Instance-level Timing Learning and Prediction at Placement using Res-UNet Network

He Liu[1,2*], Simin Tao[2], Zhipeng Huang[2], Biwei Xie[3,2], Xingquan Li[4,2,✉] and Ge Li[1]
[1]School of Electronic and Computer Engineering, Peking University, Shenzhen, China
[2]Peng Cheng Laboratory, Shenzhen, China
[3]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[4]School of Mathematics and Statistics, Minnan Normal University, Zhangzhou, China
Email: *liuh@stu.pku.edu.cn, ✉fzulxq@gmail.com

***Abstract*—Instance level post-routing timing analysis at the placement stage is of great importance for timing optimization such as gate sizing and cell movement etc. Determining the timing bottlenecks accurately and in a fast way has become significantly meaningful for accelerating the timing closure since the time-consuming iteration cycle. In this work, we propose an instance-level timing prediction framework to identify the critical cells of post-routing at the placement stage, which constructs a pixel level image-to-image timing hotspot map translation task using an encoder-decoder based Res-UNet. The network framework combines the strengths of residual learning and basic U-Net, helping in collecting the local and global features of the entire layout of the circuit over different spatial scales. Experimental results on ISCAS'89 benchmark circuits under the 28nm process node demonstrated that with the proposed model, the average prediction accuracy of the critical cells classification achieves 90.7% for unseen designs in terms of the value of the F1-score. Moreover, the framework has achieved a speedup of three orders of magnitude compared with the conventional design flow.**

***Index Terms*—timing prediction, instance level, Res-UNet, timing optimization**

I. Introduction

Static timing analysis (STA) plays a crucial role in the comprehensive chip design process [1]. At the placement stage, accurate timing analysis results are crucial for timing-driven placement to measure which placement satisfies the timing constraint. Similarly, it also guides designers in the direction of optimization [2]. Instance level optimization such as gate sizing and cell movement is performed to achieve timing closure. However, accurate timing estimation is difficult at placement without the exact routing topology. The expensive computational routing makes it impractical to execute it iteratively to evaluate optimization solutions.

All negative slack paths identified in static timing analysis are called "critical paths" and must be optimized. In recent academic research, there are primarily two categories of methods for determining critical paths. The first category involves traditional path-based approaches, the works [3], [4] introduced an efficient timing critical path generation algorithm on an STA graph, which can quickly search the critical paths under extensive path constraints. Secondly, with the advancement of machine learning, many research efforts [5], [6] have been addressed for estimating the net delay at the post-routing stage or predicting critical path slacks and the critical paths at the pre-routing stage. The work of [7] marks the first utilization of machine learning methods and compares various machine learning approaches for net-based timing prediction at the pre-routing stage. In [8], He et al. generated a look-ahead RC network and used the random forest-based machine learning model to predict the net delay and critical path slack at the placement stage. An end-to-end GNN model is proposed in [9] to predict the arrival time and slack at timing endpoints without invoking additional STA tools. A novel prediction framework in [10] can predict the post-routing critical path at the synthesis stage under multiple corners, utilized for guiding the optimization of critical paths.

Fig. 1 displays a simple circuit with only two timing paths. For large circuits, timing optimization during placement may employ techniques such as gate sizing and cell movement, making the evaluation of timing increasingly crucial. However, optimization based on path-level analysis can be computationally complex due to the generation of multiple paths and significant changes in cell positions. To enhance timing evaluation performance during placement, we emphasize the spatial information of cells within the layout to guide instance-level optimization, rather than solely relying on the path topology. We introduce a timing prediction framework that forecasts the timing and identifies the critical instances of post-routing, utilizing instance-level timing maps and timing-related features. By employing the spatial instance level timing feature maps within the mesh grid regions, Res-UNet effectively captures both local and global features throughout the entire layout. The contributions of this work include the following.

- We propose the concept and the distribution of instance slack according to the analysis of instance timing, which can be used to balance accuracy and runtime and guide incremental timing optimization during placement.
- We introduce a novel image learning-based model for predicting post-routing instance timing distribution with the instance slack map at placement, which is capable of identifying the critical instances.
- Our model combines ResNet and U-Net to effectively capture global information and local features, aiding in critical instance identification.
- Experimental results on open-source circuits show high

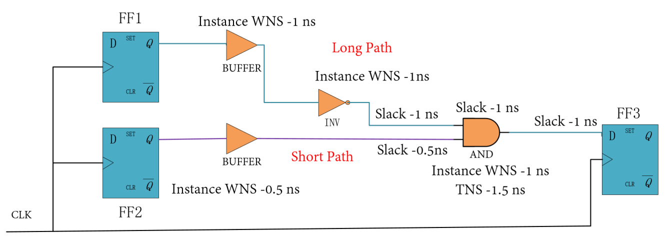


Fig. 1: A simple circuit with two timing paths.

accuracy in classifying critical instances and generalization to unseen circuits. Our approach significantly speeds up the traditional design flow, enabling machine learning-assisted timing optimization.

II. Instance Timing Analysis

A. Instance Level Timing

In the physical design stage, the relative layout of the instance significantly determines the timing performance of the circuit. It is essential to consider timing requirements across different paths to ensure that the entire design complies with timing constraints. A global perspective is crucial for identifying timing violations, analyzing timing bottlenecks, and conducting timing optimizations. Focusing on local analysis of paths with negative timing slack may lead to subsequent timing optimization efforts getting trapped in local suboptimal solutions. Therefore, we annotate the cells along the timing path with slack of the current path as follows. And the instance with negative timing slack is called "critical instance".

***Definition** 1 (**Instance Timing Slack**):* We use s^{it} to represent the timing slack of instance i, $s_i^{it} = f(\mathbf{s_i})$, $s^{it}: \mathbf{s} \in R^{k_i} \xrightarrow{f} s^{it} \in R$, where $\mathbf{s_i} = (s_1,\ s_2, \cdots,\ s_{k_i})$ is the path timing slack set pass through instance i.

In this paper, we give some possible expressions for map f as follows:

1) A possible case is that the timing slack of instance i is set as the worst path timing slack passes through instance i, then $s_i^{it} = \min\{s_1,\ s_2, \cdots,\ s_{k_i}\}$;
2) And if we want to count the total slack of all timing paths pass through instance i, then we can set the instance timing slack as $s_i^{it} = \sum_{j=1}^{k_i} s_j$;
3) On the basis of the second case, if we only need count the negative slacks, then we can set the instance timing slack as $s_i^{it} = \sum_{j=1}^{k_i} \min(s_j,\ 0)$.

As is depicted in Fig. 1, there are two timing paths through the "AND" gate, one is the "Long path" from the start point FF1:Q to endpoint FF3:D, which slack is noted as -1 *ns*, while the "Short path" from the start point FF2:Q to endpoint FF3:Q with the slack of -0.5 *ns*. The slacks of the instance "AND" will be annotated as -1 *ns*, -1.5 *ns*, -1.5 *ns*, corresponding to the calculations obtained using the three aforementioned formulas, respectively.

B. Instance Timing Distribution

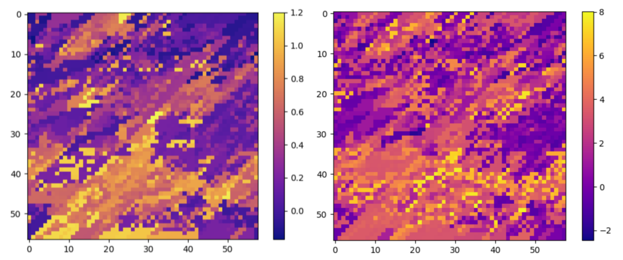
***Definition** 2 (**Instance Timing Distribution**):* To better characterize the timing situation of the layout, we define a continuous timing distribution function $d^{it}(x, y) = \sum_{i=1}^{N} \theta_i(x)\theta_i(y)$, and $\theta_i(x)$ is

$$\theta_i(x) = \begin{cases} s^{it} & |x - x_i| \le w_i, \\ 0 & \text{else}, \end{cases}$$

where N, x_i, and w_i denote the number of instances, the center coordinates and the width of instance i, respectively.

The above definition presents an analytical function for instance timing distribution. Practically, to calculate a numerical solution for instance timing distribution, this paper gives some possible numerical techniques as follows:

1) We first discretize layout as a grid map with $m \times n$ grids, and a possible instance timing distribution value in grid $G(i,\ j)$ is set to $d_{ij}^{it} = \min_{l \in L(i,\ j)}\{s_l^{it}\}$, where $L(i,\ j)$ is the set of the index number of instances located in grid $G(i,\ j)$.
2) On the basis of the first case, and another possible instance timing distribution value in grid $G(i,\ j)$ is set to $d_{ij}^{it} = \sum_{l \in L(i,\ j)} s_l^{it}$, where $L(i,\ j)$ is the set of the index number of instances located in grid $G(i,\ j)$.



(a) The worst negative slack. (b) The total negative slack.

Fig. 2: The comparison of the distribution of the worst negative slack and the total negative slack of the circuit `s15850` with the target density 0.78.

The comparison of the distribution between the worst negative slack and the total negative slack of the circuit `s15850` in ISCAS'89 benchmark is visually illustrated in Fig. 2. The X-axis and Y-axis indicate the width and height of the layout, respectively. It can be noted that the layouts with total negative slack distribution are prone to more severe timing violations, leading to a correspondingly more intricate optimization process.

III. Instance Timing Prediction Framework

In this section, we propose a Res-UNet-based timing prediction framework (Fig. 3) to identify critical instances that affect post-routing timing. Initially, static timing analysis is performed at the placement stage, and annotate the cells with slack values along the timing path. Then the layout is segmented into mesh grids, from which the transition time maps, instance density maps, and timing maps at placement are extracted as the input features of the proposed Res-UNet to predict the post-routing timing maps and identify the critical cells. The accuracy is evaluated by comparing the predicted timing maps with labeled maps from the post-routing static timing analysis using the F1-score metric.

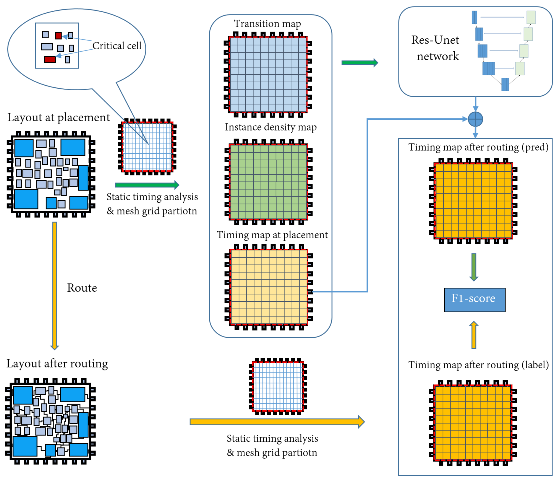


Fig. 3: Overall timing prediction framework.

A. Features Selection and Generation

For timing prediction and critical instance classification tasks, the input features and the chosen label are formulated as images for the prediction model are shown in Fig. 3.

Transition time map. The transition time significantly influences the drive strength of the instance in the timing path, whereby stronger driving capabilities lead to reduced delays as the transition time propagated. As discussed in the preceding Section II, when the critical instances of each grid have been determined based on different methods, the maps are constructed with the input transition time corresponding to the critical instance within each grid region.

Instance density map. Areas characterized by lower cell density typically present more resources available for routing, while an area with higher pin density must accommodate a greater number of routed wires to connect with the pins in that area. An instance density map is created using the calculated cell densities of each grid region, serving as an additional feature for the entire circuit layout.

Instance timing map. As the critical cell distribution is shown in Fig. 3, each grid may contain multiple critical cells, with each of these being traversed by multiple paths. In Section II, three distinct types of instance slacks are calculated using various methods tailored to different optimization tasks. These include: (1) the worst negative slack of the timing path traversing each instance; (2) the minimum total slack of the timing path through each instance; and (3) the smallest total negative slack of the timing path through each instance. The critical instance slack of each mesh grid region was extracted to construct the timing maps. The timing maps with pre-routing information at placement are used as input and the post-routing timing maps with exact detailed routing information as the label, the model then learns the residual between these timing maps along with other chosen features.

B. Res-UNet for Map Learning

The Res-UNet network integrates the concept of residual blocks into the encoder-decoder structure, similar to the U-Net architecture. The Res-UNet architecture was introduced to

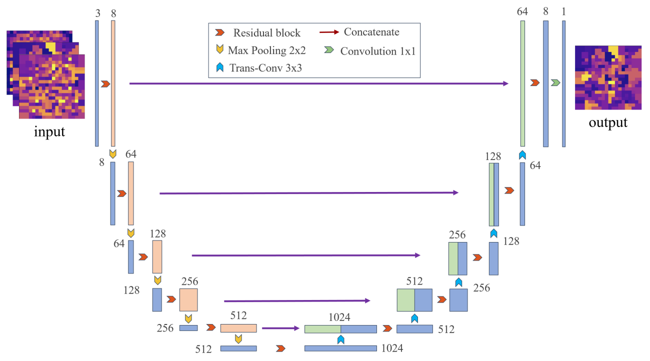


Fig. 4: Illustration of proposed Res-Unet architecture.

address issues related to the limited discriminative capability encountered in segmentation tasks involving small features [11]. It incorporates the ResNet [12] and U-Net [13] into a single structure, effectively addressing the problem of vanishing gradient problems seen in deep networks. Additionally, it also enhances feature learning and accuracy by integrating skip connections for better feature representation.

The Res-UNet architecture used in this work draws inspiration from the UNet, residual block and Inception module [14]. As is shown in Fig. 4, the structure comprises three primary components: encoding, bridge, and decoding sections. Which are built with multiple residual units, each comprising 3 x 3 convolution blocks and incorporating an identity map [15]. Each convolutional block is represented as a composite function of three sequential operations: a 3×3 convolution (Conv), batch normalization (BN), and leaky rectified linear unit (LeakyReLu). The inception module comprises multiple parallel branches, each consisting of a varying number of convolution blocks, which then concatenate their outputs to form a more complex feature map. This augmentation enhances the feature extraction capabilities of the conventional Res-UNet. Within the decoding path, preceding each residual unit, an up-sampling operation is performed to augment the feature maps from the encoding section and restore the low-resolution feature maps to their high-resolution counterparts. It is also indispensable to concatenate the feature maps from the corresponding encoding path. Following the final layer of the decoding path, a 1 x 1 convolution is applied to convert the multi-channel feature maps into the desired output.

In this study, the output of the network is also an image that represents the residual between the ground truth timing map after routing and the predicted timing map. The sum of the network output and the timing map generated from the placement stage is the predicted timing map after routing.

TABLE I: Benchmark statistics.

Ciucuit	Cell	Area (um^2)	Target density	Type
s5378	691	32×32	$\{0.64, 0.69, 0.73, 0.78, 0.83\}$	Train (seen)
s38417	5392	102×102	$\{0.58, 0.63, 0.68, 0.73, 0.78\}$	
s1488	355	20×20	$\{0.58, 0.63, 0.68, 0.73, 0.78\}$	
s38584	6450	98×98	$\{0.58, 0.63, 0.68, 0.73, 0.78\}$	
s1238	322	20×20	$\{0.58, 0.63, 0.68, 0.73, 0.78\}$	
s35932	5858	107×108	$\{0.58, 0.63, 0.68, 0.73, 0.78\}$	
s15850	1951	57×58	$\{0.58, 0.63, 0.68, 0.73, 0.78\}$	Test (unseen)
s13207	685	35×36	$\{0.58, 0.63, 0.68, 0.73, 0.78\}$	

TABLE II: The comparison of the critical instance classification accuracy in terms of precision, recall and F1-score metrics.

Circuit	T_density	CNN [16]			U-Net [13]			Res-UNet (Our work)			Runtime (s)		
		Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	C_tool (routing)	Inference time	Speed-up
s15850	0.58	65.1%	90.0%	75.6%	76.7%	85.8%	81.0%	88.5%	79.7%	**83.9%**	88	0.018	4888.9×
	0.63	78.1%	99.7%	87.6%	83.1%	99.3%	90.5%	87.0%	99.3%	**92.8%**	91	0.018	5055.6×
	0.68	74.3%	91.8%	82.1%	85.8%	90.4%	**88.0%**	87.7%	82.2%	84.9%	98	0.018	5444.4×
	0.73	68.1%	96.8%	79.9%	77.9%	95.6%	85.9%	81.9%	95.1%	**88.0%**	90	0.018	5000.0×
	0.78	72.6%	96.8%	83.0%	75.9%	94.4%	84.2%	86.8%	94.7%	**90.6%**	89	0.018	4944.4×
Avg.		71.64%	95.02%	81.6%	79.9%	93.1%	85.9%	86.38%	90.2%	**88.0%**	91.2	0.018	5066.7×
s13207	0.58	98.6%	78.0%	87.1%	89.9%	97.8%	**93.6%**	98.8%	86.8%	92.4%	64	0.018	3555.6×
	0.63	94.2%	87.1%	90.1%	85.9%	98.9%	91.9%	96.6%	92.5%	**94.5%**	65	0.017	3823.5×
	0.68	90.1%	91.9%	91%	81.1%	99.9%	89.6%	97.8%	88.9%	**93.1%**	65	0.017	3823.5×
	0.73	87.4%	89.1%	88.2%	79.2%	98.0%	87.6%	98.9%	89.1%	**93.8%**	73	0.017	4294.1×
	0.78	86.4%	95.6%	90.7%	79.8%	98.2%	88.1%	91.4%	93.8%	**92.6%**	67	0.017	3941.2×
Avg.		91.34%	88.3%	89.4%	83.2%	98.6%	90.2%	96.7%	90.2%	**93.3%**	66.8	0.017	3929.4×

IV. Experimental Results

To evaluate the efficiency of the proposed instance-level critical instance classification framework, we compared the results with another two models and the traditional design flow. Our models were implemented using PyTorch and Scikit-learn toolkit on a Linux machine with two NVIDIA A100 GPUs, 4 Intel Xeon Platinum 8380 CPUs at 2.3 GHz, and 1T RAM. In this work, the ISCAS'89 benchmark circuits are used and all designs were synthesized with 28nm industry process. Furthermore, the back-end design flow was performed using commercial physical design tools. The descriptive statistics of these benchmark circuits are summarized in Table. I. The upper six circuits are used for training along with the lower two are utilized for testing. We modified the positions of instances by changing the density parameter during placement and extracted features from the layouts with different instance positions. We compared the proposed Res-UNet based critical instance prediction model with a baseline CNN [16] model and a UNet [13] model. In Table II, column "T_density" is the target density of designs, and column "C_tool (routing)" lists the routing time of commercial Tool, and column "Inference time" lists the inference time of our trained model.

For each region of the predicted and ground truth timing maps, if the mesh grid contains critical cells, the region is labeled as 1, otherwise, mark it as 0. For the test of unseen circuits, the proposed Res-UNet-based model achieved better results in most test cases compared with the other two models. A larger F1-score indicates a higher balanced precision in the trained model's identification of critical instances. The F1-score of the proposed model ranges from 83.9%-92.8% for test circuit `s15850` and 87.6%-93.6% for circuit `s13207`, respectively. Specifically, for circuit `s15850`, the model showed a 6.4% and 2.1% improvement in average precision over the CNN and U-Net models, respectively. Regarding the unseen circuit `s13207`, the improvement was 3.9% over CNN and 3.1% compared to the U-net model. For clarity, the comparison between the predicted and actual (golden) timing map images for the circuit `s15850` is shown in Fig. 5. The images demonstrate that the predicted results exhibit a strong correlation with the ground truth timing maps.

In the physical design phase, obtaining an accurate timing map requires a time-consuming routing stage. We compared the routing time at various densities with the time required by our machine-learning model to directly infer the results. We were able to achieve significant speed improvements of up to 5444× for unseen test circuit `s15850` and up to 4294× for circuit `s13207`. This contributes to incremental timing optimization during placement, reducing iteration time and accelerating the speed of timing closure. And it's worth noting that the routing time can vary depending on placement target density and the number of standard cells. The inference time of our trained learning model is related to the size of the layout mesh grid that we partitioned. When the grid size remains constant, the inference time will also be consistent, allowing for a reasonable balance between accuracy and runtime.

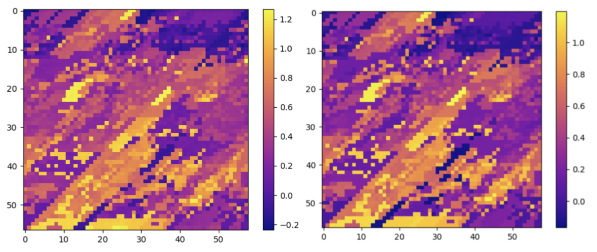

(a) Predicted timing map. (b) Ground truth.

Fig. 5: The predicted and golden timing map at post-routing of unseen circuit `s15850` with the target density of 0.63.

V. Conclusion

In this work, we proposed a post-routing critical instance prediction framework based on the Res-UNet network. By partitioning the whole chip layout into mesh grid regions to construct feature maps, we leverage the advantages of Res-UNet to extract both the global information of the entire layout and the local information of each mesh grid region. Experimental results on open-source designs demonstrated that the proposed framework achieved significant accuracy and a fast, stable prediction speed.

Acknowledgment

This work is supported in part by the Major Key Project of PCL (PCL2023A03).

References

[1] B. Jayaram and C. Rakesh, *Static Timing Analysis for Nanometer Designs: A Practical Approach*. Springer Science & Business Media, 2009.

[2] C. Guth, V. Livramento, R. Netto, R. Fonseca, J. L. Güntzel, and L. Santos, "Timing-driven placement based on dynamic net-weighting for efficient slack histogram compression," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, pp. 141–148, 2015.

[3] T.-W. Huang and M. D. F. Wong, "Ui-timer 1.0: An ultrafast path-based timing analysis algorithm for cppr," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 11, pp. 1862–1875, 2016.

[4] G. Guo, T.-W. Huang, Y. Lin, and M. Wong, "Gpu-accelerated critical path generation with path constraints," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, 2021.

[5] H.-H. Cheng, I. H.-R. Jiang, and O. Ou, "Fast and accurate wire timing estimation on tree and non-tree net structures," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2020.

[6] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, and L. Shi, "Fast and accurate wire timing estimation based on graph learning," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, IEEE, 2023.

[7] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2019.

[8] X. He, Z. Fu, Y. Wang, C. Liu, and Y. Guo, "Accurate timing prediction at placement stage with look-ahead rc network," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, p. 1213–1218, 2022.

[9] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin, "A timing engine inspired graph neural network model for pre-routing slack prediction," in *Proceedings of the 59th Annual Design Automation Conference 2022*, ACM, 2022.

[10] Q. Song, X. Cheng, and P. Cao, "Critical paths prediction under multiple corners based on bilstm network," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2023.

[11] X. Xiao, S. Lian, Z. Luo, and S. Li, "Weighted res-unet for high-quality retina vessel segmentation," in *2018 9th international conference on information technology in medicine and education (ITME)*, pp. 327–331, IEEE, 2018.

[12] A. Veit, M. J. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," *Advances in NIPS*, vol. 29, 2016.

[13] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pp. 234–241, Springer, 2015.

[14] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[16] J. P. A. Vieira and R. S. Moura, "An analysis of convolutional neural networks for sentence classification," in *2017 XLIII Latin American computer conference (CLEI)*, pp. 1–5, IEEE, 2017.