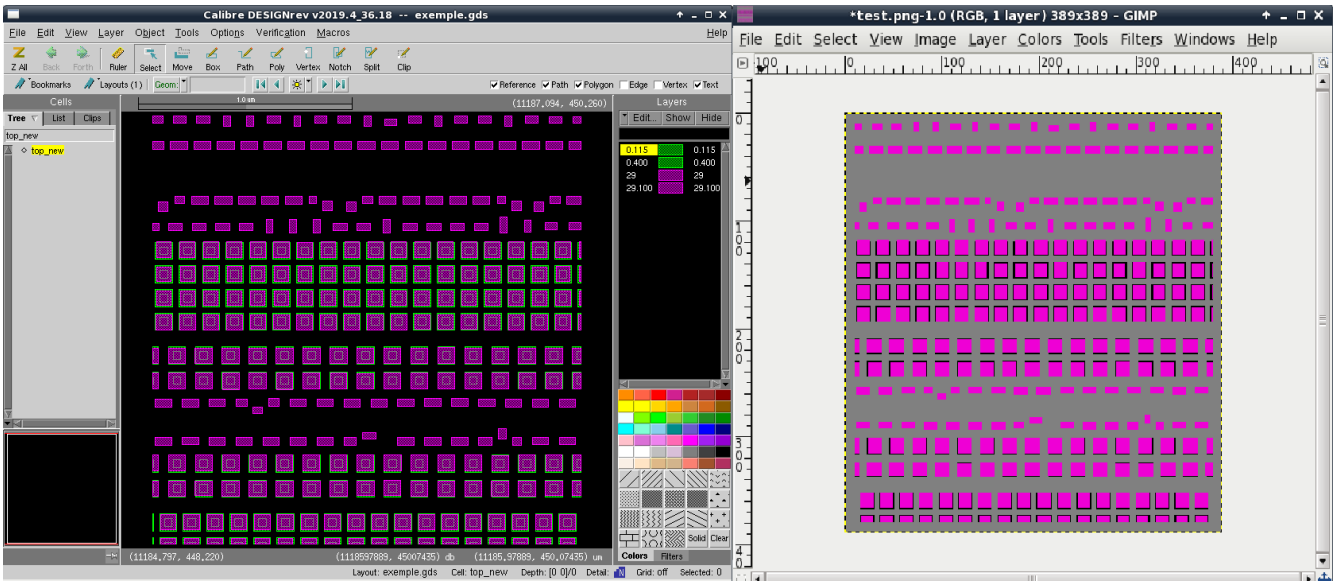


Figure 17: Comparison of GDSII and PNG result

Another intern had previously worked on a new representation of GDSII files by compressing them by a factor 36.000. Using known specificities of the GDSII format, as well as a Peano's curve, he managed to encode pixels in a complex, yet space-saving way. His representation could be used as well in our algorithms, for it respected the shapes and accuracy we wanted, nonetheless it was not used for now but could be envisioned as an improvement as seen in the Improvements section. An open-source module called Nazca-design also allows the reverse conversion from PNG to GDSII, as to re-enter the mask generated into the simulator, and it was eventually used for the two conversions, leading the team to develop their own module based on it to have their own interpretation of GDSII files in Python.

3.1.2 Size and Layers

An important parameter which had to be taken into account was the data itself. Due to the lithographic process limitations such as the diffraction, a change in the mask on a given location can have an impact on the shapes of the whole design we wish to imprint. However as a matter of simplification, we considered only the ambit to be 0.5 µm, based on the OPC department's sensible experience and criteria.

Therefore that is why we chose the size of the crops to be $4\,\mu m^2$, with an overlap every 1 µm for the impact of the mask on one of those crops would barely affect the design generated by other crops with this method. In order to realise this, we developed a python script generating the right number of Design rules check or *DRC* files and reading them with Mentor's *Calibre*.

This strategy was based on a big design of $12\,mm \times 12\,mm$, for which the GDSII file took up to 20 GB. Therefore, it would've been madness to try and run the algorithm on such huge layouts, which is the reason why we had to create small crops of the design, for both the training and the inference purposes.

Another issue we faced was the parting of the GDSII files into layers, which is an easy way for designers in their additive strategy to separate functions between blocks of the design. For instance some layers may be indicative text or different materials such as metal or polycrystalline silicon. However all of these layers should not appear on the pictures we generate, for we were working on the metal only, so that we filtered the layers in the same python script as to generate several pictures, particularly a picture of the mask after OPC, and a picture of the target design. We also had to keep a layer called BBOX which allows the software to know the size of the overall design, by being the background of the picture. This was mostly important as to reconstruct the GDSII file later with the right size.

3.1.3 Design Rule Checks

As of working closely with the OPC department; we had access to several limitations due to the manufacturability of the generated mask. Those limitations were at the SRAFs level, and mostly consisted of minimal distances in design patterns. Indeed the SRAF should not be present on the final layout, thus we needed to make sure the rules were respected. Therefore we developed a DRC script allowing us to know which parts were manufacturable or not, so that we could correct our algorithm with it later.

We based our work on a script developed by the OPC team, so that we could change the criteria according to the rules we had. Our script included all the minimal distances between the cells on a layout, which we customised according to the technology we were using. Depending on the shape of the cell, distances could be different. A summary of the different parameters we had to take into account can be found in the Appendix section.

This script can be launched using Mentor's *Calibre*, and returns a list of the different problems we may have on the generated mask, which would make the manufacturing of such lithography masks impossible.

3.2 Construction

3.2.1 Data preparation

We then had to crop the main GDSII file into thousands of smaller files, being an important step was then to turn the raw data into an understandable dataset. This was done under Google's *Tensorflow* package, and took quite a while before being successful.

Indeed it required to plunge into the functions in order shuffle the data, assign it to the right label, take buffers from the dataset, so that we would not overload the memory with several thousands of images.

Due to physical limitations, we had to take an overlap between the pictures into account as previously mentioned, meaning that the *active* region on a simulated layout is only the size of two times the ambit, therefore our crops should overlap each other as we wish to reconstruct the mask after we created all the crops of the final mask. This led us to change most of the scripts as to be able to retrieve the continuity between the masks, therefore removing part of the shuffling when creating the dataset, but rather shuffling in regions on the main target.

We then had to binarise the data as to obtain black and white pictures only. For this, we created a function turning the colourful image we had into a black and white, binary picture. We used a a lightness optimisation way of doing it, which led us to having accurate black and white pictures, that is to say turning each array of 3 channels for each pixel into $(max(RGB) + min(RGB))/2$, and then imposing a threshold of 0.9 so that any value below this threshold would become a 0, and the others would be 1. The Listing 2 in the Appendix shows the lines of code we used for this, with *image_ count* being the number of images, *train_ ds* being the training set.

3.2.2 Pre-processing and computational power

An important line one can see in Listing 1 is the reshape line :

```
train_images = train_images.reshape(train_images.shape[0], 128,128, 1).astype('
    float32') #reshaping}
```

This command is crucial as to enter our algorithm with data being in the right shape for the Input Layer. This line often led to crashes when we tested with different outputs, for example with bigger images (256x256), we had to add a *maxpool* operation as to reduce the dimension of the data, or the GPU would overload thus leading to a crash. Indeed, this line creates a tensor of the right shape for the training batch, and in our case, it was more than 4 GB to load into the VRAM.

Therefore we had to optimise the execution of the algorithm. We had already optimised the size of the dataset, by storing them as GDSII and PNG files instead of raw data on the Hard Drive, however Deep Learning requires a lot of parallel computing for the model we used consisted of around 1 Billion parameters.

Here is an example on Figure 18 of a model the team developed with larger images when testing different architectures for the Encoder, where we can clearly see the required maxpool layers preventing the model from crashing, as well as the *Variational* part with the mean and standard deviation in the end of the encoder :

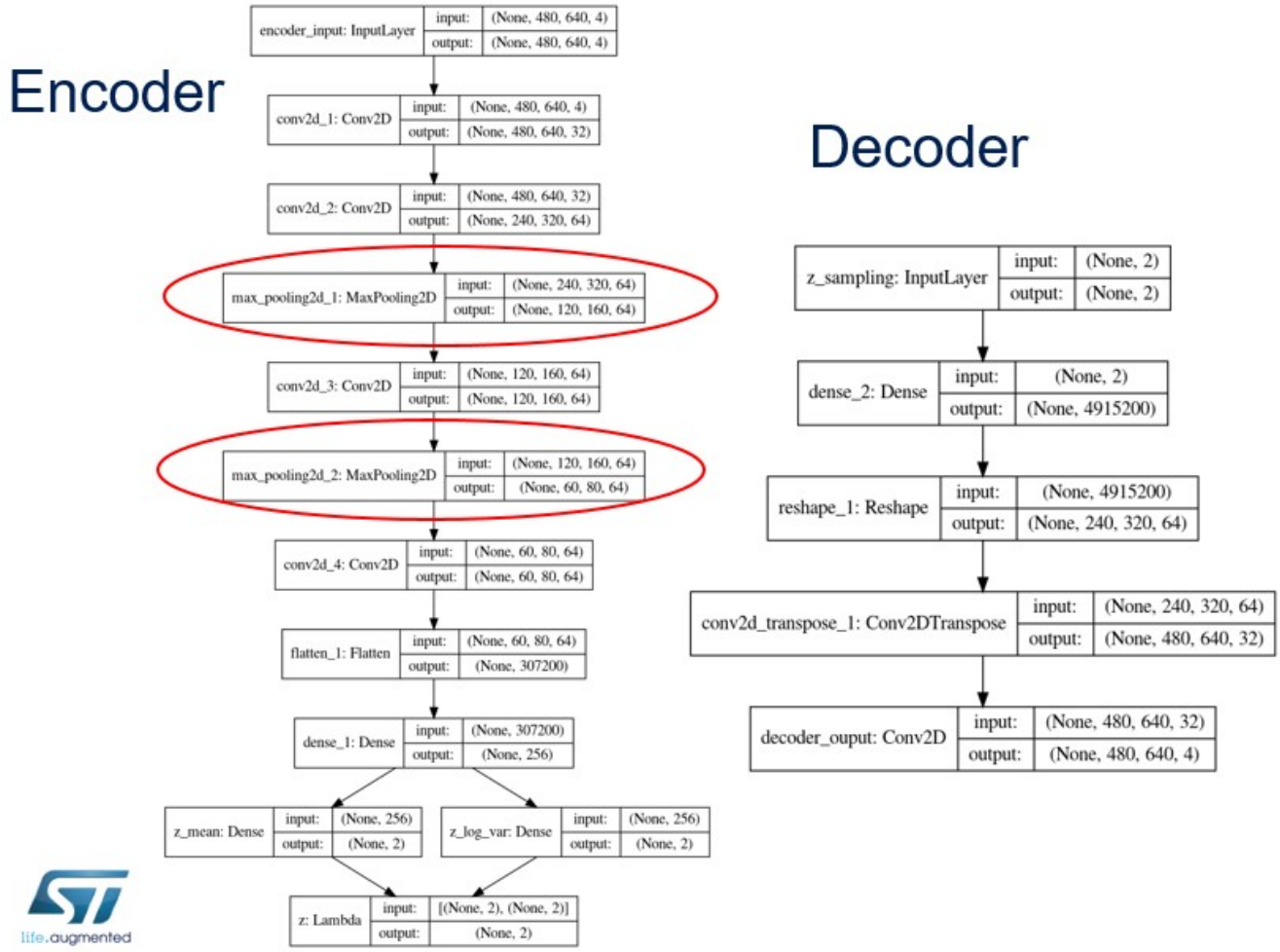


Figure 18: Variational AutoEncoder model example for 480x640 images

To run such models, we had to use a GPU, which was a Quadro P3000 running on the company's internal computing farm. In order to do this, we had to write a specific command to run our script on the computing farm, and we had to force the execution on the GPU, despite its availability being limited as many people within the company wanted to access it at the very same time. Nevertheless, once the GPU was under our control, it allowed us to **save an enormous amount of time** when executing for two reasons.

- The GPU could load the data faster within its 6GB VRAM than on the computer the team was using, and the computing farm allowed for more than 16GB of additional RAM to be used. The parallel computing using the GPU took advantage of its numerous CUDA cores, allowing for faster computing, especially as neural networks don't require difficult calculations, but mostly a lot of simple operations in parallel.

- The main bottleneck within our flow was the use of Mentor's *Calibre* software, which we had to call in order to simulate the target design using the mask we should have generated. This software could only be started on the computing farm, for it is located there, therefore it was wiser to launch everything in there to save up more space, rather than having to call for it for every training example within our algorithm. This was also the reason why the team decided to switch back to the ILT loss, for it allowed us not to require Calibre, although this was done in the very end of our internship, therefore it is detailed in the very end of this report.

To sum up the improvements we obtained by choosing to use the GPU for the Machine Learning itself, compared to using the CPU we had locally, our epoch duration decreased from more than 5 minutes, to less than 2 seconds, making the testing phase extremely efficient.

3.3 Testing and results

After understanding the issues at stake in this very project, after modelling several solutions and clarifying the whole project, then came the time of realisation and in particular the testing phase. At first it seemed pretty logical to play around with some pre-existing algorithms in order to understand the practical aspects of the realisation, and then to adapt it to our lithographic patterns. In the realisation, we mostly used the MNIST database of handwritten digits, before changing it to our layouts in the very end.

3.3.1 Building the algorithm for simple patterns

We followed and recreated a MNIST tutorial algorithm using a Variational Auto-Encoder at first, as the problematic was very similar to ours, that is to say reproduce binary patterns using a mere VAE. This VAE also has the peculiarity of classifying the data using 10 classes, each for a different digit, which is something we want to remove later as our patterns may be of any shapes and kinds.

This algorithm was made in several steps as will be faithfully described here. The code itself is public and can be found on Google Tensorflow's website under the CVAE section:

- All the necessary modules are imported, including Tensorflow, or NumPy.
- The dataset is loaded directly from the website, as 8-bit and organised arrays. Those arrays are actually multi-dimensional tensors.
- It is then reshaped as to be clear on the separation between data and labels, and binarised as to end up being binary.
- The main parameters like the batch size are initialised, and the training and testing sets are randomly defined amongst the whole dataset.
- The Variational Auto-Encoder is defined as a class containing several functions.

  The pictures being 28x28 pixels big, the Encoder has an Input layer of this size, followed by two convolution layers, the data is then flattened, which means reshaped into a smaller size, dimension-wise, and it gets through a Dense layer before being returned. This ensures the encoded data is fully linked to a number of neurons equal to the latent dimension we wish to have.

  Then the Decoder is defined as an Input layer of the latent dimension, connected with a Dense Layer. It is then reshaped into a 7x7 tensor in order to get through two transposed convolution layers making it 28x28 again. It finally gets through a third transposed convolution layer of dimension 1, as to deliver the final output.
- Several functions are then defined, such as encoding, decoding using the previous class with the calculation of the mean and standard deviation.
- The loss function is then defined, as well as the computation of gradients.
- The "main" part of the code is then written as to create a VAE, train and test it, show results and save them as pictures.

After having set the whole environment, and running this code using the company's available GPUs, it ended up working, with a loss function as shown on Figure 19.

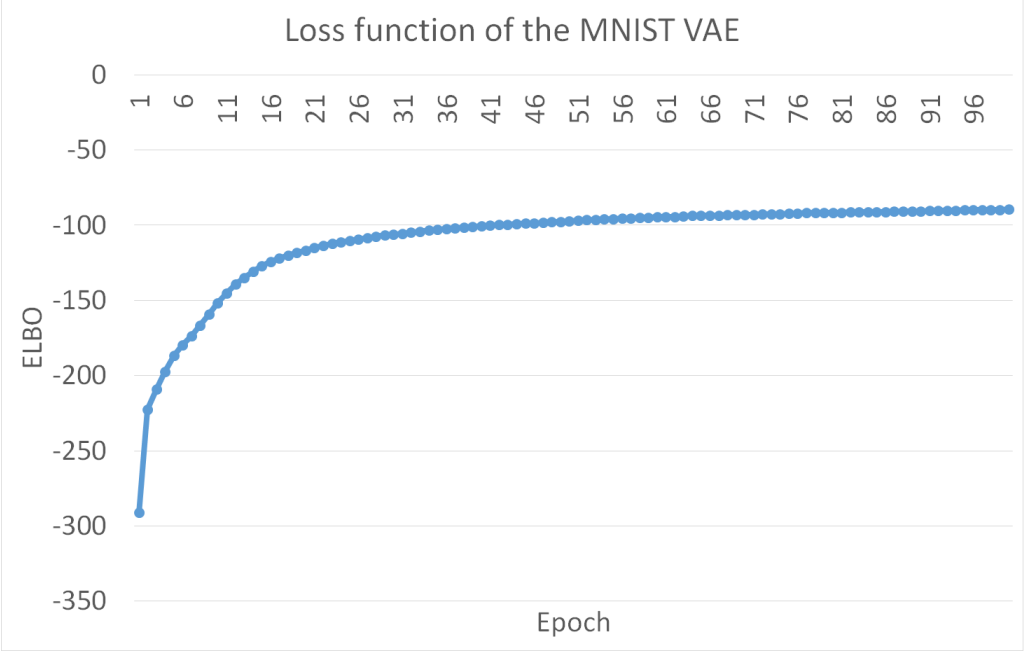
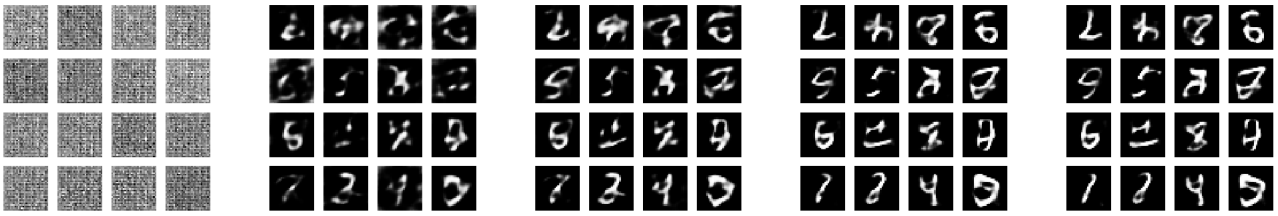


Figure 19: Evolution of the Loss Function of the VAE for MNIST digits generation

It is clear the code isn't perfect, for the convergence doesn't reach anywhere near zero but rather -90 starting from -300, however the pictures we ended up getting looked quite satisfying. Here is a short evolution of the pictures we got from random noise as input through the decoder at different epochs. With a bit more training examples, and perhaps an optimised neural structure, we may get even better recognisable digits, which is why the team is currently working on building a new structure as shown on Figure 18, as to compare with this one.



(a) Epoch 0 (b) Epoch 5 (c) Epoch 10 (d) Epoch 60 (e) Epoch 100

Figure 20: Evolution of the generated images with the MNIST VAE

After this was done, we wanted to adapt this algorithm to run with our own data being local pictures of a different and bigger dimension. We downloaded a public MNIST dataset of 15,000 128x128 pixels PNG pictures, stored them locally sorted by digit.

In order to do so, we had to implement a way of importing local pictures, treat them as to be workable, that is to say becoming a dataset in a format our algorithm could deal with. Then we had to adapt the main structure of the VAE itself, so by adding layers as an input of the encoder and as an output to the decoder. Following this we had to make it run on our GPUs, although bigger dataset and bigger networks meant more memory required and longer epochs.

3.3.2 Adaptation to our design layouts

The first difficulty was to adapt the algorithm to the size of the pictures. We had to add some lines of code, which we will further detail below, for we tried numerous customisations of the parameters to get something working in the end. We did not change the latent dimension, for the information contained in the new pictures was the same as the one in the 28x28 pixels images, that is to say a binary drawing of a pattern on a plain background. We concluded that a latent dimension of 50 neurons was enough to encode such a picture, after the tutorial, independently of its size.

Here is a picture of what happens at epoch 20 when we tried to reduce this dimension from 50 to 10 on Figure 21 as a mere test : we can see that the learning rate is quite slow and the encoding is so lossy that the decoder cannot recreate the pictures faithfully enough, albeit it makes them extremely pixelated despite the pictures being 128x128 in size:

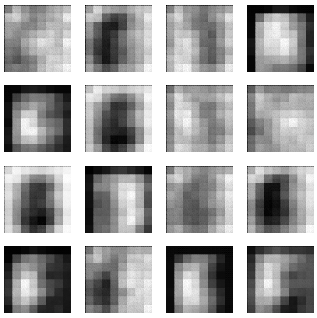


Figure 21: Image generated by the VAE with a latent dimension of 10 at epoch 20

To fix this, we got inspired by the paper [4] and [15] amongst others for the architecture, so that we added more convolution layers with larger filters, as well as more transposed convolution filters. We had the idea of setting a threshold in the last transposed convolution layer, so that we added an activation function of the form of an hyperbolic tangent. In the papers, they stated that the *tanh* function is generally preferable over a sigmoid, for its gradient is steeper and optimisation is easier with it.

In the following results, it is unnecessary to show the loss function evolution for it is sensibly the same as the one with the MNIST tutorial, except that the scale is increased. However the last result is the one with the lowest ELBO, being around -345, whereas others couldn't get closer than -850. ELBO here means evidence lower bound, and it is basically the result of the loss function, so the value the norm of which we try to minimise. It is logical to have higher ELBO for higher dimension pictures, as we increase the number of parameters, so that the error probability is higher. However it may not be visually noticeable.

At first, we tried to add many convolution layers ; here is a picture of the result with 3 more convolution layers in between, with dimension 64, 128 and 256 (also called filters) in the encoder, and two more transposed convolution layers of dimension 256 and 128.

The logic behind this approach was to adapt simply the algorithm to bigger pictures, however it didn't work much for we happened to figure out the information contained in the pictures was the same, although their size had increased. This led to overfitting as we can see on the picture here, that is to say trying to add too much information, and in the end having something trying to be too accurate, therefore looking like going through too many filters.

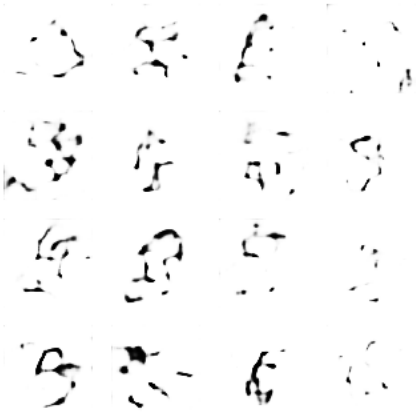


Figure 22: Image generated by the VAE with 128x128 pictures with too many filters

We can still make out the patterns of the digits, however it is not a satisfactory result, so we decided to remove some convolution layers, therefore adapting the reduction of dimensions within the encoder.

We tried to reduce the kernel dimension as well, but it led to an important loss of information, therefore we obtained images looking like this one when we changed the convolution kernel size parameter to 2 instead of 3 for all layers.

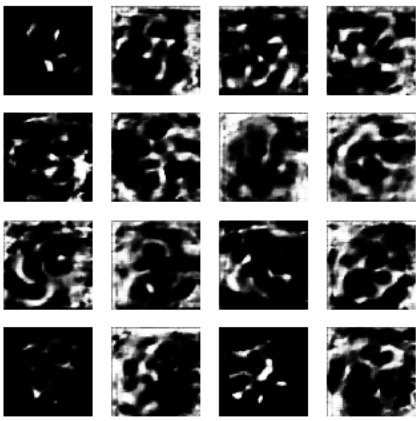


Figure 23: Image generated by the VAE with 128x128 pictures with an important loss of information

We had additional problems, for with low dimension pictures, the loss allowed to distinguish between dark patterns on blank backgrounds and opposite, however at higher dimensions, an occurrence of two different configurations led to the network being confused. This means that the decoder could not decide whether to create the patterns as white or black on the opposite background, so that in the end, it decided to have completely black or completely dark pictures.

This can be seen mathematically as trying to globally reduce two loss functions, without ever converging and never obtaining satisfying results. Here is an example of the first epoch of our algorithm, in which we also reduced the dataset as to only have the digits 3 and 6 available as to decrease the training time as well.

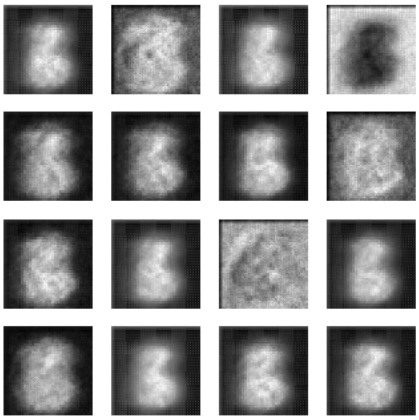


Figure 24: Image generated by the VAE with 128x128 pictures in the first epoch

Here we can clearly see the top right image being a dark digit on a blank background, leading to bad results in the end, such as those on Figure 23.

To fix this problem, we forced the algorithm to restart if any occurrence like this appeared at the very first epoch, as a way to force the convergence of the model. In the end, we finally obtained an almost working VAE where we can see it finally converges to something which looks like handwritten digits. The parameters we used were only 4 convolution and 3 transposed convolution layers, with a tanh activation function in the end of the Dense layer in the decoder. We didn't change the dimension of the main layers, except the input and output layers to take larger pictures and return larger pictures as well.

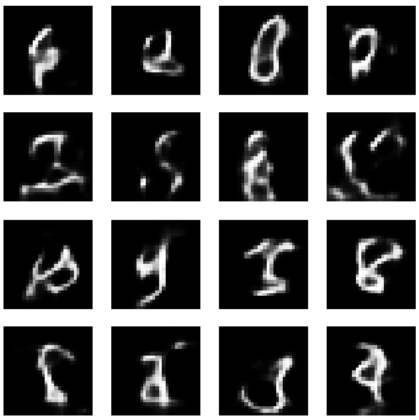


Figure 25: Image generated by the VAE with 128x128 pictures with an ELBO of -345

We didn't have the time to add the discriminator, which we had written in Python, for it required to completely change the training steps. Indeed the generator and the discriminator need to be trained alternatively, and the training must be carefully set as not to have one of the network training faster than the other, thus losing balance. Therefore, as an effort to to have something running fast we did not add it, all the more so as the whole team wondered if the discriminator would add anything relevant. Indeed, the VAE by itself with a physics-based loss function could work without a discriminator, so we decided not to add one at first and see it as a future improvement if needed.

After we had something somehow-working, it was important to set up a complete chain of commands, so that other interns and the team could keep working on improving our work.

In the end, we managed to have a fully functional chain of commands, which could do the following from a simple huge OPC pre-processed GDSII file:

- Import the GDSII file and crop it into smaller GDSII files, taking physics-based overlap into account.
- Convert each crop into a database object, including having a picture per layer in the GDSII file. Here is a Figure showing an example of our work based on a crop we generated. As we can see, all layers are stored within the same GDSII file and we can extract them as PNG files for training.

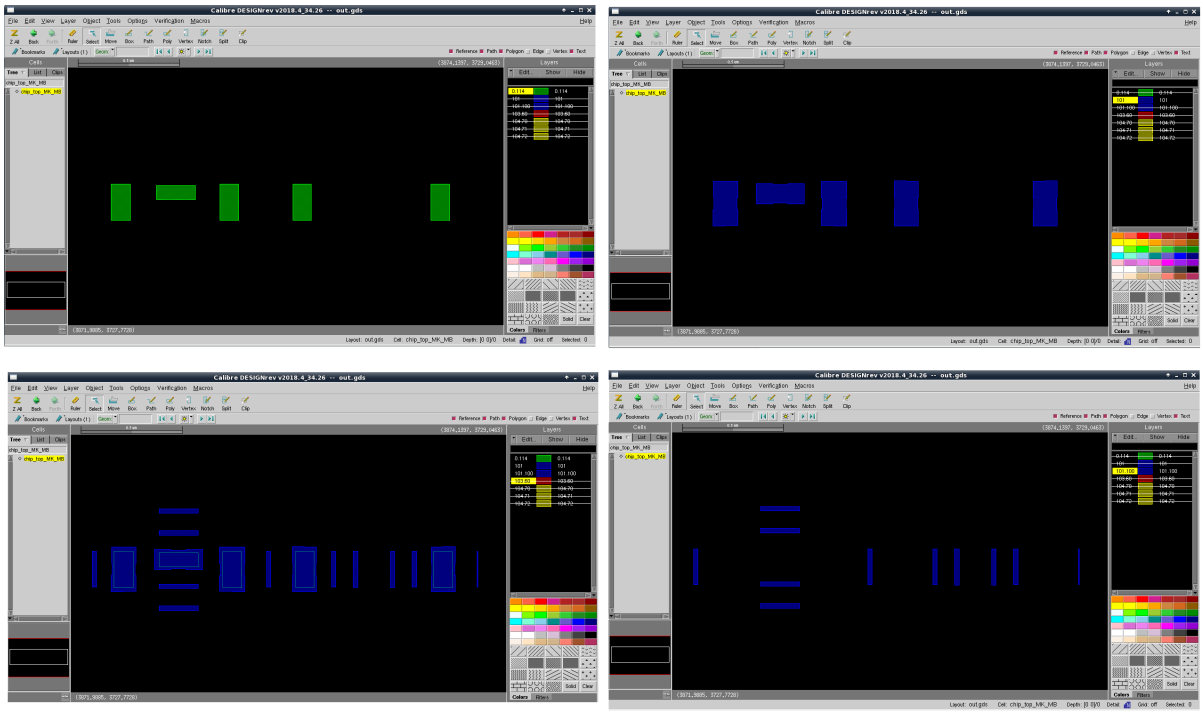


Figure 26: Different layers for the same GDSII file : target, target with OPC, SRAF layer and OPC+SRAF for simulation

- Simulate the real target based on a crop (for the lithographic process cannot physically produce the right design) and add it to the objects.

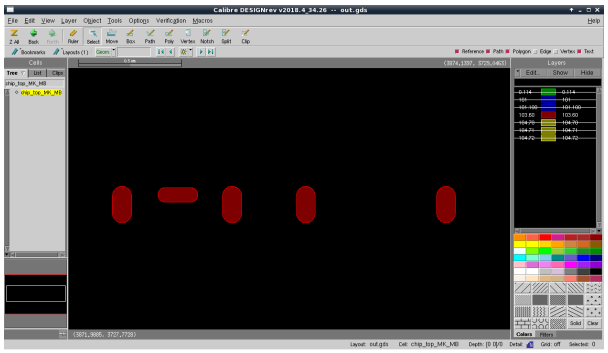


Figure 27: Rounded target simulation based on the target

- Import those objects into a Machine Learning preparation algorithm, by pre-processing the data and turning it into a dataset.
- Run a GAN made with a VAE on those objects and generate images of the mask with SRAFs. The algorithm can run on the dedicated computing servers of the company, having either a GPU or CPUs for us to test with.
- Have different loss functions to choose from a physics-based loss with a simple model, or a pixel-based loss using proprietary software.
- Convert the generated image back into a GDSII file, as to differentiate between the patterns and the SRAFs.
- Simulate the photolithography process with the newly generated mask with a variable and customisable focus value, and check the mask manufacturability if necessary, as to know if what we generated is physically relevant.

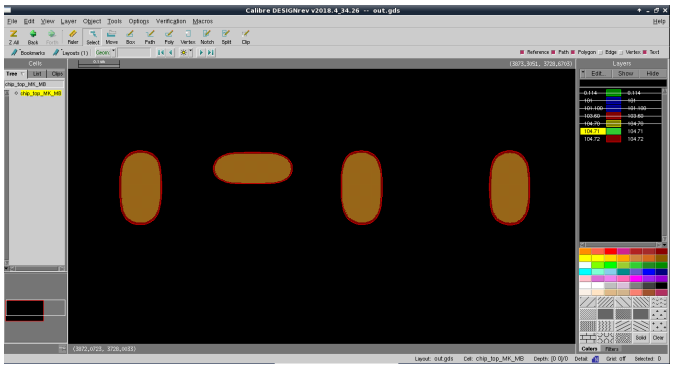


Figure 28: Different defocus values simulated by the simulator, with each colour corresponding to a different focus value

Actually the VAE-GAN which ran in the end was not completely finished for it could not really adapt to any size, as we had only tested with 128x128 pictures, and the whole architecture of it is dependent on its size, as aforementioned. The results first run with 200x200 images can be seen in The Improvements section as Figure 31.

Loss Function for Physics Modelling

Another code we realised, which was an important work to take into account, is the loss function. We created a Python script allowing the calculation of the loss function we modelled earlier for the ILT.

As we can see on Listing 1, we created a function *physicsloss* calculating both the loss function and its gradient, however this was not used as of yet as we previously mentioned Mentor's *Calibre* was more suitable on the short run.

However as we will see in the Improvements section, this work is a milestone for what is going to happen next, because it suggests an alternative way of modelling lithography without being limited by proprietary software and external system calls. We managed to test the code on simple images, as we didn't have the dataset ready when we finished writing it, and we changed the focus of the project.

We took a Numerical Aperture of 1.35 with a wavelength of 193 nm, as transmitted by the OPC department, we took the parameters for the sigmoid being a threshold of 0.7 as the OPC department told me it was realistic, and a steepness of 10 because we were then able to differentiate between most values and it was quite steep, although not looking like a Heaviside step function. We also wanted to map the Bessel cardinal coefficients to check the impact of the ambit, therefore we plotted a figure of the absolute value of the coefficients on space with a threshold at 1/150 of the maximum of intensity, and obtained Figure 29.

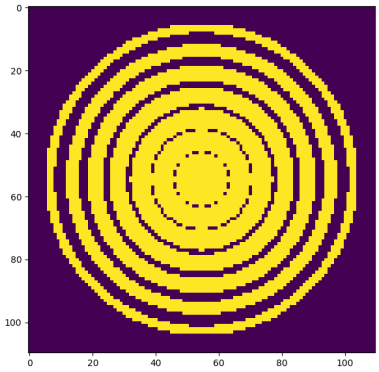
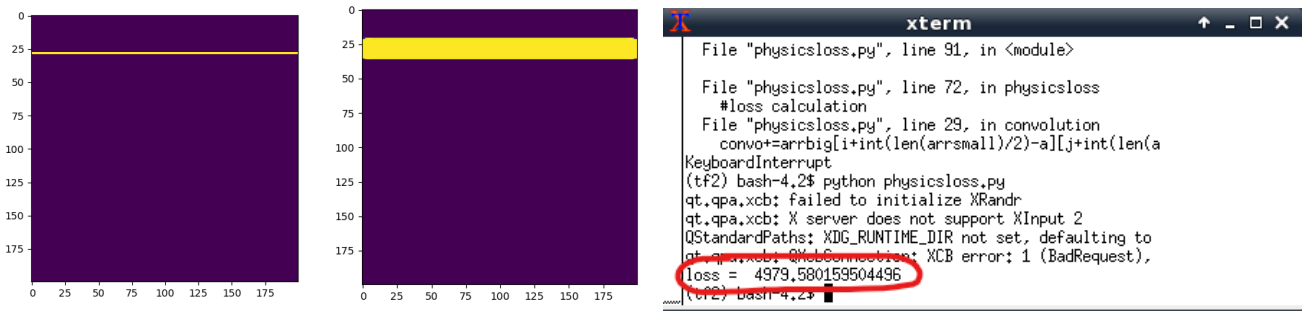


Figure 29: Plot of the impact of each point in space on the central point with a 1/150 threshold

We can see on this Figure that an ambit of around 500nm is acceptable as each pixel represents 10 nm as I've set them, and if we consider as negligible the impact of objects with coefficients below 1/150 of the maximum intensity. In reality, the central circle contains 86% of the whole intensity, so that we can reduce the size of the kernel to the two primary circles to take two orders of diffraction into account. It also confirms our implementation of Hopkin's theory [5], with those coefficient having an Airy pattern shape, which once convolved with our patterns indeed gives us diffraction patterns. This Figure is a binarised version of our convolution kernel.



(a) Mask example (b) Resist simulation $\mathbf{Z}(\mathbf{I})$ (c) Loss function calculation

Figure 30: Physics-based simulation of the ILT within the loss function : 1 pixel = 10 nm

After running our code on small pictures of 500 nm x 500 nm, we obtained a loss function of around 2500 for mask patterns being the exact opposite of the target, whereas it fell to less than 1 with small images being exactly the same. On bigger pictures of 4 microns squared being all white or all dark, the loss was of 25000 for completely opposite pictures. However when we tried to add patterns, we could see the diffraction patterns when using the aforementioned wavelength or even bigger ones. On the Figure 30 we can see the diffraction effect, and the target was the same as the mask therefore a loss of 4979, a high number as we keep it squared. We can now see that we managed to shortcut the use of Mentor's *Calibre*, by developing our own model which could then be optimised using the VAE-GAN.

We then managed to include our loss function in the team's VAE, where it showed significant improvements in the running time. However the structure of the VAE being still messy, we didn't manage to observe significant results of robust masks being created.

In the very last week, we ran simulations of more patterns to validate our model and show the effects of diffraction on bigger and more interesting images, however we didn't have the time to include them here, so that we will add them to the presentation. These are things we can develop in the next section about the improvements which are to be made.

3.4 Improvements

After realising the first step of such a project some serious improvements can come to our mind Create own database with progressive difficult of patterns for training.

- The first one may be within the structure of the VAE-GAN : First of all, we didn't have time to add the discriminator, however it could be a real asset as to make the network converge better and faster, although it would require to review the training as to obtain a balance between the two blocks : the generator (VAE) and the discriminator. Our network started to converge in the very end of the internship (the last week), and here is a picture of the first images we managed to obtain, using our strategy of VAE with slightly bigger pictures (200x200 pixels for $4\,\mu m^2$).

Figure 31: Input (top), output (bottom) of the first complete VAE run at epoch 2

Although those images are taken at the 2^{nd} epoch, which is very early, we can see that the features extraction of the encoder and the stochastic reconstruction of the decoder are quite faithful to the main features, thus the proposed improvements.

- Saving space in the database could another improvement axis. Indeed we stored GDSII files and their associated generated PNGs per layer, however we could probably encode the pictures as a single PNG file with each layer on a different channel bits. However this would limit us with 4 layers as a maximum, for PNG pictures are encoded with 4 colour channels. This would also require us to change our way of extracting the data into the algorithm, especially when forming the dataset. We could also use the structure which was created by the previous intern, as to have a real database with objects easily recognisable and accessible, and the GDSII compression.

- Another improvement would be to take advantage of the company's budget in high-end components as to run the algorithms with Scalable Link Interface GPUs, as to make calculations faster. Indeed the company plans on investing on new GPUs in the computing farm being Nvidia Quadro RTX P6000 graphics cards, allowing for far more powerful calculation power available than the P3000 we used.

- That would also be true for the loss function we calculated, which takes around 5 seconds to be computed, given the size of convolution kernels on the pictures. Including the ILT equations we developed could be more efficient by converting the whole code into Numpy arrays, rather than simple Python lists. This wasn't done yet as our loss was not extensively used at first, and is still at the stage of an experiment.

Conclusion

The objective of this internship was to develop an industrial process flow as to optimise the photomask generation.

After an introduction chapter, the theoretical background was presented in the second chapter, photolithography challenges, parameters, as well as Machine Learning knowledge up to advanced generative algorithms were presented.

In the third chapter, the model we used for two sub-problematics is explained, as well as how we decided to implement it.

In the fourth chapter, we considered the whole process flow from the very beginning with the GDSII file till the end with the Machine Learning algorithm. All the steps in between we created were detailed, taking into account the Lithography limitations, the Machine Learning requirements, the results of the testing of such algorithms on simple datasets, their adaptation to our problematic, as well as the implementation of the physics-based loss function.

In the frame of this work, we developed a reproducible and well-detailed process flow which takes layout designs as an Input, and outputs a photomask generated using both AI and physics equations.

This result although not perfect nor completely finished as of yet and still subject to improvements encourages me to work in similar fields later, with the combination of microelectronics-related domains and AI-related tools. This would probably lead me to take example on my supervisor P. Urard who switched from a technical job to project management but still being fully involved on high-tech research projects.

Appendixes

Project organisation and unfolding

The research group was formed of Mr. Urard as the lead Director, and a handful of interns working on different subjects. During this project, I realised in the beginning of December that it would be difficult to have a running flow in the end of my internship because of a lack of time. Indeed, I was the only one working on this project for 4 months, and before I arrived, no real idea of the flow, nor scripts which could help me had been done.

I also had to set up the whole UNIX environment to run different software such as Calibre or TensorFlow, which is a long procedure of more than 1 week within the company. Then I had to create by myself a Variational Auto Encoder, turn it into a GAN, and I had no knowledge at first of those tools. So that, I decided to try my best to have a whole flow running, developing DRC scripts to process the data, I also suggested the whole flow, worked on the theoretical part, and implemented it.

Then in mid-December, the project changed to the defocus robustness problematic, with the new strategy being to add SRAF, as it seemed simpler and a better path for the company. I therefore had to create new scripts, focus on new problems (such as the one of layers), and when a second intern and a colleague arrived in mid-February, I could delegate some of the tasks, especially with the DRC scripts, as to focus on more testing and assembling the flow with them.

To conclude, this project really had two phases, both of them blending together in the very end of my internship, and it required multidisciplinary skills to be achieved, although a lot of improvements are still to be done.

Figure 32: GANTT diagram of the project

Glossary

- AI : Artificial Intelligence (generally refers to Deep Learning applications).
- CUDA : Compute Unified Device Architecture, used to take advantage of multiple cores on a Graphics Processing Unit.
- DRC : Design Rules Check : a set of rules for a desifgn to be manufacturable AND a file format to run script within Mentor's *Calibre*.
- DUV : Deep Ultra Violet (around 13.5 nm).
- EDA : Electronic design automation.
- ELBO : Evidence lower bound (value of the loss function at a given epoch).
- GAN : Generative Adversarial Network : Deep Learning model to generate data, detailed in the first Chapter.
- GDSII : A vectorial file format for layouts.
- GPU : Graphics Processing Unit.
- ILT : Inverse Lithography Techniques, the goal being to generate a "perfect" mask given a design.
- MNIST : Modified National Institute of Standards and Technology, which is a large database of handwritten digits that is commonly used for training various image processing systems.
- NA : Numerical Aperture, a dimensionless number that characterises the range of angles over which the system can accept or emit light.
- OPC : Optical Proximity Correction, which is a set of methods trying to simplify ILT by iterative rules.
- PNG : Portable Network Graphics, an image file format with a lossless compression.
- PSM : Phase-shift mask, to improve the contrast of the on-resist design after exposure.
- PSF : Point Spread Function, which describes the response of an imaging system to a point source or point object.
- RAM : Random Access Memory, where is stored our data when running the algorithm.
- SRAF: Sub-Resolution Assist Features, or added "bars" on the mask to improve the contrast and a focus robustness to the mask.
- SVG : Scalable Vector Graphics, a vectorial image format with a lossless compression.
- SVM : Support Vector Machine, an algorithm to extract features of data and find hyperplanes separating those features. It is detailed in the Chapter 1, and allows for data generation.
- VAE : Variational Auto Encoder, a Stochastic Deep Learning model to encode and decode data, allowing for features extraction and generation of data. It is described in the Chapter 1.

Cost analysis

The achievement of my master thesis relies on several costs:

- My salary and a percentage of the salary of all the engineers working partially on the project should be considered.

- The software licenses (Calibre, Microsoft Office, IEEE documents): This cost also is not easy to evaluate as the same licenses are used by several engineers, and the prices are not always given within the company.

- The special hardware to which I had access such as the computer, the GPU, and the computing farm.

- The advantages over the food and other activities funded by the Company's Committee.

Python code samples

```
# -*- coding: utf-8 -*-
import numpy as np ; import sys ; np.set_printoptions(threshold=sys.maxsize) ; import math
#bessel function of order 1
from scipy.special import jv ; import matplotlib.pyplot as plt  ; order = 1 ; resolution = 10*10**-9
def Besselcardinal(order,x):
  if x==0:
    return 0.5
  return 2*jv(order, x)/(x)

def convolution(arrbig,arrsmall,i,j):
#definition of the convolution operation with an image and a smaller kernel
  convo=0.0
  for a in range(0,len(arrsmall)):
    for b in range (0, len(arrsmall)):
      if i+int(len(arrsmall)/2)-a>=0  and j+int(len(arrsmall)/2)-b>=0 and (i+int(len(arrsmall)/2)-a)< len (arrbig) and (j+int(len(arrsmall)/2)-b)< len (arrbig):
        convo+=arrbig[i+int(len(arrsmall)/2)-a][j+int(len(arrsmall)/2)-b]*arrsmall[a][b]
  return convo

def physicsloss(a, lambd, thresh, na, mask, target,resolution):
    #initialisation
  Z= [[0 for j in range(len(mask))]  for i in range(len(mask))] ; fl= [[0 for j in range(len(mask))]  for i in range(len(mask))] ; convolved= [[0 for j in range(len(mask))]  for i in range(len(mask))] ; arr2= [[0 for j in range(len(mask))]  for i in range(len(mask))] ; grad= [[0 for j in range(len(mask))]  for i in range(len(mask))] ; I = [[0 for j in range(len(mask))]  for i in range(len(mask))]
  #size of the convolution kernel, fixed at 15 for the example
  #lengh = int(len(mask)/4)*2 ; #widh = int(len(mask)/4)*2
  lengh = 15 ; widh = 15
  h=[[0 for j in range(lengh)]  for i in range(widh)]
  #common variable
  var1 = 2*math.pi*na/lambd
    #We loop on all the elements of bessel cardinal kernel to calculate it
  for i in range(0,lengh):
    for j in range(0,widh):
  #distance from center of mask
      r=math.sqrt((i-(lengh-1)/2)**2 +(j-(widh-1)/2)**2)*resolution
  #bessel function + point spread
      h[i][j]=Besselcardinal(1,r*var1)

  for i in range(0,len(mask)):
    for j in range(0,len(mask[i])):
  #convolution between h and mask
      convolved[i][j] = convolution(mask,h,i,j)
  #light intensity as defined
      I[i][j] = abs(convolved[i][j])**2
  #resist model added to the intensity
      Z[i][j]= 1/(1 + np.exp(-a*(I[i][j]-thresh)))
  #l2 norm
```

```
      fl[i][j]= (target[i][j] - Z[i][j])**2
  #useful for calculation of gradient
      arr2[i][j]= (target[i][j] - Z[i][j]) * Z[i][j] *(1 - Z[i][j]) * convolved
    [i][j]
  for i in range(0,len(mask)):
    for j in range(0,len(mask[i])):
  #gradient matrix calculation
      grad[i][j]=-2*a*convolution(arr2,np.rot90(np.array(h),2).tolist(),i,j)
  #loss calculation
  loss = np.sum(np.array(fl))
  return (loss)

#testing and setting parameters
s=50 ; a, lambd, thresh, na, mask, target = 10, (193*10**-9), 0.8, 1.35, [[0]*s
   ]*s,[[0]*s]*s

print("loss =",physicsloss(a, lambd, thresh, na, mask, target,resolution))

#print the map of Bessel cardinal coefficients

def besselmap(resolution,size,na,lambd, thr):
#function to show the map of Bessel cardinal coefficients
  h =[[0 for j in range(size)]  for i in range(size)]
  var1 = 2*math.pi*na/lambd
  for i in range(0,len(h)):
    for j in range(0,len(h[i])):
      r = math.sqrt((i-(len(h)-1)/2)**2 +(j-(len(h)-1)/2)**2)*resolution
      h[i][j]= int((1+abs(Besselcardinal(1,r*var1))-thr))
  return np.array(h)

print("\n\nbessel map =");
plt.imshow(besselmap(10*10**-9,110,na,lambd, 1/150)); plt.show()
```

Listing 1: Python code for model-based ILT

```
def prepare_for_training(ds, cache=True,
shuffle_buffer_size=1000):
    if cache:
        if isinstance(cache, str):
            ds = ds.cache(cache)
        else:
            ds = ds.cache()
    ds = ds.shuffle(buffer_size=shuffle_buffer_size)
    ds = ds.repeat()
    ds = ds.batch(image_count)
    ds = ds.prefetch(buffer_size=AUTOTUNE)
    return ds

train_ds = prepare_for_training(labeled_ds)
image_batch, label_batch = next(iter(train_ds))
train_images = image_batch.numpy()
train_images=(train_images.max(axis=3)+train_images.min(axis=3)
   )/2 #GreyScale Conversion
train_images = train_images.reshape(train_images.shape[0], 128,
128, 1).astype('float32') #reshaping

#Binarisation
train_images[train_images >= .9] = 1.
train_images[train_images < .9] = 0.

```

Listing 2: Python code for pre-processing

Specific details

Convolution

Here is an illustration of how the convolution (or cross correlation) in different part of the project works, be it in the physics-based loss function, or inside convolution layers.

The filter is also called kernel, and it is the small array which allows the convolution to take place and therefore emphasise (or not) the surroundings of the *source pixel*, depending on the kernel construction.

In the case of our physics-based model, we chose a 15x15 kernel with the coefficients of the Bessel cardinal function as defined in the body of the report.

We can also mathematically write it as :

$$\mathbf{R} \circledast \mathbf{k}(i,j) = \sum_{a=1}^{n} \sum_{b=1}^{n} \mathbf{R}(i - a + \left\lfloor \frac{n}{2} \right\rfloor, j - b + \left\lfloor \frac{n}{2} \right\rfloor) \times k(a,b), \tag{9}$$

with $\mathbf{R}$ our initial image, $\mathbf{k}$ our kernel of dimension $n \times n$ and (i,j) the coordinate of the pixel we want to compute (the source pixel).

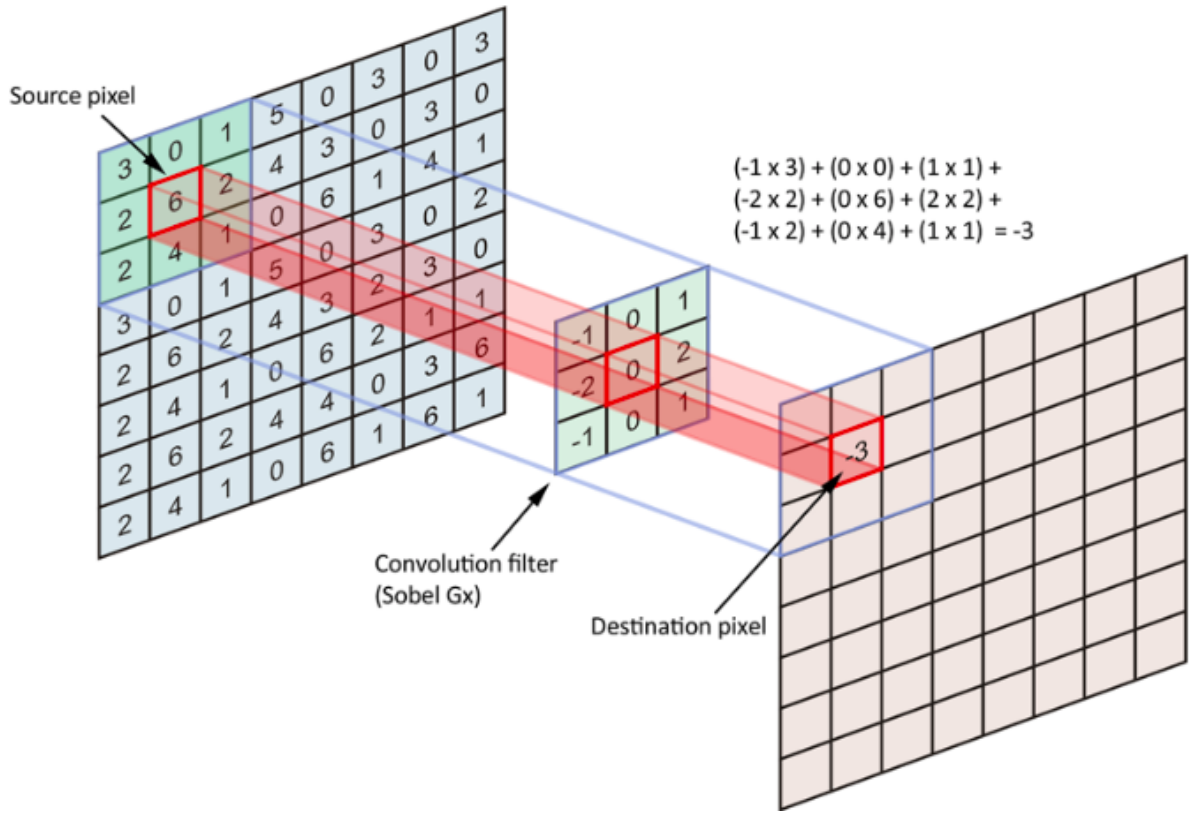


Figure 33: Convolution Layer summary

Manufacturability Rules (DRC)

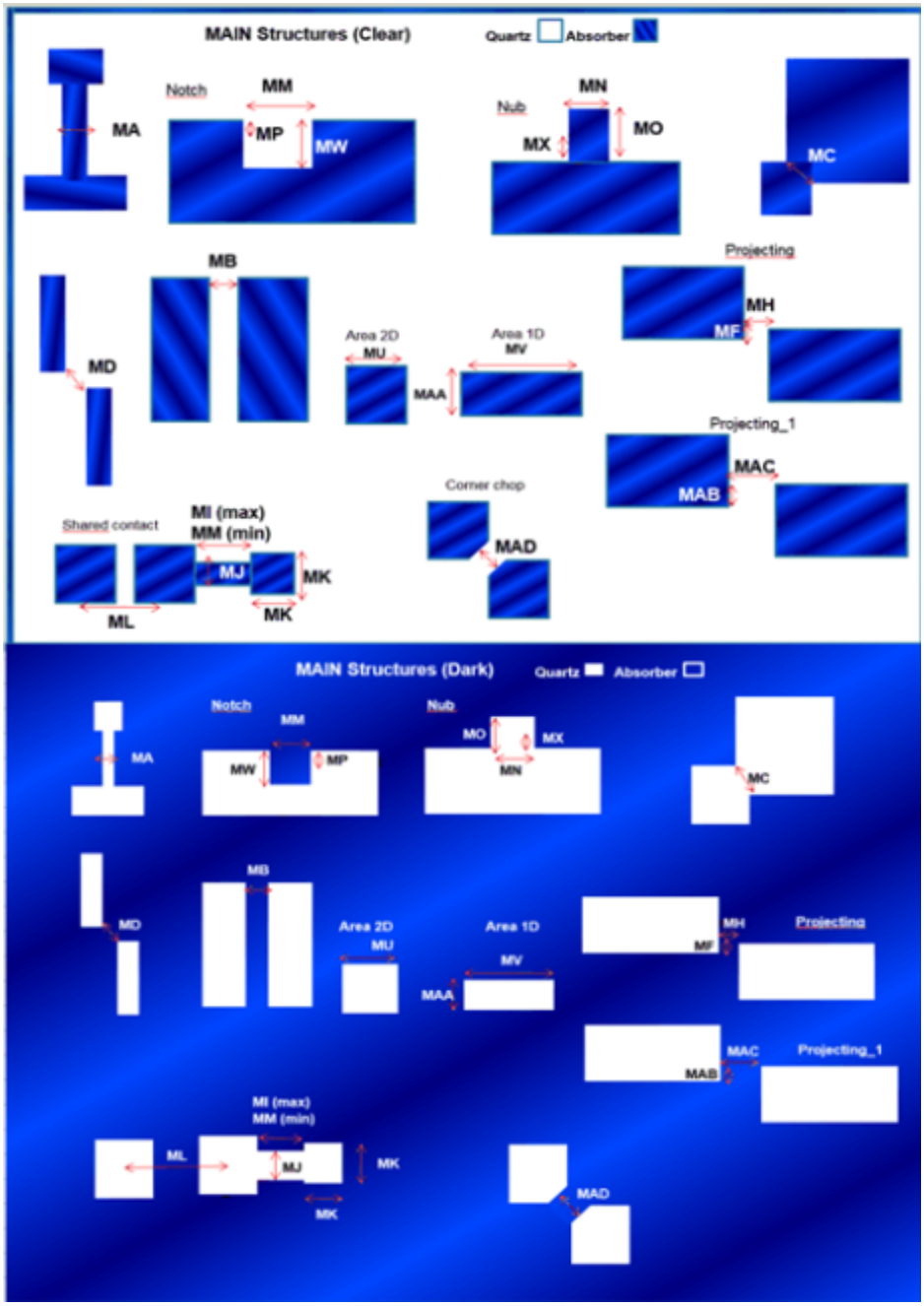


Figure 34: Manufacturability rules 1

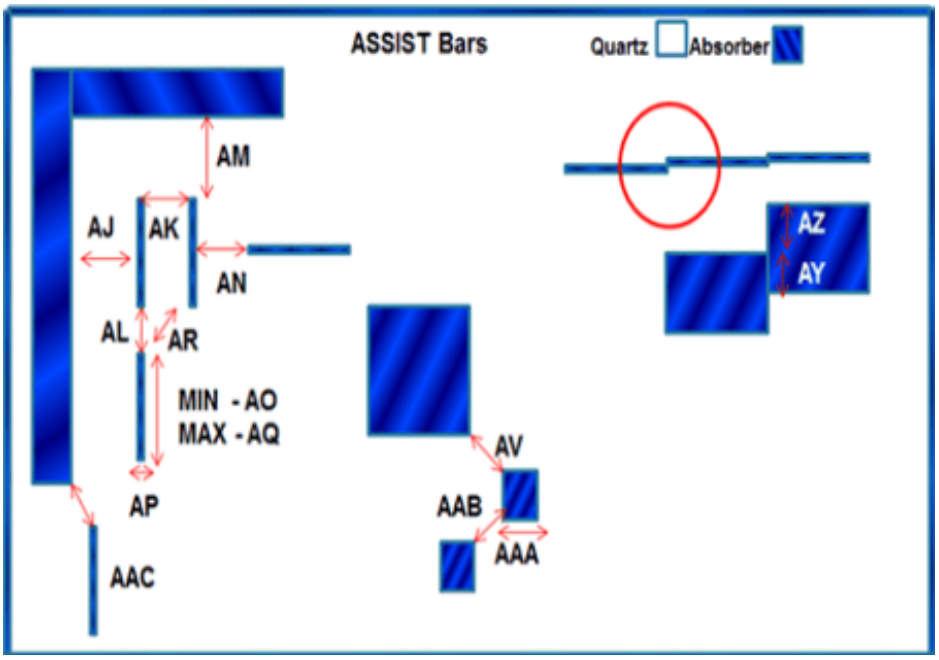


Figure 35: Manufacturability rules 2

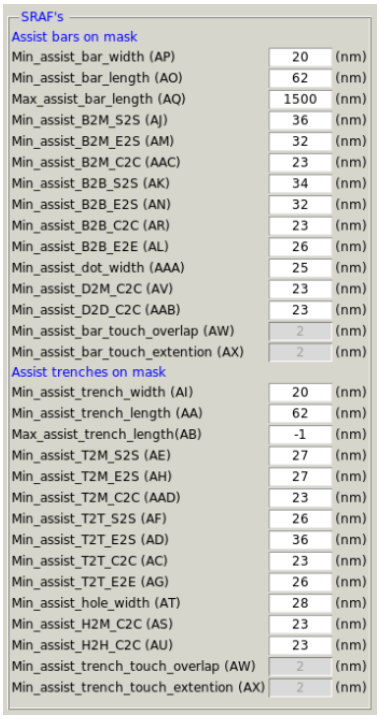


Figure 36: Manufacturability rules 3

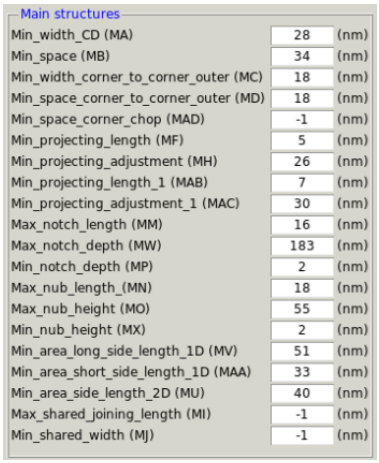


Figure 37: Manufacturability rules 4

DRC code sample for manufacturability check

Here is an extract of the code, and the parameters we had to manually set up in order to be able to check if a given mask is manufacturable. The complete code for this is exactly 1115 lines of commands, so that we will only give the definition of some of variables here and an example of a zone check. This is only useful to show what a DRC script looks like and the work we had to deal with to remotely control Mentor's *Calibre*.

```
//  ###################################################################
//  ######                  VARIABLE DEFINITION                  #####
//  ###################################################################
// script parameters
//###################################################################
VARIABLE precision_gds 1000
VARIABLE dbu 1.0 / precision_gds
VARIABLE number 29

// MDRC Variables
//###################################################################
VARIABLE Min_width_CD_MDRC                                          0.048
VARIABLE Min_space_MDRC                                             0.03
VARIABLE Min_width_corner_to_corner_outer_MDRC                      0.036
VARIABLE Min_space_corner_to_corner_outer_MDRC                      0.022
VARIABLE Min_projecting_length_MDRC                                 0.005
VARIABLE Min_projecting_adjustment_MDRC                             0.022
VARIABLE Min_projecting_length_1_MDRC                               0.068
VARIABLE Min_projecting_adjustment_1_MDRC                           0.025
VARIABLE Min_projecting_width_length_MDRC                           0.032
VARIABLE Min_projecting_width_adjustment_MDRC                       0.036
VARIABLE Min_projecting_width_length_1_MDRC                         0.056
VARIABLE Min_projecting_width_adjustment_1_MDRC                     0.042
VARIABLE Max_shared_joining_length_MDRC                             -0.001

[...]

// ###################################################################
// #####//    5.1 MIN_SPACE_CHECK_ZONE_DEF
// ###################################################################
MIN_SPACE_MDRC                          = EXTERNAL MDRC_CHECK < Min_space_MDRC
   OPPOSITE PROJECTING >  0.003
MIN_SPACE_REGION_MDRC                   = EXTERNAL MDRC_CHECK < Min_space_MDRC
   OPPOSITE PROJECTING >  0.003 REGION
//Region for defect view
```

References

[1] Daniel Abrams and Linyong Pang. Fast inverse lithography technology. *Proc SPIE*, 6154, 03 2006.

[2] Siu Kai Choy, Ningning Jia, Chong-Sze Tong, Man-Lai Tang, and Edmund Lam. A robust computational algorithm for inverse photomask synthesis in optical projection lithography. *SIAM Journal on Imaging Sciences [electronic only]*, 5, 01 2012.

[3] Daphne Cornelisse. An intuitive guide to convolutional neural networks, 2018.

[4] Partha Ghosh, Mehdi S. M. Sajjadi, Antonio Vergari, Michael Black, and Bernhard Schölkopf. From variational to deterministic autoencoders. 03 2019.

[5] H. H. Hopkins. The concept of partial coherence in optics. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 208(1093):263–277, 1951.

[6] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. 04 2018.

[7] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP:1–1, 01 2020.

[8] Diederik Kingma and Max Welling. Auto-encoding variational bayes. 12 2014.

[9] Yibo Lin, Mohamed Baker Alawieh, Wei Ye, and David Pan. Machine learning for yield learning and optimization. 11 2018.

[10] Xu Ma, Qile Zhao, Hao Zhang, Zhiqiang Wang, and Gonzalo Arce. Model-driven convolution neural network for inverse lithography. *Optics Express*, 26:32565, 12 2018.

[11] Rance Rodrigues, Aswin Sreedhar, and Sandip Kundu. Optical lithography simulation using wavelet transform. *Proceedings - IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 427 – 432, 11 2009.

[12] Hao Tang, Dan Xu, Hong Liu, and Nicu Sebe. Asymmetric generative adversarial networks for image-to-image translation. 12 2019.

[13] H.Y. Yang, Shuhe Li, Zihao Deng, Yuzhe Ma, Bei Yu, and Evangeline Young. Gan-opc: Mask optimization with lithography-guided generative adversarial nets. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, PP:1–1, 09 2019.

[14] Yujie Zhang and Wenjing Ye. Deep learning based inverse method for layout design. 2018.

[15] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.

Optimisation of Photolithography at STMicroelectronics

Emilien Chevallier

Nanotech student

Deep Learning

High-Tech Engineering

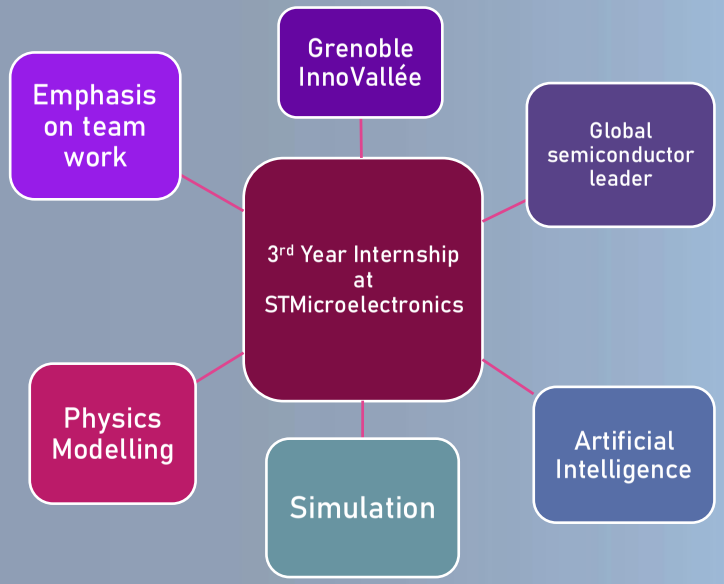
Passionate about Dynamic Research Environments

Internship scope:

- Create and test Deep Learning algorithms to improve manufacturing processes
- Develop a physics-based model for AI.
- Improve the current process flow
- Perform simulations

What interests me?

- Manufacturing of Nanotechnologies
- Applications of AI
- Multidisciplinary Research groups
- Future PhD ?



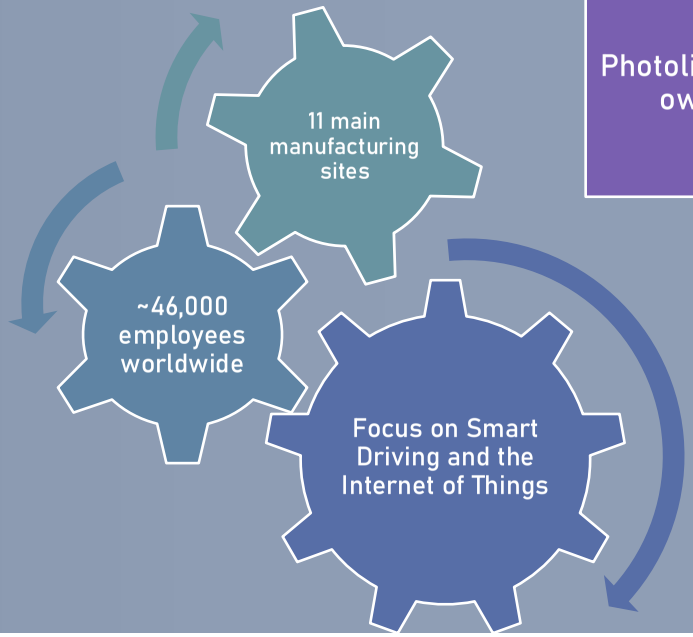
life.augmented

+33 652110885
https://sg.linkedin.com/in/emilien-chevallier

STMicroelectronics

Artificial Intelligence R&D	Bipolar CMOS DMOS	Vertical Intelligent Power technology	Phase-Change Memory technology
• General purpose microcontrollers • AI-specific Hardware and Software IP	• High speed and gain • High-frequency analog sections	• Vertical power transistors • Control and Driving circuits • Protection circuits	• Increase embedded memory sizes • Combined with FD-SOI and high-end processes

STMicroelectronics ' site in Crolles



During this internship, I...

- Understood the environment for advanced manufacturing processes
- Developed a Deep Learning model for physics optimisation
- Simulated Photolithography with my own Python code
- Created and tested a complete process flow for the creation of Photomasks
- Learned a lot about multinational companies and the work in dynamic research groups

+33 652110885
https://sg.linkedin.com/in/emilien-chevallier