Haoxing Ren
Jiang Hu *Editors*

Machine Learning Applications in Electronic Design Automation

Springer

Chapter 2
Deep Learning for Routability

Zhiyao Xie, Jingyu Pan, Chen-Chia Chang, Rongjian Liang, Erick Carvajal Barboza, and Yiran Chen

2.1 Introduction

With the advance of semiconductor technology, an increasing number of complicated design rules need to be followed in VLSI design, and a chip can only be taped-out after passing the design rule checking (DRC). This basic requirement is often difficult to be satisfied in modern chip design, especially when routability is not adequately considered in early design stages. In light of this fact, routability is widely recognized as a main objective in placement, and routability prediction has received considerable attention in both academic research and industrial tool development. Routability prediction at early design stages can help designers and tools perform preventive measures so that design rule violations can be avoided in a proactive manner.

In some industrial design flows, fast trial global routing is often employed for overall routability prediction at the placement stage. However, this is still too slow from the routability prediction point of view, as it needs to be invoked many times in the placement process. In comparison, even the full-fledged global routing is not accurate enough to identify precise locations of DRC hotspots, due to complicated design rules imposed upon design layout for manufacturing. Overall, existing

Z. Xie (✉)
Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong
e-mail: eezhiyao@ust.hk

J. Pan · C.-C. Chang · Y. Chen
Duke University, Durham, NC, USA
e-mail: jingyu.pan@duke.edu; chenchia.chang@duke.edu; yiran.chen@duke.edu

R. Liang · E. C. Barboza
Texas A&M University, College Station, TX, USA
e-mail: liangrj14@tamu.edu; ecarvajal@tamu.edu

H. Ren, J. Hu (eds.), *Machine Learning Applications in Electronic Design Automation*, https://doi.org/10.1007/978-3-031-13074-8_2

routing-based solutions are neither fast enough for overall routability forecast nor accurate enough for pinpointing DRC hotspots.

In recent years, machine learning (ML), especially deep learning (DL), has been demonstrated as a powerful technique in many fields. In VLSI design, many DL-based routability estimators have achieved promising results. Unlike traditional algorithms, these data-driven techniques avoid constructing solutions from scratch and achieve orders-of-magnitude acceleration by directly learning complex correlations from prior data. Compared with traditional ML methods, DL-based methods are superior in capturing the global information from a larger region of the circuit layout. In this chapter, we will cover DL-based routability estimators in detail, from the background to the latest research efforts, with more focus on feature engineering and DL model architecture design. In addition, we will also cover existing explorations on the efficient deployment of these routability estimators in the physical design flow to benefit the final chip quality.

2.2 Background on DL for Routability

2.2.1 Routability Prediction Background

2.2.1.1 Design Rule Checking (DRC) Violations

In practice, layout routability may be evaluated with different metrics. The most accurate, ultimate, and widely adopted measurement of routability is design rule violation (DRV). DRC verifies whether a specific layout meets the constraints derived according to manufacturing process requirements. Such checking of design rules is an essential part of physical design flows. Although design rules cannot guarantee success for manufacturing a design, their violations definitely make the manufacturing more difficult and increase the failure rate.

Design rules are provided by process engineers and/or fabrication facilities. Each process technology has its own set of rules, commonly defined in a DRC rule deck file. The number and complexity of DRC rules increase as the transistor feature size shrinks at advanced technology nodes. As a result, routability prediction for different technology nodes may require essentially different methods and ML models. Some basic and common types of DRC rules include minimum width, minimum spacing, minimum area, wide metal jog, misaligned via wire, special notch spacing, end of line spacing, etc. Most design rules can be broadly categorized into three types:

- Size rules. They define the minimum length or width of components/shapes in a layout.
- Separation rules. They define the minimum distance between two adjacent objects at each layer.

- Overlap/enclosure rules. They define the minimum amount of overlap/coverage between two connected shapes in two adjacent layers.

Background Information
DRC violation (DRV) can be accurately evaluated based on rule decks after routing, using physical verification sign-off tools. Widely used commercial tools include Siemens EDA (formerly Mentor Graphics) Calibre®, Cadence Pegasus™ and PVS, Synopsys IC Validator™, etc. Before the sign-off stage, some digital layout tools like Cadence Innovus™ and Synopsys IC Compiler™ II also provide quick design rule checking after routing. These tools can mark the precise locations with DRV and return the total number of DRVs in the whole layout.

2.2.1.2 Routing Congestion and Pin Accessibility

Although DRC violation is the ultimate optimization goal, it is not the only metric to measure routability. In practice, routability is also approximated with the routing congestion at early design stages like global routing. For each region on the layout, congestion measures the gap between the demand of routing resources and the supply. This measurement is based on the routing model. In a basic routing model, the entire layout is tessellated into an array of global routing cells (gcells), also referred to as grids or bins. They are further gridded using horizontal and vertical gridlines, referred to as routing tracks, along which wires can be created. Each gcell can only accommodate only a finite number of routing tracks, and the number of available tracks is referred to as its routing *supply* or *capacity*. In comparison, the routing *demand* is contributed by each wire crossing the gcell, requiring one routing track in either horizontal or vertical direction. The congestion of each gcell is measured by the number of excess tracks in routing *demands* over the *supply*, referred to as track overflow. Such overflow for horizontal and vertical tracks is calculated separately and is set to zero if supply is larger or equal to demand. Routers also report the overall percent of gcells with overflow in an entire layout, as an indicator of the overall congestion. The existence of routing congestion often results in detoured wires, poor layer assignment, or even incomplete routes containing opens and shorts. It is viewed as one major contributor to DRC violations.

Compared with DRC violation, routing congestion is much easier to estimate based on placement and routing solutions. In some industrial tools, routing congestion can be measured during global routing (GR), or even at early global routing (eGR), also named trial routing (TR). Thus, it is commonly used as an early measurement of layout routability, and many traditional routability improvement techniques are based on it. However, many studies [5] have demonstrated the mis-correlation between routing congestion and DRC violations, especially at advanced technology nodes when the complexity of design rules increases. This also makes routability improvement increasingly difficult.

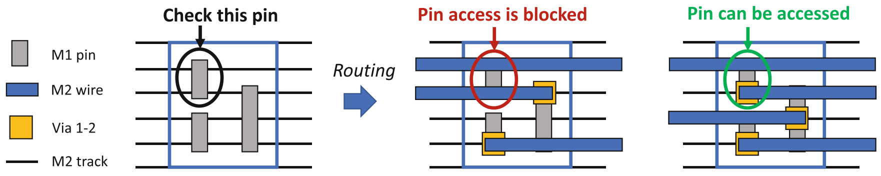


Fig. 2.1 Pin accessibility example: the detailed routing around pins in a standard cell [9]

At advanced technology nodes like sub-20nm, as design rules keep increasing, pin accessibility becomes another key contributor to DRC violations. As Fig. 2.1 demonstrates, the problem arises from the difficulty to route to standard cell pins on low-level metal layers, even when the routing congestion is low. At an advanced node, even a standard cell designed for easy pin access may not be easily routable if surrounded by other cells, which restrict wire access to its pins. As indicated in previous works [30], the pin access problem is partially contributed by high pin densities in a local region, but they are not strictly correlated.

2.2.1.3 Relevant Physical Design Steps

Now, we inspect the routability problem in a design flow, which typically starts with a design in register-transfer level (RTL). After logic synthesis tools convert design RTL into a gate-level netlist, physical design is performed, where all design components, including macros, cells, and wires, are instantiated with concrete geometric representations on metal layers. Routability problems and the majority of routability estimations arise at this stage. A typical physical design flow consists of several major steps, including floorplanning, power planning, placement, clock tree synthesis (CTS), routing, and physical verification. Each physical design step may involve multiple sub-steps. For example, placement includes global placement (GP) and detailed placement (DP). Routing includes global routing (GR) and detailed routing (DR). In addition, optimizations are performed after major steps including placement, CTS, and routing.

Background Information
All routing blockages and different types of wires, including power grids, clock wires, and signal nets, all share the common area resource in a layout. Thus, decisions at many layout steps directly affect routability, and there exist multiple trade-offs between routability and other design objectives. During floorplanning, a higher utilization rate for smaller area directly leads to less routing spaces and worse routability. During power planning, if more routing resources are devoted to power grids, the design suffers less from IR drop

(continued)

violations but results in worse routability. During CTS, a clock tree can achieve better skew and clock tree capacitance with wire sizing but requires more space and often leads to additional DRC violations. Lastly, if routability is not considered and optimized during floorplanning and placement, the layout may achieve better wirelength, timing, and power, at the cost of more DRC violations. In summary, improving routability is not a stand-alone design problem. Better routability not only helps secure DRC clean at physical verification but also allows trade-offs for improvement on other design objectives according to specific design goals.

2.2.1.4 Routability Prediction

As mentioned, DRC violations can only be precisely measured after routing finishes, when the room for fixing DRV has become very limited. Besides fixing DRV manually, one option is to perform engineering changing order (ECO), which tries to complete unrouted and partially routed nets while maintaining existing wires as much as possible. Another fixing method is to delete part of existing wires and reroute them. But it is difficult for these minor modifications at the post-routing stage to fix all violations for layouts with poor routability. As a result, designers have to trace back to earlier stages, change their layout solution accordingly, and start a new design iteration. It can take many iterations to reach a DRV-clean layout, leading to a very long turnaround time.

To improve layout routability, design rule violations should be avoided with preventive measures in a proactive manner. This heavily relies on early routability prediction methods. However, accurate early routability prediction is difficult since the behavior of placement and routing engines in modern EDA tools is highly complex and rather unpredictable. One possible solution is to develop some fast trial routing algorithms, but it is hard to achieve ideal accuracy and speed at the same time. Another promising research direction nowadays is to learn from prior data by developing data-driven routability estimators with machine learning (ML) algorithms. A main strength of ML methods is the automatic extraction of complex correlations between separated design steps based on prior knowledge. Once the ML model has been trained, it can produce routability predictions in a very short time, without constructing solutions from scratch.

For routability prediction, researchers have tried different prediction granularities according to their application scenarios. Figure 2.2 shows two common routability prediction scenarios with different granularities. Some coarse-grained predictions only evaluate the overall routability of an entire layout. Such routability is usually measured with the total number of DRC violations, also named DRV count. Generally, it is easier to achieve DRV clean for layouts with less DRVs. Another similar metric is the total number of nets with DRC violations, also named violated

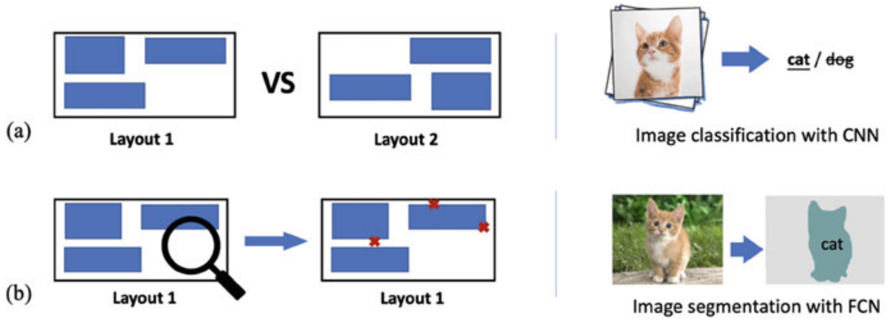


Fig. 2.2 Two common routability prediction scenarios. (**a**) Use coarse-grained prediction on the overall routability of an entire layout to identify more routable layouts among candidates. (**b**) Use fine-grained prediction on detailed DRV locations to guide mitigation techniques

net count. Since the same net may cause multiple DRC violations, using violated net count instead of DRV count avoids counting the same problem repeatedly. Such a coarse-grained prediction evaluates the whole layout and enables the identification of more routable layout solutions among many candidates. In comparison, fine-grained routability prediction tries to pinpoint the detailed locations with DRC violations. This guides layout modifications at early stages to proactively prevent DRC violations. Based on these predictions, many different applications of routability estimators have been proposed.

2.2.2 *DL Techniques in Routability Prediction*

The most commonly adopted DL techniques in routability prediction are from computer vision, including convolutional neural networks (CNN) and fully convolutional network (FCN) [17]. The basic idea is to process the layout input like processing an image for image classification, segmentation, and generation. There are many variations for each DL technique. For CNN models in routability prediction, popular models include ResNet [10], DilatedNet [28], and DeepLabv3 [8]. For FCN models, popular models include the vanilla FCN model [17] and the U-Net model [23]. In addition, some prior works train their FCN-based model with a conditional generative adversarial network (cGAN) [19] framework.

2.2.2.1 CNN Methods

Compared with traditional machine learning algorithms, CNN learns more abstract patterns from images. A typical structure of CNN is composed of convolutional (Conv) layers, pooling (Pool) layers, and fully connected (FC) layers. The final output is a single vector of class scores, whose length equals the number of classes. The ResNet [10] model is a classical variant of CNN model proposed in 2015. It takes advantage of scalable residual blocks to allow skipping layers. As a result, it well solved the gradient vanishing problem when the depth of CNN increases. Besides the standard convolution, another widely adopted convolution is named atrous or dilated convolution (DC) [28]. It introduces another parameter to convolutional layers called the dilation rate, which defines a spacing between the values in a kernel. A 3×3 kernel with a dilation rate of 2 will have the same field of view as a 5×5 kernel, while only using 9 parameters. Compared with basic convolution, it can effectively enlarge receptive fields of filters.

2.2.2.2 FCN Methods

Compared with traditional CNN targeting image classification, the FCN, a CNN variant without FC layers, is firstly proposed to perform end-to-end semantic segmentation. Given an arbitrary input size, it can output an image with its size equal to the input. Many FCNs [17] adopt an encoder-decoder framework. In the encoder, by downsampling operators, the depth and spatial dimensions of the feature map gradually get deeper and smaller. In the decoder, the width and height of the feature map are gradually recovered to those of the input by upsampling operators. Transposed-convolutional (Trans) layers are usually added at the decoder part to upsample feature maps and control the size of the final output. Such architecture is widely used in many computer vision problems, like crowd counting and biomedical image segmentation. Besides eliminating FC layers, many FCNs have multiple shortcuts, which concatenate feature maps in the front directly to feature maps near the end. A popular example is U-Net [23] for medical image segmentation. As a result, both longer and shorter paths exist between the input layer and the final output layer. Such multipath architecture reserves both shallow and deep embedding information.

2.2.2.3 GAN Methods

Generative adversarial networks (GANs) are used in unsupervised tasks. A GAN consists of a generator G and a discriminator D. The discriminator D distinguishes between samples generated from the generator and samples from the training dataset. The generator G generates a mapping of input samples that cannot be distinguished by the discriminator D. In the conditional GAN (cGAN) [19], ordinary GAN is extended to a conditional model by conditioning both the generator and discriminator with some extra information, which could be any kind of auxiliary information, such as class labels or data from other modalities [19].

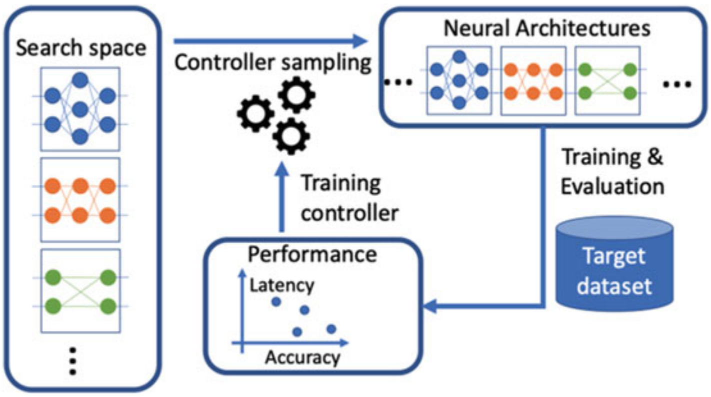


Fig. 2.3 A basic NAS framework

2.2.2.4 NAS Methods

A recent study in routability prediction explores the automated development of estimator structures with very little designer expertise or human effort. It utilizes the popular neural architecture search (NAS) technique [21]. NAS automatically conducts architecture engineering to find effective neural network models for specific tasks without (or with minimum) human interventions. It has demonstrated great potential in applications like image classification, object detection, and semantic segmentation. Figure 2.3 shows a basic NAS framework. It consists of three key ingredients: *search space, evaluation strategy*, and *search strategy*. The *search space* defines a family of candidate architectures that can be explored in NAS. The *evaluation strategy* determines the metric to estimate the design quality (e.g., accuracy) of a candidate architecture and provides feedback to the search process. The *search strategy* is the method to explore the search space and guide the search process toward the choice of a promising ML model.

2.2.3 Why DL for Routability

Data-driven routability estimators are initially constructed with traditional ML models, including support vector machine (SVM)-based estimator [4, 5], multivariate adaptive regression spline (MARS)-based estimator [4, 31], and artificial neural network (ANN)-based estimator [25, 26]. These ML models typically only process a limited number of input features. For fine-grained routability prediction on DRV locations, traditional ML methods are applied to make decisions based on a small cropped region with limited features from the layout. Such a small input region strongly limits the *receptive field* (the field visible to the model) of these traditional ML methods. As a result, they cannot capture the *global information* from a larger

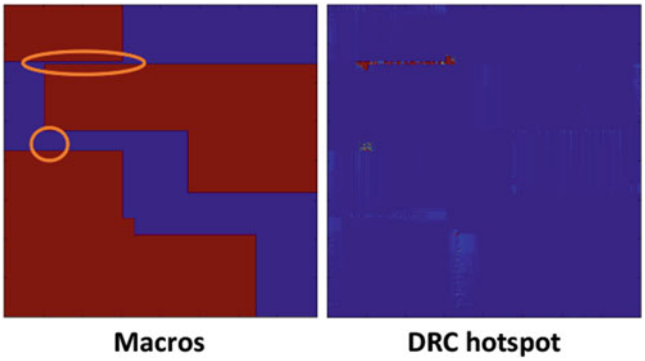


Fig. 2.4 Macros and DRC hotspot distribution. Orange circles indicate regions with high DRVs

region of the layout. For example, the nets spanning a large region have a large impact on the routability. Another important global impact is the tendency for DRV hotspots to aggregate in the space between adjacent macros. This is illustrated in Fig. 2.4. A cropped region is usually too small to capture the impact from neighboring macros. In addition, routability can also be affected by clock wires and power grids, which may also be captured by a large receptive field.

DL models are naturally good fits to capture the global information from large regions. In computer vision, various CNN/FCN models have been designed to identify the objects or semantic of images by capturing the pattern of a whole image. Many routability estimators [6, 7, 11, 12, 15, 18, 20, 27, 29] utilize these DL techniques to process placement or routing solutions like images. Also, DL models have strong abilities to process the interactions among different features/channels, like the RGB colors in different channels. For routability prediction, researchers can extract many relevant features for routability based on their own intuitions and then feed these features into DL models as different channels.

Notice that it is actually not necessary for DL-based models to take the whole layout as input. If developers choose to take cropped regions as input, DL models can still support a much larger cropped region as input compared with traditional ML models.

Here, we provide a simple case study on fine-grained DRV location prediction, assuming F different features are defined for each method. The whole layout is tessellated into $w \times h$ grids. More details of such setup will be introduced in the next section on methodologies.

- For traditional ML-based methods [31] without neighboring information, to make predictions on each 1×1 grid, the F features are measured only

(continued)

on this grid; thus, the model input is a vector with length F. The receptive field is 1×1 grids.

- For traditional ML-based method [5] with both small (1×1 grid) and large (3×3 grids) measurement windows, there are F features collected with the small window and another F features with the large window. The model input will be a vector with length $2F$. The receptive field is 3×3 grids.
- For traditional ML-based method [25] with features on neighboring grids calculated separately, for a cropped region with 3×3 grids, the model input is a vector with length $9F$. The receptive field is 3×3 grids.
- For DL-based methods applied on the whole layout, the raw input is in the shape of $w \times h \times F$. Their receptive fields are equal to or smaller than $w \times h$ grids and can be calculated [2] based on actual model structures.

This case study verifies our previous claim that DL-based estimators can process more inputs and support a much larger receptive field.

2.3 DL for Routability Prediction Methodologies

We will introduce the general flow for routability prediction with DL methods in this section. It includes four major steps: data generation and augmentation, feature engineering, DL model architecture design, and model training and inference. The prediction flow itself can be stand-alone, without being part of placement or routing algorithms. The majority of previous routability prediction methods are applied after the floorplanning stage, including post-global placement, post-detailed placement, and post-global routing. When applied at later stages, more layout design decisions have been made; thus, more features are available and the prediction can be more accurate. As a trade-off, there is less room for DRV mitigation at later stages. Before floorplanning, in contrast, no layout information is available, and the design has to be processed as a graph with graph neural network (GNN) based on the gate-level netlist, instead of an image. Very few works [14] target such pre-layout stage due to the difficulty. Except for this rare case, the general flow for routability prediction based on layouts is introduced step-by-step in this section.

2.3.1 Data Preparation and Augmentation

The model construction process starts with data generation. Designers collect representative circuit designs and then go through the physical design process with EDA tools to generate training data. In this process, relevant raw data is dumped out for feature and label extraction and preprocessing.

Before starting the data generation process, designers need to decide whether the model targets *cross-design*, according to their application scenario. A cross-design ML estimator means the estimator directly applies to new designs that are not in the training set. Usually, researchers require the new design to be different from training designs at the netlist level. Thus, the following examples are not viewed as cross-design: (1) models trained and tested with different layout implementation of the same netlist and (2) models trained and tested on multiple designs, which appear in both training and testing sets. Most representative routability estimators are cross-design, which means they do not need model retraining or fine-tuning for new designs. In this case, the only prediction cost is the short model inference time, which is the main advantage of performing predictions with ML models. However, it is more challenging for cross-design models to achieve high accuracy on all new designs never seen by the model, especially for new designs that are largely different from the training data. Notice that, although cross-designs support training and testing on different designs, almost all estimators require the same technology library used for training and testing designs. For different technology nodes, the complex correlation between features and routability can be quite different, preventing the model from learning a more generalizable pattern.

To construct a typical cross-design ML model, designers start with collecting representative circuit designs and constructing a training dataset based on them. Many prior works perform their experiments on different private industrial benchmarks. This makes the direct comparison among different solutions very difficult. One popular public benchmark is from the ISPD'15 detailed-routing-driven placement contest [3]. Based on these circuit designs, the physical design process can be finished with commercial layout tools, which are viewed as the source of ground truth. All raw data related to features and labels are collected and dumped out at corresponding design stages.

To construct high-performance DL-based estimators, a sufficient amount of training data is necessary. Take those widely used datasets in computer vision as an example, CIFAR-10 and MNIST contain 50 thousand and 60 thousand training images, respectively. The larger-scale dataset ImageNet contains more than 14 million images. In comparison, it is challenging to generate as much training data in routability prediction. The challenge from limited data is twofold. First, it is difficult to collect a large number of different circuit designs for training data generation, especially for researchers in academia. Second, even if many circuit designs can be collected, generating a large amount of labeled data can be highly time-consuming. The generation of each label requires going through the standard physical design flow once. As a result, routability estimators are usually trained with a limited amount of training data. Currently, there is no consensus on the minimum amount of data to use for training, considering the largely different scenarios in different works. But as a rule of thumb, it is recommended that at least hundreds of layouts from several different designs should be generated for training cross-design estimators.

Although we face challenges from limited training data, there are several operations that can enlarge the size of the training dataset. For the same circuit design, multiple netlists may be generated by providing different parameters to the logic synthesis tool. Similarly, for the same netlist, multiple layouts can be generated. For designs with macros, changing the locations of macros can very efficiently generate many essentially different layout solutions. In addition, based on the same layout solution, researchers can perform data augmentation on-the-fly during training, which is well supported by current deep learning techniques. This on-the-fly augmentation avoids enlarging the storage requirement of the training dataset. The most common augmentation techniques for routability problem include horizontal flipping, vertical flipping, and rotation by 180°. Some researchers suggest more aggressive augmentation techniques like rotation by 90°, 270°, random cropping and padding, random cropping and scaling, etc. However, these augmentation techniques tend to create representations of unrealistic layouts and thus are not very reasonable in this routability prediction problem and should be used very carefully.

Based on the generated raw data, input features X and labels Y are generated in the subsequent data preprocessing stage. Just like the basic routing model in EDA tools, for routability prediction, the whole layout is firstly tessellated into a two-dimensional matrix of equal-sized grids/tiles, with w grids in the width and h grids in the height. A visualization of the label extraction process is shown in Fig. 2.5. Notice that this step is not directly related to the gcell or the routing algorithm, and it is up to the designers to decide the grid size in tessellation. There is a trade-off between the granularity and the computation cost. A finer-grained tessellation with a smaller grid size allows more detailed identification of DRV hotspot locations, but at the expense of a higher computation cost during preprocessing, training, and inference.

After the layout tessellation, each feature or the label of the layout is extracted as a two-dimensional density distribution map. Details of extracted features will be introduced in the next subsection. To calculate this, for each grid on the tessellated layout, we measure the feature value in this grid as a real number. Then, each feature of the whole layout with $w \times h$ grids is in $\mathbb{R}^{w \times h}$. Assuming altogether there are F different input features, they can be calculated independently and then stacked together as one input tensor $X \in \mathbb{R}^{w \times h \times F}$. Similarly, for the label collection, assuming we are performing DRV hotspot detection, the label can be extracted in the same way and denoted as $Y \in \mathbb{R}^{w \times h}$. Sometimes designers are only interested

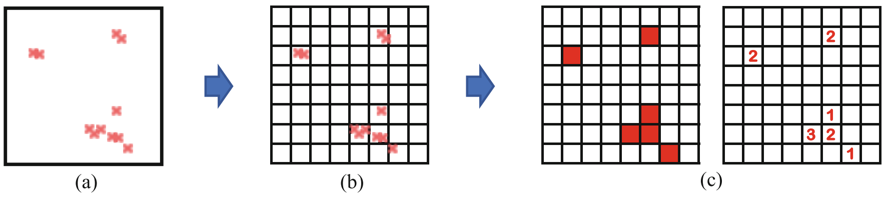


Fig. 2.5 Visualization on the feature and label extraction. (**a**) The layout with DRC violation distributions as labels. (**b**) The tessellation of the layout with $w \times h$ grids. (**c**) The label on DRC violation distribution extracted as $Y \in \mathbb{R}^{w \times h}$ for regression (right) or $Y' \in \{0, 1\}^{w \times h}$ for binary classification (left)

in whether there is any violation at each grid, so this task degrades to a binary classification problem for each grid, and the label is $Y' \in \{0, 1\}^{w \times h}$. This process is also shown in Fig. 2.5. For DRV count prediction, the label is summation of all numbers of violations, $y = \text{sum}(Y) = \sum_{j=1}^{w} \sum_{i=1}^{h} Y[i][j] \in \mathbb{R}$.

2.3.2 *Feature Engineering*

Feature engineering plays a key role in DL for routability prediction methodologies since it determines the upper limit of the performance of ML methods. If certain key information is missing in input features, it is almost impossible for the ML model to learn the corresponding correlation and make accurate predictions. Notice that available features depend on the stage we apply the model. For example, if the model targets to be applied before placement, then only macro locations can be included as features, while cell locations are unknown yet. Figure 2.6 shows the basic physical design flow with available features and labels at each design stage. The selection of features heavily relies on designers' expertise in both physical design and deep learning, and the solution is specific to this routability prediction problem. After years of exploration, there are multiple fundamental features that are recognized by most EDA researchers and engineers. They can be roughly categorized into several types. We introduce these features in detail below.

2.3.2.1 Blockage

A routing blockage defines a region where routing is not allowed on specific layers. This is often considered as a hard constraint and directly affects the final design routability. A higher density of blockages naturally leads to poor layout routability. The blockage information includes locations of macros, cells, and pins on the layout. To extract more information for the ML engine, different types of blockages can be captured separately into different two-dimensional density distribution maps. Below are some of the most commonly adopted blockage-related features:

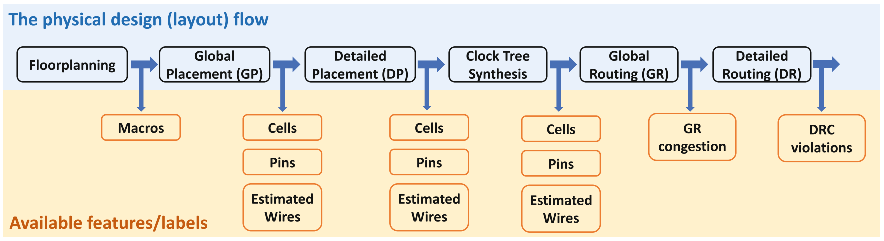


Fig. 2.6 A basic physical design flow with available features and labels at each design stage

- Macro density. After floorplanning, the locations of all macros are fixed. The regions occupied by macros are included in features.
- Cell density. Cell locations are mostly fixed after detailed placement. The density distribution of cells is included in features.
- Pin density. Besides macros and cells as blockages, the density distribution of all pins from both macros and cells is included in features.

Besides these widely adopted features, other features that may be useful include:

- Decomposed cell density. In some works, to capture more information, the cell density of different types of cells is extracted separately as different features. For example, densities of flip-flop cells, clock tree cells, and fixed cells that cannot be moved can be additional features besides the density of all cells.
- Decomposed pin density. Similar to cell density, pin density can also be decomposed into multiple features, including macro pins at each metal layer, the pins of flip-flop cells, and the pins of cells on the clock tree.
- Horizontal/vertical track capacity maps. As mentioned in the background, routing congestion depends on the gap between the *demand* of routing tracks and the *supply*. As a result, the density map of routing track *supply* can also be a useful input feature. Track capacity is measured separately in two directions: horizontal and vertical. Thus, we have separate features for the two directions.
- Detailed pin configurations. As mentioned, pin accessibility problems can cause DRC violations. In some works [15] targeting advanced technology nodes, the detailed pin patterns/configurations are captured specifically with "high-resolution" as input features.

2.3.2.2 Wire Density

In each region, a higher wire density will lead to higher demand on routing tracks and thus higher routing track overflow, more routing congestions, and likely more DRC violations. To capture this effect, wire density information is an essential input to routability estimators. Compared with blockage information, which is directly available after locations of components are fixed by placement, the wire distribution on a layout cannot be known explicitly until routing finishes, when it is too late to perform routability prediction. As a result, the extraction of wire density itself requires some estimations, and various solutions have been proposed. Below are some commonly adopted wire density-related input features. The most widely used feature is RUDY. Figure 2.7 shows visualizations of multiple features.

- RUDY. Rectangular uniform wire density (RUDY) [24] is derived by the total uniform wire density spreading in the bounding box of a net. The wirelength of each net is estimated by the half-perimeter wirelength (HPWL) of the net bounding box. A higher RUDY indicates a higher wire density. At each location (x, y) on the layout, the RUDY contributed by the ith net is:

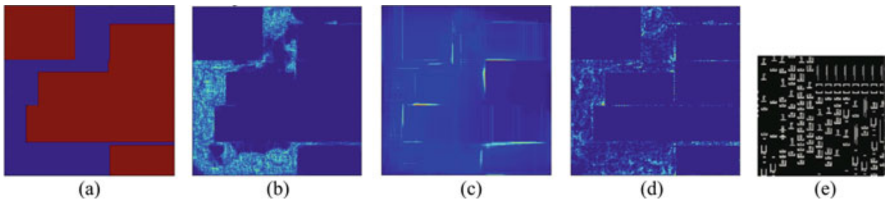


Fig. 2.7 Visualization of some common extracted features. (**a**) Pin density. (**b**) Macro density. (**c**) Long-range RUDY. (**d**) RUDY pins. (**e**) Detailed pin configurations in high resolution (from part of the layout)

$$\mathrm{RUDY}^i(x, y) = \begin{cases} \mathrm{HPWL}(i)/\mathrm{area}(i) & \text{if } (x, y) \text{ is in the bounding box of } i\text{th net} \\ 0 & \text{Otherwise} \end{cases}$$

where area(i) is the bounding box area of the ith net. The final RUDY at location (x, y) is the summation of contributions from all nets in the design.

$$\mathrm{RUDY}(x, y) = \sum_i \mathrm{RUDY}^i(x, y)$$

RUDY is originally proposed as a pre-routing congestion estimator [24] and is first adopted as an input feature in the work of [27].

- Short/long-range RUDY map. The original RUDY feature can be decomposed into long-range and short-range RUDY. Long-range RUDY is from nets covering a distance longer than a threshold. Similarly, short-range RUDY is for nets shorter than this threshold. According to the work of [27], there is a stronger correlation between long-range RUDY and DRV than short-range ones.
- Horizontal/vertical net density map. Similar to long-range RUDY, these two features estimate how many nets are expected to go through each grid horizontally and vertically [7]. The probability of the grid being routed through by a net will be evenly distributed to each grid in a column or a row in the net bounding box. Thus, in the horizontal (vertical) direction, all grids covered by the same net will receive the same density from this net, which equals one divided by the number of covered grids in this row (column).
- RUDY pins. It is similar to pin density, while the contribution of each pin to the density map now equals the long-range RUDY of the net connected with it.

In addition to these methods, some works adopt a heuristic named fly line, which is a line that connects pins. It is also referred to as flight line in some previous works [6]. There are different types of fly lines that may reflect wire congestions. Figure 2.8 demonstrates examples of these wire density features, and some explanations are given below:

- Pair-wise fly lines. For each net, the pair-wise fly lines are connected from each pin to all the other pins of the same net.

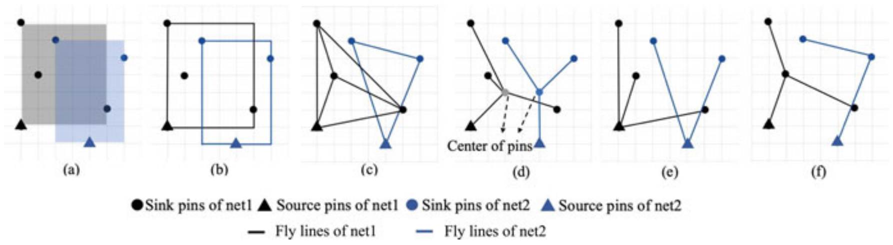

Center of pins
(a)
(b)
(c)
(d)
(e)
(f)
● Sink pins of net1 ▲ Source pins of net1 ● Sink pins of net2 ▲ Source pins of net2
— Fly lines of net1 — Fly lines of net2

Fig. 2.8 Wire density feature examples. (**a**) RUDY. (**b**) Bounding box. (**c**) Pair-wise fly lines. (**d**) Star fly lines. (**e**) Source-sink fly lines. (**f**) MST fly lines

- Star fly lines. For each net, the star fly line connects each pin to the center of the pins in the net.
- Source-sink fly lines. In timing optimization, tools tend to connect sinks with the source through the shortest path. To capture this effect, source-sink fly line connects the source pin with all sink pins in the same net.
- MST fly lines. The three aforementioned fly line features tend to overestimate the routing demand. In traditional routing, the minimum spanning tree (MST) is an effective algorithm to guide the router. Thus, the fly line connects all the edges in its MST.

2.3.2.3 Routing Congestion

As introduced, the routing congestion of a layout is a good, although not perfect, indicator of routability and DRC violations. Thus, estimations on congestion are very useful input features. Besides being a feature, it is also used as the prediction label in many prior works. There are two types of congestion reports that can be generated by commercial layout tools, as shown below:

- Trial routing (TR) congestion. After detailed placement, the trial global routing, also denoted as trial routing, can be performed. It produces an estimation of routing congestion, named TR congestion.
- Global routing (GR) congestion. Compared with trial routing, the full-fledged global routing can generate a more detailed congestion map, denoted as GR congestion.

2.3.2.4 Pin Accessibility

Besides congestion, pin accessibility is another main cause of DRC violations, especially at the advanced technology nodes. As a result, DRVs may correlate with pin shapes and the proximity relationship among pins. Compared with other input features, such fine-grained pin patterns at the advanced node are difficult to be directly quantitatively measured when using large grids, with each grid containing multiple pins. To solve this, one option is to use high-resolution "images" showing the pin patterns/configurations as input features [15]. However, such a high resolution may lead to a higher computation cost in the routability estimator.

2.3.2.5 Routability Label

Besides all features mentioned above, we also discuss different types of labels used in routability prediction in this subsection.

- All DRC violations. Most prior works directly predict all DRC violations in the layout. Some works use accurate DRC violations from sign-off validation EDA tools, while others adopt fast DRV estimations from digital layout tools after detailed routing finishes.
- Part of DRC violations. Some prior works choose to only predict certain types of DRC violations (like low metal layer short violations) in order to improve the layout flow for better routability.
- Routing congestion. Considering the complexity and difficulty in DRV prediction, many works choose to only predict the routing congestion for better accuracy, especially for models targeting to be applied at earlier design stages.

2.3.3 DL Model Architecture Design

As mentioned, the most widely used DL models for routability prediction are CNN- and FCN-based methods, targeting coarse-grained and fine-grained routability prediction tasks, respectively. We use the term "FCN based" broadly to include all DL models with both downsampling and upsampling structures. In this part, we will introduce the model architecture design. It starts with the common operators and connections, which are the basic building blocks of estimators. After that, we introduce representative prior works as case studies to demonstrate the design of different routability estimators.

2.3.3.1 Common Operators and Connections

We start with the detailed introduction of operators used in CNN/FCN methods for routability prediction. The fundamental operators include convolutional layers, pooling layers, and fully connected (FC) layers. CNN performs downsampling before the FC layers in order to enlarge its input receptive field and capture global information of the whole circuit layout. The downsampling in CNN-based routability estimators is usually achieved by either a maximum pooling layer or a convolutional layer with stride equal to 2. The stride is a parameter of the

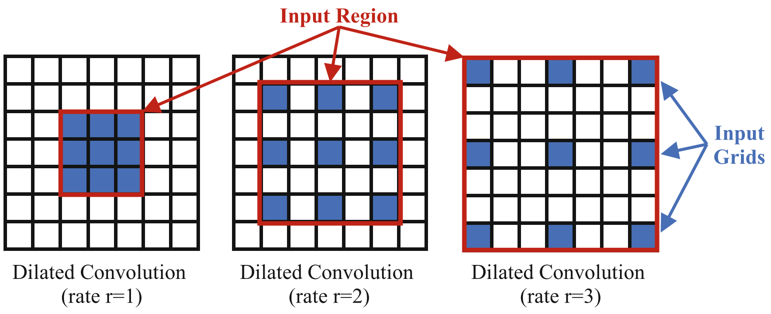
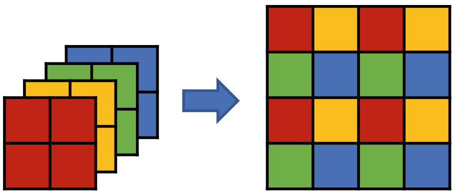

Input Region
Input Grids
Dilated Convolution (rate r=1)
Dilated Convolution (rate r=2)
Dilated Convolution (rate r=3)

Fig. 2.9 The input regions of atrous or dilated convolution with rate $r = 1, 2, 3$ for a 3×3 kernel

Fig. 2.10 Visualization of sub-pixel upsampling (Sub) operation. It is adopted in the routability estimator PROS [7]



neural network's filter that modifies the amount of movement over the input. Many researchers prefer the latter option for downsampling since it allows one more convolutional operation. After the downsampling, the output is flattened and processed by multiple FC layers, which generate one vector or scalar as the prediction. To evaluate the overall routability, CNN-based estimators perform binary classification or regression.

In addition to the standard convolutional layer, the atrous convolution, also known as dilated convolution, is constructed by inserting zeros between each value in the kernel. An atrous convolution with dilation rate r would insert $r - 1$ zeros between adjacent values in the kernel. A visualization of atrous is shown in Fig. 2.9. When rate $r = 1$, it degrades to a regular convolution operator. This operator is designed to further increase the receptive field at the cost of higher memory usage. It is commonly seen in image segmentations and starts to be adopted for routability prediction in recent years [7].

Compared with CNN, FCN performs both downsampling and upsampling, or named encoding and decoding, in order to achieve a two-dimensional output as the fine-grained prediction on DRC violation distribution. The downsampling process is performed with the front half of the FCN structure, which is usually very similar to the CNN model with convolutional and pooling layers. After downsampling, the upsampling is commonly performed with transposed-convolutional layers in the latter half of the FCN structure. In addition to transposed convolution, the other common choice of upsampling is the sub-pixel upsampling block (Sub). It rearranges the elements of $W \times H \times r^2C$ tensor to form a $rW \times rH \times C$ tensor with pixel shuffling. Figure 2.10 provides a simple example, where $W = 2, H = 2, r = 2, C = 1$, converting a tensor in $2 \times 2 \times 4$ to a tensor in $4 \times 4 \times 1$. This operation adds no extra trainable parameters and takes less computation cost. Notice that standard convolutions are also applied during the upsampling process, and the width and height of the final output are commonly designed to be the same as the input.

These widely adopted operations need to be connected to build the routability estimator. Besides the standard connections between every two neighboring layers, the *shortcut* or *skip connection* is commonly adopted in CNN-/FCN-based routability estimators. Such shortcuts can be roughly categorized into short-range and long-range shortcuts. The short-range shortcut can be achieved with structures like the residual block (resBlock) in ResNet [10]. It adds the input value of this residual block to the output. The addition is element-wise and thus requires exactly the same dimension between the two ends of the shortcut. The size of a residual block typically contains only two convolution layers, limiting the range of such shortcut. The long-range shortcut can be based on structures like U-Net. It applies to FCN-based estimators, which concatenate or add feature maps in the front (before downsampling and upsampling) directly to feature maps near the end (after downsampling and upsampling). In this way, it supports a combination of both shallow and deep feature maps, and the successive layers can learn from the feature mixture. If it performs feature map concatenation instead of addition, it only requires the same width and height between the two ends of shortcut, without limiting the dimension in channels.

These are the basic building blocks of routability estimators; then, we introduce representative prior works as case studies below. The visualizations of these estimator structures are shown in Fig. 2.11.

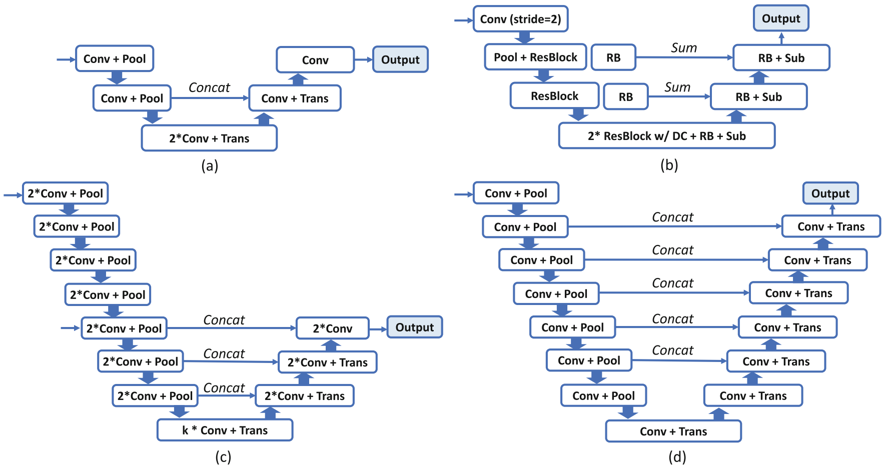


Fig. 2.11 Model structures for fine-grained DRC hotspot location detection as case studies. (**a**) RouteNet [27] structure. (**b**) PROS [7] structure. DC stands for dilated convolution, RB stands for refinement block, and Sub stands for sub-pixel upsampling block. (**c**) J-Net [15] structure. (**d**) Painting [29] generative framework structure

2.3.3.2 Case Study: RouteNet [27]

The work RouteNet [27] proposed two different models, one for the overall routability prediction and the other to pinpoint the DRC hotspots. A CNN-based model is adopted to predict the overall routability in RouteNet. It directly adopts the 18-layer ResNet but replaces the output layer to produce a single-scalar prediction, indicating four different levels of routability. In addition, this 18-layer ResNet is already pretrained on the ImageNet. Thus, the training process can be viewed as transfer learning with data about routability. This model is applied after global placement.

In comparison, to pinpoint the DRC hotspots, RouteNet proposes an FCN-based model with six convolutional layers, two pooling layers, two trans-convolutional layers, and one long-range shortcut structure. According to its ablation study, using fewer convolutions, removing the shortcut, or removing pooling/trans-convolution will all lead to accuracy degradation. This model is applied after global routing.

2.3.3.3 Case Study: PROS [7]

The work of PROS [7] proposed a more complex FCN-based model to pinpoint GR congestion locations before routing. Compared with DRC hotspot detection, this is also a fine-grained prediction task but easier to obtain a higher accuracy. In the downsampling (encoding) part of its model, similar to RouteNet [27], it directly adopts a whole pretrained ResNet network and then performs transfer learning. To avoid too much reduction in feature map size while maintaining the receptive field, some standard convolutions with stride 1 or 2 are replaced by dilated convolution (DC) with stride set to 1 and rate set to 2 or 4.

In the upsampling (decoding) part of its model, it adopts the sub-pixel upsampling blocks (Sub) to upsample and recover the fine-grained predictions. In addition, it introduces refinement blocks (RB), which are quite similar to residual blocks in ResNet, to perform convolutions on the feature maps.

2.3.3.4 Case Study: J-Net [15]

The work of J-Net [15] proposed an FCN-based model to pinpoint DRC hotspots before routing. Compared with previous works, it is targeted to the advanced sub-10nm process nodes, where pin accessibility becomes a major contributor to DRC violations. The main model structure is also FCN based with three long-range shortcuts. To deal with the pin accessibility problem, it adopts "high-resolution" pin configuration images as extra inputs and designs extra downsampling structures to extract pin accessibility information from these pin configuration inputs.

2.3.3.5 Case Study: Painting [29]

The work of Painting [29] adopted a training framework based on cGAN for routability prediction on FPGA. The generative network structure is an FCN-based model with five long-range shortcuts. This FCN-based model also generates predictions on routing congestion locations. But instead of directly optimizing this FCN-based generative network like the prior works, it is trained under the cGAN framework together with a CNN-based discriminator network. The generative network is trained to produce predictions similar to labels such that it confuses the discriminator network. Currently, the difference in performance between this cGAN-based training method and common FCN-based methods still remains rather unclear. For routability predictions on FPGA, we also observe some research efforts [1] adopting such cGAN-based framework and training methods.

2.3.3.6 Case Study: Automated Model Development [6]

Most aforementioned routability models are very carefully designed and achieve good results on their own benchmarks. However, these very well-designed methods usually take a long development time and high engineering efforts. A recent work proposes to avoid this cost by automating the model development step by neural architecture search (NAS) methods [6]. It supports a large search space allowing various types of operations and highly flexible connections. Candidate operations include standard convolutions, atrous convolutions, and mixed depth-wise convolutions. In the search space, a model is represented by a graph. Specifically, vertices represent operations, and edges indicate the directed connections of operations. Its search space is shown in Fig. 2.12. As for the search strategy, it samples edges from multiple completely ordered graphs with adjustable probabilities defined on each component of the graph. After each sampling, the sampled model is trained and evaluated on the validation set. Then, the evaluation result is used to update

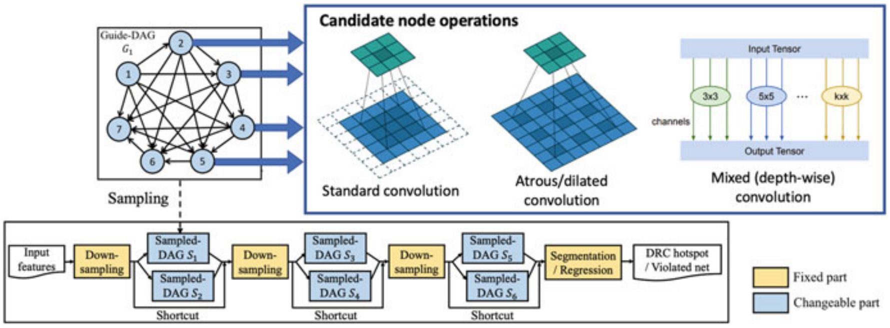


Fig. 2.12 The graph-based search space for routability estimators, in the work of [6]

the sampling probability. The basic idea is if the performance is good, then the probability of corresponding structures being sampled again becomes higher.

Compared with human-developed routability estimators, the automated-developed estimators construct much more parallel branches and flexible interactions. This inherits from the topology of the graph-based search space. Also, the framework supports many different convolutional operators, increasing the diversity of feature representations. When comparing generated models between the coarse-grained and fine-grained predictions, the generated model for the fine-grained DRC hotspot detection is significantly more complex [6]. It indicates that the essential difference between these two routability prediction tasks is captured and reflected by the two discovered models. In summary, automated model development relieves developers from the long model development cycle and may become the trend of designing routability estimators in the future. However, notice that this method only automates the model development step, while the data generation and feature engineering still rely on human engineers. We believe there is still a long way toward fully automated routability estimator development.

2.3.4 DL Model Training and Inference

After the feature and label collection and preprocessing finishes and the model architecture is determined, the DL model training and testing is a relative simple task. In the training process, regularization like L2 norm is commonly adopted to avoid overfitting. The batch normalization should be applied. For optimizers, the common choices are stochastic gradient descent (SGD) and Adam [13]. Some researchers tend to believe parameter tuning with Adam optimizer is easier. The hyperparameters such as learning rate and regularization strength are adjusted according to the model and the training dataset.

The training and inference are usually accelerated by one GPU card. The training process usually finishes in one day, and the inference of one layout usually only takes seconds. But notice that the feature extraction overhead for each testing layout may take much longer time than the model inference time itself. Such feature extraction overhead may be reduced if the model is well integrated into a physical design tool. During training, for a large FCN model with a large input size, the memory size of GPU may limit the maximum batch size, which affects batch normalization quality and the model accuracy. This memory limit may require multi-GPU training or adopting the cropping of layouts as the input.

2.4 DL for Routability Deployment

Besides the prediction of design routability, another equally important topic is how to apply these developed estimators in real design flows to actually benefit design routability, which is designers' ultimate goal. However, this topic is less explored, and researchers haven't reached a consensus on a unified best solution to deploy routability estimators. In the following, we will introduce some representative research explorations on this topic.

2.4.1 Direct Feedback to Engineers

The most straightforward way to apply routability estimators is to directly provide feedback to human engineers. After finishing the placement of a design, engineers can apply the model to predict the routing congestion or the DRC violation that will emerge after the routing stage. Based on this prediction, engineers can select more routable layout solutions or proactively improve the layout solution to achieve better routability.

2.4.2 Macro Location Optimization

In the macro placement process, there is no effective cost metric to accurately evaluate the output layout quality at this early stage, since its final performance depends on the subsequent highly complex cell placement and routing stages. The oversimplified early estimations on the HPWL and area cannot precisely reflect the layout qualities after routing. Therefore, DL-based routability estimators are good fit to improve the quality of macro placement. The work of [11] integrates the DL model into the simulated annealing (SA)-based macro placement algorithm. After deriving a placement result from perturbation of SA, it utilizes the DL model to predict the number of design rule violations to see whether this solution will be accepted or not. With the integration of a DL model, it achieves solutions with fewer DRVs.

2.4.3 White Space-Driven Model-Guided Detailed Placement

Routability estimators can be applied to mitigate DRC violations proactively in the placement flow. This is achieved by spreading the white space of the regions with potential DRC hotspots [5]. After predicting the DRV hotspot locations, the mitigation method collects the white space in local regions around those estimated hotspots. Then, it redistributes white space among overlapped local windows and keeps incrementally moving cells to redistribute white space. After this re-legalization method, the output layout achieves fewer DRVs after routing.

2.4.4 Pin Accessibility-Driven Model-Guided Detailed Placement

Based on a pin accessibility-focused routability estimator, a model-guided detailed placement algorithm [30] can guide the detailed placer to avoid generating DRV-prone pin patterns. First, the estimator is used to generate a set of cell spacing rules, which applies to any designs using the same cell library. The rules define the minimum spacing between every pair of cells. These spacing rules are integrated into a detailed placer for optimization, minimizing the total amount of inserted placement blockages in a cell row considering cell orientations [30]. With the pre-inserted placement blockages, all pin patterns with predicted bad pin accessibility will be removed in the following legalization step. In this way, this model-guided detailed placement can effectively reduce pin accessibility-induced DRVs in the subsequent routing step.

2.4.5 Integration in Routing Flow

Besides improving the placement step, another perspective is to use routability estimators to facilitate global routing [7]. In global routing, based on congestion predictions, the routing cost of gcell congestion can be modified. For the gcells that are predicted congested, the router can shrink the available tracks so that the router will reduce the number of wires passing through these potential congestion areas. Also, the router can increase the wire/via cost of these congestion gcells when routing nets spanning large regions. Nets spanning large regions can find a way to detour and avoid potentially congested regions. With the help of a DL model, the router can reduce DRV count and routing congestions.

2.4.6 Explicit Routability Optimization During Global Placement

Some recent works [16] choose to explicitly optimize routability by directly integrating the routing congestion prediction into the global placement objective function as a new penalty term. In this way, the placer can utilize gradients to adjust all movable cells toward better routability during the optimization of such placement objective function. This may be the most direct and explicit way to optimize placement solutions with respect to a given routability estimator.

2.4.7 Visualization of Routing Utilization

In addition to most applications on optimizing layout solutions, some works [29] apply routability estimators in visualization. They visualize the routing utilization on-the-fly during FPGA placement and generate such real-time forecast results in GIFs or videos.

2.4.8 Optimization with Reinforcement Learning (RL)

Last, besides the deployment of DL-based routability estimators, the work of [22] indicates that the order of nets to be routed can significantly impact the routing quality. It proposes an RL-based algorithm to learn the ordering policy that minimizes the DRC violations from the net features.

2.5 Summary

In this chapter, we introduce deep learning-based methods for routability estimations. After introducing background on both routability and relevant deep learning algorithms, we emphasize the importance of global information and model receptive field, which motivates the adoption of DL models for routability predictions. After that, we introduce the methodology in detail, covering topics including data preparation and augmentation, feature engineering, model architecture design, and model training and inference. Finally, we introduce existing explorations in the application and deployment of these routability estimators in the physical design flow.

Although there have been many existing research efforts showing promising results in this direction, we believe DL-based routability estimations still face several challenges. First, the accuracy of DL estimators may degrade when applied to certain new designs, making them unreliable in practice. In addition, it is usually challenging to get access to adequate training data with sufficient diversity in VLSI design, which limits the development of high-quality and generalized DL estimators. In addition, the model size of recent DL estimators keeps increasing. While it may not be a problem as a stand-alone tool, this may prevent an efficient integration of the DL model into existing chip design flows. Considering these challenges, we believe DL-based routability estimators should target higher accuracy, better generalization, less inference cost, and improved interoperability in design flow in the future.

References

1. Alawieh, M.B., Li, W., Lin, Y., Singhal, L., Iyer, M.A., Pan, D.Z.: High-definition routing congestion prediction for large-scale FPGAs. In: Asia and South Pacific Design Automation Conference (ASP-DAC) (2020)
2. Araujo, A., Norris, W., Sim, J.: Computing receptive fields of convolutional neural networks. Distill (2019) doi:10.23915/distill.00021, https://distill.pub/2019/computing-receptive-fields
3. Bustany, I.S., Chinnery, D., Shinnerl, J.R., Yutsis, V.: ISPD 2015 benchmarks with fence regions and routing blockages for detailed-routing-driven placement. In: International Symposium on Physical Design (ISPD) (2015)
4. Chan, W.T.J., Du, Y., Kahng, A.B., Nath, S., Samadi, K.: BEOL stack-aware routability prediction from placement using data mining techniques. In: International Conference on Computer Design (ICCD) (2016)
5. Chan, W.T.J., Ho, P.H., Kahng, A.B., Saxena, P.: Routability optimization for industrial designs at sub-14nm process nodes using machine learning. In: International Symposium on Physical Design (ISPD) (2017)
6. Chang, C.C., Pan, J., Zhang, T., Xie, Z., Hu, J., Qi, W., Lin, C.W., Liang, R., Mitra, J., Fallon, E., Chen, Y.: Automatic routability predictor development using neural architecture search. In: International Conference on Computer-Aided Design (ICCAD) (2021)
7. Chen, J., Kuang, J., Zhao, G., Huang, D.J.H., Young, E.F.: PROS: A plug-in for routability optimization applied in the state-of-the-art commercial EDA tool using deep learning. In: International Conference On Computer Aided Design (ICCAD) (2020)
8. Chen, L.C., Papandreou, G., Schroff, F., Adam, H.: Rethinking atrous convolution for semantic image segmentation (2017). Preprint. arXiv:1706.05587
9. Ding, Y., Chu, C., Mak, W.K.: Pin accessibility-driven detailed placement refinement. In: International Symposium on Physical Design (ISPD) (2017)
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
11. Huang, Y.H., Xie, Z., Fang, G.Q., Yu, T.C., Ren, H., Fang, S.Y., Chen, Y., Hu, J.: Routability-driven macro placement with embedded cnn-based prediction model. In: Design, Automation & Test in Europe Conference & Exhibition (DATE) (2019)
12. Hung, W.T., Huang, J.Y., Chou, Y.C., Tsai, C.H., Chao, M.: Transforming global routing report into DRC violation map with convolutional neural network. In: International Symposium on Physical Design (ISPD) (2020)
13. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization (2014). Preprint. arXiv:1412.6980
14. Kirby, R., Godil, S., Roy, R., Catanzaro, B.: CongestionNet: routing congestion prediction using deep graph neural networks. In: International Conference on Very Large Scale Integration (VLSI-SoC) (2019)
15. Liang, R., Xiang, H., Pandey, D., Reddy, L., Ramji, S., Nam, G.J., Hu, J.: DRC hotspot prediction at sub-10nm process nodes using customized convolutional network. In: International Symposium on Physical Design (ISPD) (2020)
16. Liu, S., Sun, Q., Liao, P., Lin, Y., Yu, B.: Global placement with deep learning-enabled explicit routability optimization. In: Design, Automation & Test in Europe Conference & Exhibition (DATE) (2021)
17. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
18. Maarouff, D., Shamli, A., Martin, T., Grewal, G., Areibi, S.: A deep-learning framework for predicting congestion during FPGA placement. In: International Conference on Field-Programmable Logic and Applications (FPL) (2020)
19. Mirza, M., Osindero, S.: Conditional generative adversarial nets (2014). Preprint. arXiv:1411.1784

20. Pan, J., Chang, C.C., Xie, Z., Li, A., Tang, M., Zhang, T., Hu, J., Chen, Y.: Towards collaborative intelligence: routability estimation based on decentralized private data. In: Design Automation Conference (DAC) (2022)
21. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. In: International Conference on Machine Learning (ICML) (2018)
22. Qu, T., Lin, Y., Lu, Z., Su, Y., Wei, Y.: Asynchronous reinforcement learning framework for net order exploration in detailed routing. In: Design, Automation & Test in Europe Conference & Exhibition (DATE) (2021)
23. Ronneberger, O., Fischer, P., Brox, T.: U-Net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI) (2015)
24. Spindler, P., Johannes, F.M.: Fast and accurate routing demand estimation for efficient routability-driven placement. In: Design, Automation & Test in Europe Conference & Exhibition (DATE) (2007)
25. Tabrizi, A.F., Rakai, L., Darav, N.K., Bustany, I., Behjat, L., Xu, S., Kennings, A.: A machine learning framework to identify detailed routing short violations from a placed netlist. In: Design Automation Conference (DAC) (2018)
26. Tabrizi, A.F., Darav, N.K., Rakai, L., Bustany, I., Kennings, A., Behjat, L.: Eh? predictor: a deep learning framework to identify detailed routing short violations from a placed netlist. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. (TCAD) (2019)
27. Xie, Z., Huang, Y.H., Fang, G.Q., Ren, H., Fang, S.Y., Chen, Y., Hu, J.: RouteNet: routability prediction for mixed-size designs using convolutional neural network. In: International Conference on Computer-Aided Design (ICCAD) (2018)
28. Yu, F., Koltun, V.: Multi-scale context aggregation by dilated convolutions (2015). Preprint. arXiv:1511.07122
29. Yu, C., Zhang, Z.: Painting on placement: Forecasting routing congestion using conditional generative adversarial nets. In: Design Automation Conference (DAC) (2019)
30. Yu, T.C., Fang, S.Y., Chiu, H.S., Hu, K.S., Tai, P.H.Y., Shen, C.C.F., Sheng, H.: Pin accessibility prediction and optimization with deep learning-based pin pattern recognition. In: Design Automation Conference (DAC) (2019)
31. Zhou, Q., Wang, X., Qi, Z., Chen, Z., Zhou, Q., Cai, Y.: An accurate detailed routing routability prediction model in placement. In: Asia Symposium on Quality Electronic Design (ASQED) (2015)