Lithographic hotspot detection using Auto-ML: Simulated Annealing and Genetic Algorithms

David Villarreal[1], Ziwei Liu[2]

[#]*Electrical and Computer Engineering, University of Texas at Austin*

[1]dlv554@utexas.edu

[2]zliu28@utexas.edu

Abstract— **In this paper we describe the implementation of automated forms of machine learning model development to address the issue of lithographic hotspot detection with deep learning. We implemented simulated annealing and genetic algorithms and got results on par with state of the art neural networks on ICCAD 1 and ICCAD 2. We also leveraged downsampling and pruning on 3 different neural network model with ICCAD1 to ICCAD5**

Keywords— **Machine Learning, Auto-ML, Simulated Annealing, Genetic Algorithms, Downsampling, Structured Pruning, Neural Networks**

I. Introduction

The semiconductor industry is based on an ever decreasing feature size. Today, we push forward the boundaries of manufacturing so much that the technology node in 2020 is 3 nm by companies such as TSMC. For this, very sophisticated hardware is needed such as EUV lithography, and entire departments within semiconductor companies are devoted to simulating lithography and making such small features possible. However, the risk of hotspots, which are errors in the lithographic process, such as unwanted shorts and opens, becomes more problematic and difficult to understand in smaller nodes.

Traditionally, hotspots are dealt with using lithographic simulations, built within frameworks such as Calibre; these software solve the problem with analytical approaches such as Fourier Optics, etc. Here, we approach the problem with machine learning, which has been done extensively in the past, most notably by [1] at CUHK. The accuracy is pretty much established. Neural networks can effectively predict hotspots. However, these algorithms are currently perceived as black boxes, and we have no insights as to how they operate and make their own predictions. In an effort to make the process more scientific, we implemented ideas from the field of Auto-ML, whose essential premise is to automate the creation of neural network architectures.

Therefore, we implement simulated annealing and genetic algorithms, both of which are heuristic methods well suited for high dimensional problems, to address the issue of how to construct the ideal neural network for the problem of lithographic hotspot detection. We find that both algorithms can find a neural network architecture equally competent as that of CUHK, however because we have used Auto-ML to find this architecture, we argue that we do a better job of having an explainable model. We can say how we arrived at our given architecture, whereas CUHK cannot really say that. Manual manipulation and navigating through the space of possible architectures is highly inefficient and we believe that having a systematic approach to finding the right network is the way to go.

On another note, the datasets given by ICCAD 2012 contest are highly imbalanced because hotspot images only account for less than 10% of the whole datasets. To get higher accuracy, besides using Auto-ML methods to get optimal hyperparameters, we also merged datasets from ICCAD1 to ICCAD5 to collect more data. However, the testing process would take a large amount of time because the dataset size is 1200*1200 pixels, and the number of images is also huge.

To save the total inference time, we tried pruning and downsampling. Pruning is beneficial to save resources by reducing the computation and the size of neural networks. We found that downsampling (and pruning) could help us save more than 90% of the inference time and more than 50% of the computation with comparable accuracy.

II. Problem formulation

The neural networks in question have many different hyperparameters. These hyperparameters include, most importantly, the number of convolutional layers (and its kernel sizes and output channels, and similar details) and the number of fully connected layers (and the neurons per layer and the activation functions). Other hyperparameters are more fine grained and include the optimization algorithm used for backpropagation, the number of epochs and the batch size.

The main point is that these hyperparameters are what is meant by a neural network architecture and determine a network’s performance. Models can be good or bad based on their complexity and their tendency to overfit; models can be good or bad based on their speed; or most importantly they’re bad based on their accuracy on a test set. One way to approach the problem of image classification with neural networks is to implement the same network used for classifying the MNIST dataset and then change the hyperparameters manually. This is a typical approach but it is unscientific and cumbersome.

Our proposed solution is to use an optimization algorithm that does this tweaking of hyperparameters and arrives at the best neural network. The tweaking is done by genetic algorithms or simulated annealing (GA or SA).

After selecting the best architecture we focus on further optimization by several means. Model pruning is a widely used approach to reduce the number of deep neural network (DNN) weights, which can effectively reduce the computation and storage costs and increase inference performance. A typical procedure to prune a DNN model consists of **1**.training a model to high accuracy, **2**.pruning the well-trained model, and **3**.fine-tuning the pruned model. There are two main methodologies: non-structured and structured pruning. The non-structured pruning methods lead to irregular weight distributions and inevitably introduce extra indices to store the locations of pruned weights. [8] In this project, we implement structured pruning because we want to save the extra processing time spent on index reference. Although the optimal networks we found are very small, most of the inference time is spent on data reading instead of computation. To save data reading time, we leverage downsampling to preprocess the datasets to shrink the image size.

III. Algorithms & Approaches

Simulated annealing - DAVID:

Our first algorithm is simulated annealing. This algorithm is well known and is based on statistical physics. The essential premise is to explore different solutions iteratively, each time selecting a neighbor solution, and either select or reject the new candidate solution according to an exponential weighting. The pseudo code is the following

```
• For i iterations until T=Tf
    – For each hyperparameter:
        • hyperparameter(i) =
          randn(1)*sigma+hyperparameter(i-1)
  ****(gaussian perturbation of the previous hyperparameters. This you can modify. Picking a neighbor is hard)*****
    – Train NN. Check validation accuracy.
    – Generate random number p in range [0,1]
    – If NN validation_accuracy(i) > validation_accuracy(i-1):
        • Accept NN
    – Elseif exp(-(val_accuracy(i-1)-val_accuracy(i))/T)>=p
        • Accept NN
    – Else
        • Reject NN
  T = T0 / (1+log(1+i)) (log decay)
```

This pseudocode says that we start at a temperature T and iterate until we reach a final temperature Tf. Then each hyperparameter is randomly perturbed by a gaussian perturbation. We then train the neural network according to the newest hyperparameters and check its validation accuracy, we then either select or reject depending on how **exp(-(val_accuracy(i-1)-val_accuracy(i))/T)** compares to a uniformly distributed probability **p**. Then we log decay the temperature.

This exponential weighting and comparison with **p** gives us the opportunity to avoid local minimas and explore hyperdimensional space.

Genetic algorithms - DAVID:

The second heuristic algorithm implemented is based on the evolutionary process of natural selection, chromosomal crossover, and mutation. This algorithm is easier to explain. Basically, we initialize a population of **n** neural networks, randomly generated. Then we need to train and score each of these neural networks. The top **x** number of neural networks get selected for the next generation. In the next generation of neural networks, there is breeding, which is where new neural networks are created when two parent neural networks mate. This breeding process happens through crossover, which basically selects some hyperparameters of one parent and some hyperparameters of the other parent in order to create a child, and then once the child neural network is defined, mutations occur that change the hyperparameters in a gaussian way, similar to our implementation of simulated annealing.

Combined SA and GA - DAVID:

Another idea that was tested was using both algorithms as one single algorithm. What we chose to do is use simulated annealing to generate the initial population for GA. This gives a good starting point for GA, and then we evolve according to GA. This was proven to run the fastest in our experimental results section.

Note about simulated annealing and genetic algorithms-DAVID:

In principle you can vary any of the hyperparameters. Given the greater relevance of some hyperparameters we only choose to vary or search over the space of number of fully connected layers, the number of neurons per layer, the number of convolutional layers, the kernel sizes, the output channel sizes, and the number of epochs. The max pooling is kept as a (2,2) window, stride=1, padding="same", activation functions are always ReLu, we use adam optimization, binary cross entropy loss, and a fixed batch size. The user can use our source code and make additions to search over an ever wider space, although this shouldn't be necessary.

Pruning & downsampling-ZIWEI:

Before we derived the optimal neural network model, we did some experiments to test the feasibility of pruning and downsampling on a VGG-like neural network model. First, we tried pruning on a VGG-16 neural network with all the datasets from ICCAD1 to ICCAD5. We only changed the size of fully connected layers to match the size of the image to VGG-16. However, we found that the GPU exploded due to the large size of both the neural networks and datasets. Therefore, we optimized the neural networks by using only ¼ of the original channels and removed extra neurons in fully connected layers. We pruned channels in convolutional layers for the pruning part and compared the overall inference time, accuracy, and computation with our baseline method (without pruning & downsampling). We also downsampled the original image size(1200*1200) with a stride of 2 to get a smaller image size(600*600) by taking the 2 by 2 square's average value.

After the optimal network hyperparameters are derived, we found that some of the layers only have 1 channel, and the 2 models are small. So we did not prune these 2 models. However, the inference process still took us much time even though we use a small neural network model and only ICCAD1 and ICCAD2 for training. Therefore, we did downsampling for the 2 derived optimal models. Because downsampling will sacrifice original images' information, we used the merged datasets from ICCAD1 to ICCAD5.

IV. Experimental Results

Simulated annealing-DAVID:

Shown in figures 1-4, are the performance of simulated annealing over the combined ICCAD1 and ICCAD2 datasets. Figure 1 shows how by the 12th generation we already have the optimal accuracy. Figure 2 shows the thermal annealing process and how when the system gets colder, we arrive at the optimal solution. Figure 3 shows receiver operating curves which are necessary plots for binary classification. These ROC show a curve whose ideal area is = 1. The area is 0.996. Moreover ROC shows true positive rate and false positive rates. Figure 4 shows the full classification report and more nuanced statistical measures of the classification process. In summary this model is as good as CUHK.

Another very important aspect of simulated annealing is the output architecture. CUHK uses 18 convolutional layers and 3 dense layers. We use a lot less parameters because our final optimal network has 4 convolutional layers and 5 fully connected layers. The specific architecture at the end of the process is: output channels are [1,4,1,1]. The kernel sizes are [2,7,2,5]. The model also has 5 fully connected layers with [13,177,127,122,133] neurons. Stride = 1, padding = "same", max pool = (2,2) and ReLu functions with adam optimization, a sigmoid function at the last layer, and binary cross entropy loss. In general though, every single run of this code will produce a different final architecture. We will present runtime comparison for SA and GA in a separate section.

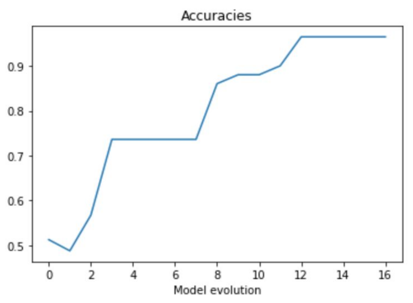


Figure 1: Model accuracy for SA. After 12 iterations, we arrive at 0.98 accuracy.

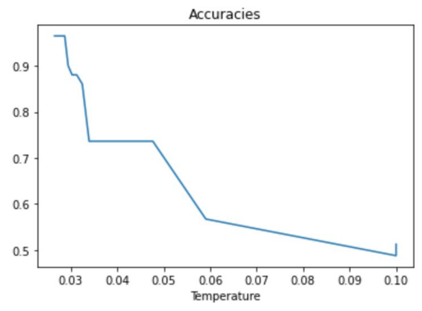


Figure 2: Model accuracy for SA. This plot shows how the quenching of the algorithm results in higher accuracy.

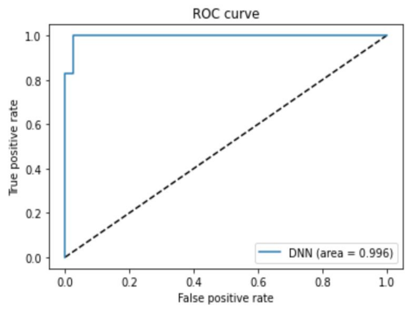
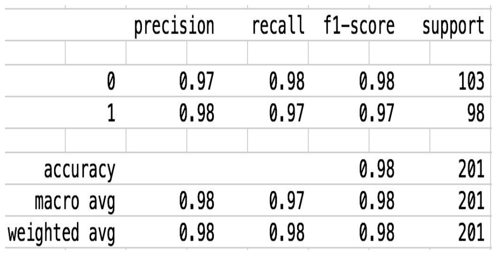


Figure 3: To address a binary classification problem we need ROC curves in addition to an accuracy metric. This plot shows how there is little misclassification.



	precision	recall	f1-score	support
0	0.97	0.98	0.98	103
1	0.98	0.97	0.97	98
accuracy			0.98	201
macro avg	0.98	0.97	0.98	201
weighted avg	0.98	0.98	0.98	201

Figure 4: This is our full classification report for simulated annealing

Genetic algorithm-DAVID:

Shown in figures 5-7 is the performance of GAs. Figure 5 shows the evolution of each generation and their accuracies. It takes 14 generations to achieve an accuracy of 0.98. Figure 6 shows ROC curves discussed in the previous section and we show how there is little misclassification. Figure 7 shows the full classification report. Again, GAs are capable of achieving a 0.98 accuracy. This algorithm was specifically implemented varying the same parameters as SA, and each generation has a population of 10 neural networks. The final network has 5 conv layers with output channels [1 1 4 1 4] and conv kernel sizes [3 7 1 5 1] and 2 dense layers with neurons [42 29]. More fine grained details are: Adam optimization, binary cross entropy, a sigmoid function at the end layer, relu activation, and 12 epochs. Stride = 1, pad = "same", max pool between conv layers.

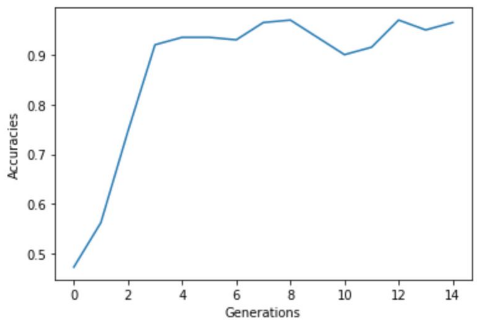


Figure 5: Model accuracy for GA. After 14 generations, we arrive at 0.98 accuracy.

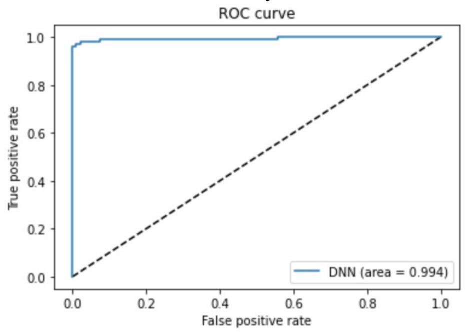
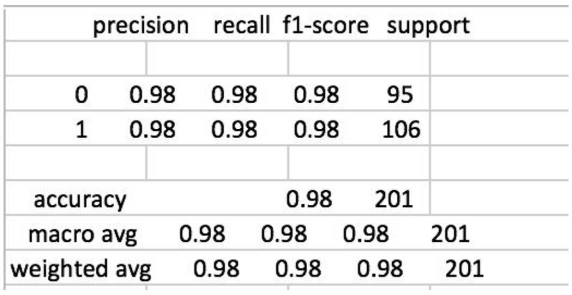


Figure 6: To address a binary classification problem we need ROC curves in addition to an accuracy metric. This plot shows how there is little misclassification for GA.



	precision	recall	f1-score	support
0	0.98	0.98	0.98	95
1	0.98	0.98	0.98	106
accuracy			0.98	201
macro avg	0.98	0.98	0.98	201
weighted avg	0.98	0.98	0.98	201

Figure 7: This is our full classification report for GA.

Simulated Annealing and Genetic Algorithms runtime comparison:

In figure 8 we see the comparison of runtime between SA and GA. GA is slower but steadily increases. SA is faster and consumes 40% of the time, but it can have volatility.

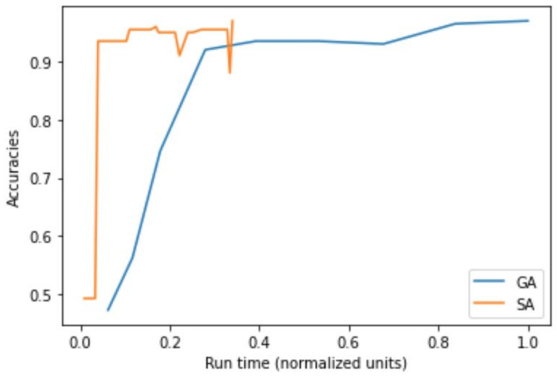


Figure 8: Runtime comparison for SA and GA. We see how long it takes for either GA or SA to get a 0.98 accuracy on the merged ICCAD1 and 2 datasets. SA takes 40% of the time GA takes to achieve 0.98 accuracy.

GA with initial SA-DAVID:

Figure 9 demonstrates the combination of GA and SA. It turns out that starting a population with SA that reaches a threshold accuracy of 80% and then evolving according to GA leads to faster results. One possible reason for this is that SA has great volatility, while GA tends to steadily increase. Therefore it seems our fastest algorithm is initializing a population with SA and then using GA. We specifically make the initial population after the SA algorithm produces parent networks with 80% accuracy. This threshold can be further explored.

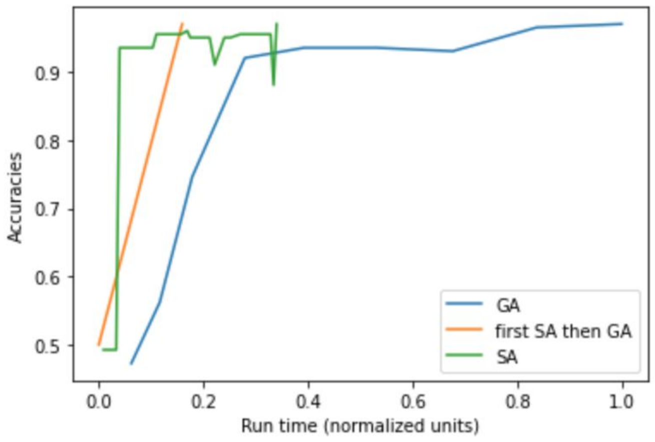


Figure 9: Using SA and then evolving with GA consumes the less runtime and taes about 20% of the time to achieve 0.98 accuracy.

Pruning-ZIWEI:

Before doing Auto-ML, we did some experiments about pruning and downsampling to see if these 2 methods could efficiently reduce the inference time. Our first experiment used a popular neuron network VGG-16 to train all the datasets from ICCAD1 to ICCAD5 without preprocessing the given images. However, the GPU exploded because of the large size of our network. Therefore, we chose a VGG16-like model by reducing ¾ of the channels in VGG-16 and reducing fully connected layers' neurons, and the network configuration is shown in Table1. Table2 and Table3 show the results we got after training and pruning with the Table1 network configuration .

Layer	Stride	Padding	original		pruning	
			Kernel size	Output Vertexes	Kernel size	Output Vertexes
Conv1-1	1	1	3×3×16	1200×1200×16	3×3×15	1200×1200×15
Conv1-2	1	1	3×3×16	1200×1200×16	3×3×13	1200×1200×13
Pool1	2	0	2×2	600×600×16	2×2	600×600×13
Conv2-1	1	1	3×3×32	600×600×32	3×3×26	600×600×26
Conv2-2	1	1	3×3×32	600×600×32	3×3×20	600×600×20
Pool2	2	0	2×2	300×300×32	2×2	300×300×20
Conv3-1	1	1	3×3×64	300×300×64	3×3×39	300×300×39
Conv3-2	1	1	3×3×64	300×300×64	3×3×39	300×300×39
Conv3-3	1	1	3×3×64	300×300×64	3×3×26	300×300×26
Pool3	2	0	2×2	150×150×64	2×2	150×150×26
Conv4-1	1	1	3×3×64	150×150×64	3×3×26	150×150×26
Conv4-2	1	1	3×3×64	150×150×64	3×3×26	150×150×26
Conv4-3	1	1	3×3×64	150×150×64	3×3×26	150×150×26
Pool4	2	0	2×2	75×75×64	2×2	75×75×26
Conv5-1	1	1	3×3×64	75×75×64	3×3×26	75×75×26
Conv5-2	1	1	3×3×64	75×75×64	3×3×26	75×75×26
Conv5-3	1	1	3×3×64	75×75×16	3×3×64	75×75×16
Pool5	2	0	2×2	37×37×16	2×2	37×37×16
FC1	-	-	-	128	-	128
FC2	-	-	-	64	-	64
FC3	-	-	-	2	-	2

Table1: Network configuration after FC optimization

From table2, we found that after pruning, the overall accuracy, HS and NHS accuracy actually increased.

TABLE 2: accuracy after FC optimization

method	HS accuracy	NHS accuracy	accuracy
original	98.65%	96.38%	98.6%
pruning	98.68%	96.71%	98.62%

Table 3 (after FC optimization) shows the resource cost with fewer FC layer neurons and Table4 (before FC optimization) shows the resource cost with more FC layer neurons.

TABLE 3: resource cost after FC optimization

method	MACs($\times 10^{10}$)	weights($\times 10^{6}$)	inference time(s)
original	2.04	3.12	1326
pruning	0.93(-55%)	2.88	1195

TABLE 4: resource cost before FC optimization

method	MACs($\times 10^{10}$)	weights($\times 10^{7}$)	inference time(s)
original	2.06	1.16	1323
pruning	0.94(-55%)	1.13	1251

From Table3 and Table4, we found that although the number of weights is only 10% of the original one after FC optimization, their inference time remains nearly the same. Moreover, the computation is reduced by more than 55% after pruning, while the inference time only decreased by 10%. Therefore, we downsampled the images to shrink the datasets' size to explore the relationship between data reading and inference time.

Downsampling-ZIWEI:

We down-sampled the original image size (1200*1200) with a stride of 2 to get a smaller image (600*600) by taking the 2 by 2 square's average value. We use downsampled images in VGG16-like network model, we found that after downsampling, the accuracy reaches 98.6%, while the inference time is only 168.5, about 12.2% of the original method.

The above experiments proved the benefits of both pruning and downsampling, so we planned to combine the 2 methods with the optimal network architectures derived by SA and GA. However, these 2 network models are very simple, some layers only have 1 channel, so we only did downsampling for them.

Because downsampling would lose some information, when we only use ICCAD1 and ICCAD2, the NHS accuracy is not good. So we merged all the available datasets from ICCAD1 to ICCAD5. From table5, we found that for both SA and GA, if we only use ICCAD1 and ICCAD2 for downsampling, we would lose much accuracy for NHS. Moreover, downsampling helps us to save more than 88% of inference time without losing accuracy.

TABLE 5: accuracy and inference time after downsampling

method	datasets(ICCAD)	downsampling	HS accuracy	NHS accuracy	accuracy	inference time
SA	1&2	YES	99.3%	85.4%	99%	63
SA	1&2	NO	98.8%	98.9%	98.9%	550
SA	1-5	YES	99.3%	96.5%	99.2%	150
SA	1-5	NO	98.4%	99.0%	98.3%	1342
GA	1&2	YES	98.9%	89.3%	98.6%	72
GA	1&2	NO	99.3%	98.8%	99.2%	562
GA	1-5	YES	97.8%	97.2%	97.6%	163
GA	1-5	NO	99%	99.1%	99.1%	1358

V. Conclusions

We have explored various ways of adding to the body of work previously done on lithographic hotspot detection. As mentioned in the introduction, the accuracy of neural networks on lithographic hotspot detection is pretty much established. We have contributed therefore in the direction of increasing model efficiency. Part of this idea of efficiency is explainability. Explainability can come in many different flavors, but one of them is answering the question of how one comes up with a particular network. Moving forward in the creation of machine learning models for problems in physical design, and in general, any problem that requires machine learning, the vast majority of literature and trends suggests that selecting a model will become an automated process. Computer algorithms will select a particular model after automatically trying several and then choose this one. This we can say adds to the aspect of explainability because we can say that the chosen model was arrived at after an optimization algorithm.

Two particular optimization algorithms that work are SA and GAs. Here show that SA and GA arrive at the same conclusions as more complex networks.

We also improve efficiency by reducing computation and reference time with the help of pruning and downsampling. Pruning is beneficial in deep complex neuron networks by increasing the compression ratio and saving resources. Data reading consists of most of the training and inference time. In our experiment, we proved that downsampling could reduce inference time by more than 88%. This is important because, with less training and inference time, people could easily do more fine-tuning.

Therefore our main conclusion is that Auto-ML, Pruning and downsampling should be adopted

References:

[1] Duo Ding, Xiang Wu, J. Ghosh and D. Z. Pan, "Machine learning based lithographic hotspot detection with critical-feature extraction and classification," 2009 IEEE International Conference on IC Design and Technology, Austin, TX, 2009, pp. 219-222, doi: 10.1109/ICICDT.2009.5166300.

[2] K. Madkour, S. Mohamed, D. Tantawy and M. Anis, "Hotspot detection using machine learning," 2016 17th International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, 2016, pp. 405-409, doi: 10.1109/ISQED.2016.7479235.

[3] V. Borisov and J. Scheible, "Lithography Hotspots Detection Using Deep Learning," 2018 15th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), Prague, 2018, pp. 145-148, doi: 10.1109/SMACD.2018.8434561.

[4] Nagase, Norimasa, et al. "Study of hot spot detection using neural networks judgment." Photomask and Next-Generation Lithography Mask Technology XIV. Vol. 6607. International Society for Optics and Photonics, 2007.

[5] X. Yindong and H. Xueqian, "Learning lithography hotspot detection from ImageNet," 2019 14th IEEE International Conference on Electronic Measurement & Instruments (ICEMI), Changsha, China, 2019, pp. 266-273, doi: 10.1109/ICEMI46757.2019.9101673.

[6] GULCU, A., & KUS, Z. (2020). Hyper-Parameter Selection in Convolutional Neural Networks Using Microcanonical Optimization Algorithm. *IEEE Access, 8*, 52528-52540. doi:10.1109/ACCESS.2020.2981141

[7] Haoyu Yang, Luyang Luo, Jing Su, Chenxi Lin and Bei Yu, “Imbalance Aware Lithography Hotspot Detection: A Deep Learning Approach”, SPIE Intl. Symp. Advanced Lithography Conference, San Jose, CA, Feb. 26–Mar. 2, 2017.

[8]J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635, 2018