A Deep Transfer Learning Design Rule Checker with Synthetic Training

Luis Francisco, Paul Franzon, *Fellow, IEEE*, and W. Rhett Davis , *Member, IEEE*
Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC 27606 USA

The number of design rules is drastically increasing as technology nodes scale down. This increase makes the design rules decks' creation and the checking process complex and time-consuming. This work presents design rule checking using deep learning. The core of the work consists of a framework to generate convolutional neural networks and a parametrized synthetic data generator for training. The models incorporate incremental transfer learning to reduce the training time when adding new rules. The results show that we can capture most of the checks from a design kit rule manual with less than 1% error and up to $7x$ faster than traditional design rule checkers.

***Index Terms*—Design Rule Checking; Machine Learning; IC Verification; Convolutional Neural Network; Deep Learning; Synthetic Data Training; Transfer Learning.**

I. Introduction

Design rule checking (DRC) is the process of verifying that a design layout meets a set of rules that make it suitable for fabrication. The rules define geometric constraints to satisfy the physical limitations of the lithography and the fabrication process. Satisfying these constraints guarantee that the design will achieve a high fabrication yield. The design rule manual establishes the parameters for each check for each layout layer.

Typically, the design rule manual specifies checks consisting of width, length, spacing, area, enclosure, and overlap rules. The number of rule checks has increased to thousands in advanced nodes. This increase is due to the aggressive scale down of the nodes and the increase in complexity of the fabrication processes. Creating a DRC runset/deck for a technology node and running a rule checker for an entire layout is a tedious and time-consuming process.

DRC is an iterative process required as early as possible in technology development and chip designs. When the designer fixes a rule violation, it may insert other violations which requires to re-run the DRC checker, generating the need for a fast, and easy to develop design rule checker. In [1], we establish the proof of feasibility to take advantage of machine learning to create such an engine. Note, we are not suggesting replacing final sign-off DRC with a ML based one, but using the faster machine learning (ML) solution while iterating on the design.

This work presents a framework that allows using deep learning (DL) models to detect all possible design rule violations found in a typical process design kit (PDK). The framework includes a unique synthetic dataset generator and transfer learning to train the model when adding new checks quickly and easily. It can identify multiple simultaneous violations.

II. Background on Existing DRC with ML

Deep learning (DL) is a subset of Machine learning (ML), where the model progressively learns in layers by extracting data representations. Other ML algorithms not considered as DL require more manual adjustment to learn those representations. This difference implies that DL does not require an exhaustive manual feature extraction from the training data and captures significant effects beyond the physical parameters that an engineer can include in a model. It also opens the possibility to transfer the learning from problems that share a similar nature, in this case, from one DRC violation type to another.

Machine learning (ML) has proved to solve multiple problems showing significant improvement to traditional methods. In chip design and electronic design automation (EDA), ML is being explored from design exploration to physical verification to accelerate the design cycle [2].

Recent works applying machine learning to the design rule problem, most of them focusing on predicting the number of violations, include [3], [4], [5], [6] and in a VLSI physical design flow [7]. We can include these works on the side of traditional machine learning.

The work in [3], [5] predicts DRC violations before the routing stage by using random forest, gradient boost, and a voting strategy to combine both. [4] uses a set of input knobs to the physical design flow to construct a surrogate model and predict the number of violations after the detailed route. [6] employs an ensemble of neural networks (NN) to predict rule violations after the detailed route stage. The NNs use a set of parameters from the global route as training data. Finally, after a voting algorithm, a final prediction is done and compared to a random forest model.

On the side of deep learning, we can find some work to predict DRC violations [7], [8]. Both works focus on a VLSI physical design flow. [7] uses a convolutional neural network (CNN) to predict routability. In this process, they predict the number of DRC violations and hotspot areas. The model takes input features from placement and route stages represented as images. [8] uses a CNN and trains it with images of the pin locations in the layout and other features extracted from the global route. Both [7], [8] can locate hotspots areas around placed cells.

In contrast with the works previously described, our pro-

posed design rule checker can handle multiple types of DRC. This approach not only finds the number of violations and locate possible hotspot areas but it can classify them and point to the violations in a small action window. This difference allows the framework to be ideal for both tool generated and full custom layouts. It is suitable to use on chip layouts to quickly and accurately locate and classify violations while iterating on designs.

We can find transfer learning applied to some EDA problems [9], [7]. [9] implements neural network models that partially share weights for power and performance estimation in memories. [9] provides some guidance on how sharing weights affects performance. The work in [7] starts with a pre-trained RouteNet structure for image pattern recognition and tunes it with the specific training data.

In [1] we have a model to detect three width and spacing violations. For one rule violation, the detection accuracy is 92%, but it reduces when adding more rules. The accuracy reduction is due to overfitting in the model for the lack of diversity in the training dataset. The decrease in performance for data never seen is a recurrent issue in this type of problem [10].

In this work, we focus on expanding [1] to support a complete design rule set, including rules that involve the interaction of multiple layers. Using a synthetic dataset generator, we solve overfitting and avoid the model from learning the rough global structure of specific layout polygons. With the inclusion of transfer learning, we make the process of adding new rules fast and straightforward. Transfer learning reduces the amount of data to train the model for new rules and the training time.

III. Design Rule Checking Framework

The proposed design rule checker framework consists of an ensemble of deep learning (DL) models which share weights (transfer learning) and a parametrized synthetic dataset generator (PSDG). The PSDG takes the input of layout and polygons parameters to generate a sampling space for each design rule violation (DRV). The main advantage of this framework is that it can be expandable to any number of rules. With the PSDG, we can generate data to include new checks to the model. The transfer of weights allows reducing the training time and training resources.

The inputs to the model are layout clips in the form of $W \times W$ image tensors $I(x, y)$. In the case of rules that involve only one layer, each pixel $I(x, y)$ is a scalar value representing a grayscale image. When multiple layers are involved, $I(x, y)$ has three components to represent an RGB image. Each sample that the PSDG creates an image as described before. We choose the image size $W \times W$ to make that one pixel correspond to $1nm$. By reducing action windows W, we increase the resolution of the detection. In each image, there is an area without polygons creating centered images. This centering area is given by the distance c. Figure 1 shows an example of one and two layers image clips. For Figure 1, the layout clip on the left is for a single layer, for example, a metal layer, while the layout on the right is for two interacting layers, for example, a metal layer and a via layer.

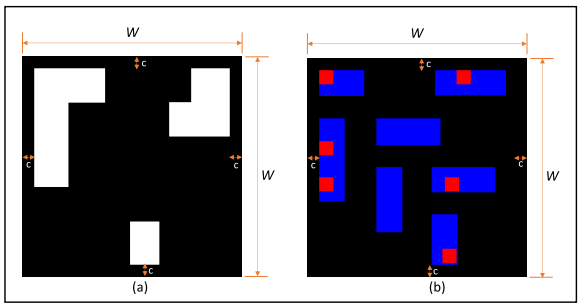

Figure 1. Layout clip samples for one layer (a) and two layers (b).

The rest of this section explains the deep learning model structure, the parameters selection and the synthetic dataset generator.

A. Deep Learning Architecture with Transfer Learning

The deep learning model involves a convolutional neural network (CNN) to perform automatic feature extraction and a set of fully connected layers to perform the DRC violations classification. The CNN contains convolutional layers (CL) and downsampling layers. The CLs perform a convolution operation to their input, and the downsampling reduces their output size. We use max pooling for the downsampling layers.

A set of weights map/storage the knowledge learned in each layer. To adjust those weights, the DL models optimizes a loss function during the training process. The difference between the predictions and the labels in the data creates the loss function. For our models, we use cross-entropy as the loss function and root mean square proportional ($RMSprop$) for the optimizer. The activation function in the CLs and the FC layers is rectified linear unit (ReLU) [1].

To classify each DRC violation, we create an ensemble of CNN and fully connected (FC) layers, creating a complex deep network structure. Figure 2 illustrates the final deep network architecture. We use an ensemble of models instead of a single multiclass to reduce the degradation in performance found in [1]. This new approach reduces the number of trainable parameters and provides an incremental framework to add new DRC checks.

The idea of our deep learning architecture is to have a base model that consists of a set of hidden layers or CLS plus the FC layers. A subset of the CLs share the weights for all the violations. We retrain the remaining CLs and the FC layers with data specific to each violation. This training strategy applies the concept of incremental training with transfer learning [11].

B. Selecting the Layers Size and Adding DRCs

To reach the final base model structure in Figure 2 we took the work in [1] to create a starting point. The number of CLs in the final base model is five (CL_1 to CL_5). To obtain this number, we follow an intuitive approach that consists of increasing the number of CLs until the model starts overfitting. When we reach overfitting, we go back to the previous number of CLs. We use the same intuitive approach to the FC layers

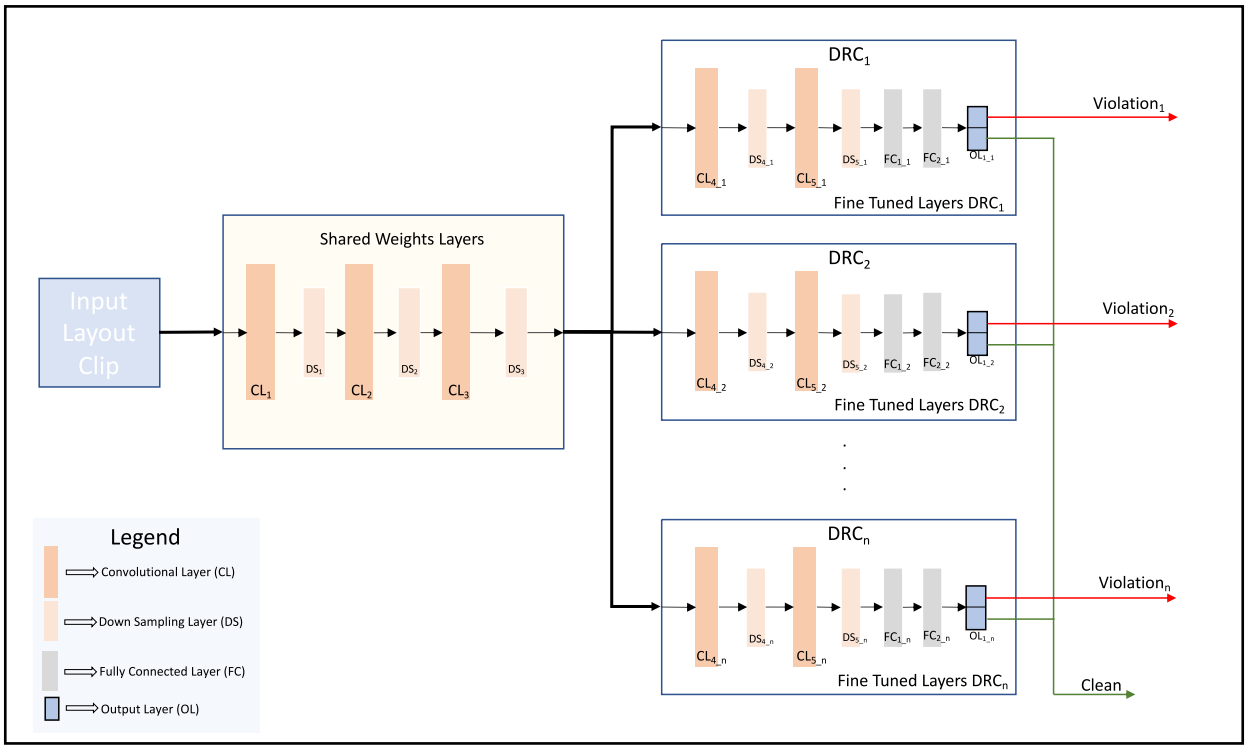

Figure 2. Deep network architecture to add new DRC violations incrementally by sharing weights.

and the size of both FC layers and CLs. The CLs size is the number of filters and filter size. Note that there is no deterministic path to find the number of layers and their size; the best way is to use intuition and experimentation.

To select the percentage of sharing between rules, we follow some experimentation as in [11], [9]. From [11], we can see that sharing a high percentage of weights drastically affects the accuracy. They test sharing up to 60% of the parameters for a generic image classification problem. In our case, we are working with a particular and complex problem; we try to keep the percentage of shared weights around 10% or lower to prevent the model from learning polygons related to a single violation. Also the synthetic dataset generator allows us to quickly and easily access training data.

To incrementally add new design rule checks, we start with a pre-trained base model. The base model is pre-trained for the first rules added to the model (CL_1, CL_2, CL_3). For each new rule we train the layers CL_4, CL_5, $FC1_1$, and FC_2.

The output layer size is two to classify the layout clips as a specific violation or clean. The output layer contains two probability values. To assign the clip to a class, we take the maximum of the two. In the case, two or more networks flag an input as a violation, we assign that input to have multiple violations. If the networks does not flag the input in any DRC we consider it as clean.

C. Synthetic Dataset Generator

One of the key elements of the framework is the parameterized synthetic dataset generator (PSDG). The generator takes a set of input parameters for clean and violation layout clips and generates image clips to train the model. With the parameters, it creates a sampling space using the Latin hypercube (LHC) sampling strategy. We have a wide range of clean and violation clips with different polygon arrangements by creating a sampling space; this increases the variance in the dataset. With the PSDG, we avoid the model from learning only the rough global structure of the polygons and make sure it learns more about the violations. Another advantage of the PSDG is that we can adapt our models to a new technology node by changing the parameters.

In the PSDG, we can define parameters for clean and violations for each layout layer via layers, alignment, enclosure, and area. Next, we list the parameters and what each of them defines in the layout clips:

- **Min and max layer width**: range of polygons width.
- **Min and max layer length**: range of polygons length.
- **Min and max horizontal spacing**: spacing between polygons
- **Min and max horizontal vertical**: spacing between polygons
- **Min and max number of polygons** per sample.
- **Polygons orientation:** only vertical, only horizontal, or both.
- **Polygons overlap ratio**: how much overlap there will be between horizontal and vertical shapes. Allows creating complex polygons.
- **Clean to violation ratio:** how many polygons will contain violations.

- **Min and max vias width and length:** defines vias shapes.
- **Vias or layer enclosure (min/max horizontal and vertical distances):** defines the enclosure distance for vias and layers.
- **Vias or layer alignment (min/max horizontal and vertical distances)**: determines layers and vias alignment.
- **Min and max area for polygons**: defines enclosure and alignment areas.

To create an N samples dataset, the generator creates a sample space of N samples of the parameters (P). We chose LHC to do the intelligent sampling because it is not entirely random; each new sample considers the previous one. Another benefit of LHC for this approach is that we can have a multi dimensional sampling space. Algorithm 1 describes the process to generate the training dataset for a DRC violation. Each P is re-sampled to create a new sampling space for the polygons corresponding to that layout clip.

Algorithm 1 Synthetic Dataset Generator for a DRC

$N \leftarrow$ Number of samples
Obtain clean parameters vector $P_c = [p_0, ..., p_m]$
Obtain violation parameters vector $P_v = [p_0, ..., p_m]$
Sample layout clean parameters $L_c = LHC\,(P_c)$
Sample layout violation parameters $L_v = LHC\,(P_v)$
$i = 0$
while $i < N$ **do**
    Create polygons sample space $pi_c = LHC\,(P_c[i])$
    Create polygons sample space $pi_v = LHC\,(P_v[i])$
    DRP(pi_c)
    DRP(pi_v)
    $i = i + 1$
procedure DRP(poly, n)
    create empty image I
    while $j < n$ **do**
        $xy =$ Create coordinates from $poly[j]$ parameters
        if xy not inside image & center distance **then**
            adjust xy to image size
        Draw coordinates xy in image
        Save image
        $j = j + 1$

To choose the value for the parameters to create the dataset, we only require the constraints for the DRC violation from the process design manual. For the clean layouts, we need to understand the design types we are making with this process. The dataset generator enables creating data representing custom layouts, standard cells VLSI layouts, memories, among others.

IV. Results

We implemented, tested and validated our design rule checking framework using Python and TensorFlow. For demonstration purposes, we use the rule set for FreePDK15 to avoid proprietary issues. The FreePDK15 is a multi-gate 15nm FinFET open process design kit (PDK) [12]. This PDK contains over 200 DRC rules. Those rules cover width, length, spacing, area, enclosure, overlap, and alignment.

In the rest of this section, we detail the final model architecture and the dataset generated. We also analyze the transfer learning and the overall performance of our framework.

A. Base Model Architecture and Generated Dataset

After following the procedure described in section III-B, we ended up with five convolutional layers and two fully connected layers for our base model. The layers size $[CL_1, CL_2, CL_3, CL_4, CL_5]$ is $[8, 16, 24, 24, 16]$. The fully connected layers size $[FC_1, FC_2]$ is $[64, 64]$. The size of each convolution filters is $3x3$ and $2x2$ downsampling layers. This base model has a total of $43,362$ trainable parameters for an input of $450x450$ pixels resized as by 2.

To train the base model, we generated a dataset with a width and a spacing violation. The samples in this dataset are $80,000$ clean and $40,000$ with the two violations. The dataset generated uses an action window of $400nmx400nm$ centered with a distance of $50nm$; this results in $450x450$ pixels images. To choose this action window, we considered that in the FreePDK15 polygon width is $24nm$ and the minimum spacing of $2nm$. The resolution inside a $400nmx400nm$ area is pretty reasonable to detect the violations.

After training the base model, we added $20,000$ samples per each new DRC added to the model. We use this dataset to fine-tune CL_4, CL_5 and the fully connected layers. For new rules, the model shares 11% of the hyperparameters (weights) for a total of trainable parameters $38,634$.

To analyze the effect of sharing 11% of the model weights, we created two DRC models with and without transfer learning. Figure 3 shows that the loss in accuracy when sharing the hyperparameters is less than 0.01%. With a neglectable impact on the accuracy, we obtain a decrease in the training time of about 30%. The training time goes from 2.52 hours to 1.1 hours for a $2.3x$ speedup in training. This training time reduction is because the model converges faster and benefits from using transfer learning.

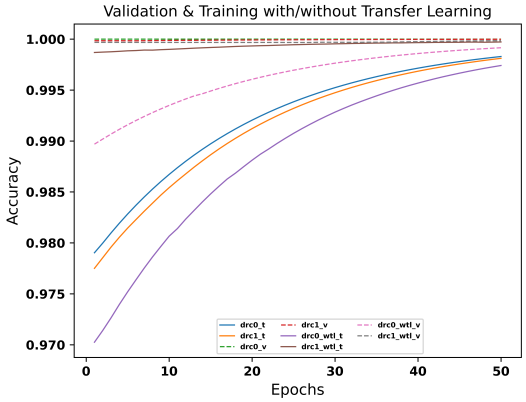

Figure 3. Training and validation accuracy for rules with a completely deep network trained vs partially trained with transfer learning. Legend description: $drcx\_t$ is training accuracy, $drcx\_v$ is validation accuracy, and $drcx\_wtl\_v/t$ is training/validation accuracy with transfer learning model.

B. Model Training

The hardware used to train the models is a machine with four Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz processors and a total of 376GB of memory. This machine does not have any GPU. The total time to train the model for 200 violations is 186.7 hours, and the time to generate the dataset to train the models is 36.4 hours.

To test the model and present the paper results, we selected 20 rules in different metal layers that cover most of the rules in the FreePDK15. Figure 4 shows the model accuracy during training and validation. From Figure 4 we can verify that there is no overfitting or underfitting. Both the training and validation accuracy converges between 98% and 99% after 40 training iterations.

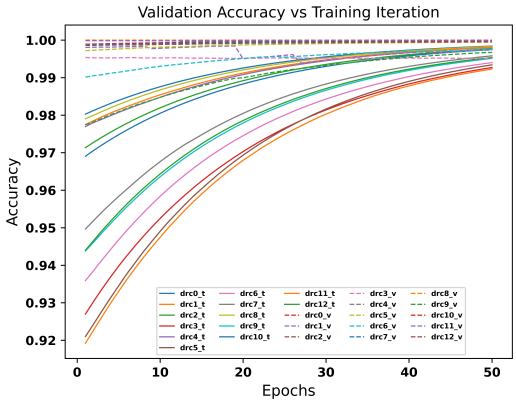

Figure 4. Training vs. validation accuracy for 13 of the rules implemented. Legend description: $drcx\_t$ training accuracy and $drcx\_v$ validation accuracy.

Figure 5 shows that the training and validation losses also settle down after 40 training iterations. At the beginning of the training, there is some noise and high losses. These are more significant complex rules that include the iteration of multiple layers.

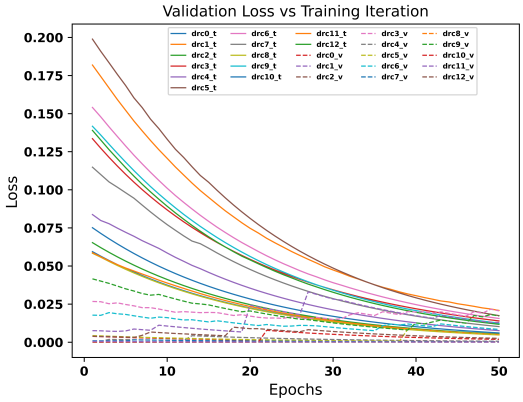

Figure 5. Training vs. validation loss for 13 of the rules implemented. Legend description: $drcx\_t$ training loss $drcx\_v$ validation loss.

C. Testing the Model with Unseen Data From Real Layouts

Table I shows the results summary. The model achieves a true positive rate (TPR) between 96.4% and 100%, and a false positive rate (FPR) between 1.1% and 5.3%. If we compare with our previous models [1] we have a significant improvement in both TPR and FRP, and now we have a fully scalable framework that can extend to cover any number of violations.

The results in Table I that with the synthetic dataset generator, the model is not learning the global structure of the polygons but is learning the violations. The model identifies multiples violations in the defined action windows of $400x400nm$ and labels each of them. Note, this action window is another parameter we can adjust.

The works we can find in the state-of-the-art as described in II does not identify or label each violation. But if we compare the overall TPR and FPR 99.2% and 3.3% respectively, our approach presents a significant improvement. Those works estimate the total number of violations using random forest, neural network, surrogate models, and CNN with a TPR as low as 60% and up to 97% in the best-case scenario. Our approach not only can improve those TPRs, but it detects multiple violations and labeld them.

Finally, we measure the run-time of a conventional DRC engine vs. the inference time for the deep learning model. We use the machine used for training but limiting the number of cores to 8. We test one violation at a time and only load the layout with the layers needed for that violation. As a result the deep learning model completed the checking $7.4x$ faster.

V. Conclusion

In this article, we presented a deep learning approach to design rule checking. The approach uses a parameterized synthetic dataset generator and transfer learning to add new design rule checks. Our framework detects DRC violations with a detection rate of 96.4% to 100%, depending on how complex the rule is. The false alarm rate is 5.3% in the worst-case scenario. The approach can locate the violations and identify them in action windows $7.4x$ faster than conventional checkers. This solution to DRC provides an alternative while iterating on the design.

Since we demonstrated that this is scalable to any number of rules as part of future work, we will be focusing on transferring to another technology PDK. We also plan on doing more analysis on the transfer learning and work with a model that can take as input all the layers in the layout. Another are we will work on is to improve the inference time by optimizing the framework implementation.

TABLE I
RESULTS SUMMARY OF TESTING 20 DIFFERENT TYPES OF VIOLATIONS IN LAYOUTS DATA UNSEEN BY THE MODEL. THE DATA CONSIST OF AN SRAM LAYOUT AND A ROCKET CORE DESIGN.

Design Rule Checks		Violations				Clean			
DRC	**Type**	**True Label**	**Found**	**Missed**	**TPR**	**True Label**	**Found**	**Missed**	**FPR**
drc0	Horizontal width	32,630	32,150	480	**98.5%**	18,850	17,854	996	**5.3%**
drc1	Horizontal Spacing	36	36	0	**100.0%**	7,720	7,484	236	**3.1%**
drc2	Vertical overlap	4,464	4,326	138	**96.9%**	15,796	15,027	769	**4.9%**
drc3	Vertical length	59,159	58,813	346	**99.4%**	18,289	17,693	596	**3.3%**
drc4	Vertical spacing	120	119	1	**99.2%**	31,043	30,036	1,007	**3.2%**
drc5	H/V width	20,912	20,875	37	**99.8%**	6,414	6,162	252	**3.9%**
drc6	Polygon area	21,331	21,197	134	**99.4%**	16,779	16,311	468	**2.8%**
drc7	H/V Spacing	18	18	0	**100.0%**	37,034	34,985	2,049	**5.5%**
drc8	Spacing dependant on width/length	2,687	2,657	30	**98.9%**	48,350	47,804	546	**1.1%**
drc9	H/V width	10,698	10,633	65	**99.4%**	4,853	4,789	64	**1.3%**
drc10	Polygon area	2,405	2,352	53	**97.8%**	9,841	9,522	319	**3.2%**
drc11	H/V spacing and notch	30	30	0	**100.0%**	9,207	8,898	309	**3.4%**
drc12	Spacing dependant on width/length	19	19	0	**100.0%**	18,351	18,109	242	**1.3%**
drc13	Via shape square	93	93	0	**100.0%**	13,118	12,634	484	**3.7%**
drc14	Via shape rectangular	77	77	0	**100.0%**	35,816	34,749	1,067	**3.0%**
drc15	Via spacing	51	51	0	**100.0%**	63,513	61,045	2,468	**3.9%**
drc16	Via inside layer	230	228	2	**99.1%**	2,017	1,945	72	**3.6%**
drc17	Via enclosure horizontal	241	238	3	**98.8%**	503	483	20	**4.0%**
drc18	Via enclosure vertical	99	98	1	**99.0%**	738	699	39	**5.3%**
drc19	Horizontal width	22,788	22,711	77	**99.7%**	16,241	15,791	450	**2.8%**
	Totals	178,088	176,721	1367	**99.2%**	374,473	362,020	12,453	3.3%

ACKNOWLEDGMENT

The authors would like to thank the member companies of the CAEML IUCRC and the National Science Foundation (award CNS 16-244770) for their partial support of this work.

REFERENCES

[1] L. Francisco, T. Lagare, A. Jain, S. Chaudhary, M. Kulkarni, D. Sardana, W. R. Davis, and P. Franzon, "Design rule checking with a cnn based feature extractor," in *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, ser. MLCAD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 9–14.

[2] B. Khailany, H. Ren, S. Dai, S. Godil, B. Keller, R. Kirby, A. Klinefelter, R. Venkatesan, Y. Zhang, B. Catanzaro, and W. J. Dally, "Accelerating chip design with machine learning," *IEEE Micro*, vol. 40, no. 6, pp. 23–32, 2020.

[3] R. Islam and M. A. Shahjalal, "Soft voting-based ensemble approach to predict early stage drc violations," in *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2019, pp. 1081–1084.

[4] B. Li and P. D. Franzon, "Machine learning in physical design," in *2016 IEEE 25th Conference on Electrical Performance Of Electronic Packaging And Systems (EPEPS)*, Oct 2016, pp. 147–150.

[5] R. Islam and M. A. Shahjalal, "Predicting drc violations using ensemble random forest algorithm," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: Association for Computing Machinery, 2019.

[6] W. Zeng, A. Davoodi, and Y. H. Hu, "Design rule violation hotspot prediction based on neural network ensembles," *CoRR*, vol. abs/1811.04151, 2018.

[7] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and N. Corporation, "Routenet: Routability prediction for mixed-size designs using convolutional neural network," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '18. New York, NY, USA: Association for Computing Machinery, 2018.

[8] R. Liang, H. Xiang, D. Pandey, L. Reddy, S. Ramji, G.-J. Nam, and J. Hu, "Drc hotspot prediction at sub-10nm process nodes using customized convolutional network," in *Proceedings of the 2020 International Symposium on Physical Design*, ser. ISPD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 135–142.

[9] F. Last and U. Schlichtmann, "Partial sharing neural networks for multi-target regression on power and performance of embedded memories," in *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, ser. MLCAD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 123–128.

[10] G. R. Reddy, K. Madkour, and Y. Makris, "Machine learning-based hotspot detection: Fallacies, pitfalls and marching orders," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[11] S. S. Sarwar, A. Ankit, and K. Roy, "Incremental learning in deep convolutional neural networks using partial network sharing," *IEEE Access*, vol. 8, pp. 4615–4628, 2020.

[12] K. Bhanushali and W. R. Davis, "Freepdk15: An open-source predictive process design kit for 15nm finfet technology," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ser. ISPD '15. New York, NY, USA: ACM, 2015, pp. 165–170.

Luis Francisco is currently a PhD candidate in Electrical Engineering at North Carolina State University. He received an MS degree in Electrical Engineering from the University of Puerto Rico at Mayaguez, in 2012. His research interests include EDA, Physical Verification, and Machine Learning.

Paul Franzon is currently a Cirrus Logic Distinguished Professor of Electrical and Computer Engineering at North Carolina State University. He received a PhD in Electrical Engineering from the University of Adelaide, Australia, in 1988. His research interests include, applications of machine learning to EDA, 3DIC design and Cognitive Computing.

W. Rhett Davis is currently a Professor of Electrical and Computer Engineering at North Carolina State University. He received PhD in Electrical Engineering from the University of California, Berkeley, in 2002. His research interests include EDA for integrated systems in emerging technologies, 3DIC design and system-level power-modeling.