

A Model Stealing Attack Against Multi-Exit Networks

Pan Li, Peizhuo Lv*, Kai Chen[†], Shengzhi Zhang, Yuling Cai, Fan Xiang,

SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, China
 School of Cyber Security, University of Chinese Academy of Sciences, China
 Department of Computer Science, Metropolitan College, Boston University, USA

ABSTRACT

Compared to traditional neural networks with a single output channel, a multi-exit network has multiple exits that allow for early outputs from the model’s intermediate layers, thus significantly improving computational efficiency while maintaining similar main task accuracy. Existing model stealing attacks can only steal the model’s utility while failing to capture its output strategy, i.e., a set of thresholds used to determine from which exit to output. This leads to a significant decrease in computational efficiency for the extracted model, thereby losing the advantage of multi-exit networks. In this paper, we propose the first model stealing attack against multi-exit networks to extract both the model utility and the output strategy. We employ Kernel Density Estimation to analyze the target model’s output strategy and use performance loss and strategy loss to guide the training of the extracted model. Furthermore, we design a novel output strategy search algorithm to maximize the consistency between the victim model and the extracted model’s output behaviors. In experiments across multiple multi-exit networks and benchmark datasets, our method always achieves accuracy and efficiency closest to the victim models.

Index Terms— Model Stealing Attack, Multi-Exit Neural Networks, Defense

1. INTRODUCTION

With relentless pursuit of higher AI performance, the scale and complexity of models keep growing, greatly increasing computational cost [1, 2, 3]. This poses a significant challenge for model deployment in scenarios with real-time requirements or limited computing power, e.g., mobile or IoT devices [4, 5, 6]. Inspired by the fact that canonical samples can produce sufficiently confident results with fewer computation [7], multi-exit networks have emerged as a promising technique to enhance the computational efficiency of models.

Unlike traditional neural networks with a fixed execution path, multi-exit networks [8, 9] offer multiple classifiers as potential exits, as shown in Fig.1. This architecture allows

multi-exit networks to output in advance based on output strategies, thereby avoiding unnecessary computational costs. With a good output strategy, a multi-exit network can allocate fewer computational resources for “simple” samples and more for “complex” samples, thereby greatly improving the computational efficiency of the model while maintaining similar recognition accuracy. The output strategy for a multi-exit network is typically a set of predefined thresholds. When the model’s confidence in the current exit exceeds the corresponding threshold, it indicates that the prediction is sufficiently confident and can be output early.

Due to the advantages of fast inference [10, 11, 12] and energy-efficient computation [13, 14, 15], stealing multi-exit networks is very appealing to attackers. However, we find that the unique structure and output strategies of multi-exit networks render existing model stealing methods ineffective since they only focus on stealing the model’s classification functionality but ignore the output strategy. This leads to a significant downgrade in the computational efficiency of the extracted model, thus losing the advantage of multi-exit networks.

In this paper, we propose a novel model stealing attack against multi-exit networks that steals not only the model’s classification functionality but also the output strategy, making the extracted model approximate the output behavior of the victim model as closely as possible. We first employ Kernel Density Estimation (KDE) [16, 17] to estimate the time threshold of all the exits of the victim model and predict exit numbers for each query sample. Then, we introduce performance loss and strategy loss to train the extracted model, enabling it to learn the corresponding function of the victim model. Finally, we propose a new output strategy search algorithm to determine the optimal output strategy that maximizes the consistency between the extracted model and the victim model’s output behaviors.

2. METHOD

2.1. Threat Model

In this paper, we assume the attacker has no knowledge of the victim model’s internal structure and can only access it in a black-box manner. The attacker can collect a task-irrelevant

*Corresponding Authors.

[†]The IIE authors are supported in part by NSFC (92270204, U24A20236), CAS Project for Young Scientists in Basic Research (Grant No. YSBR-118).

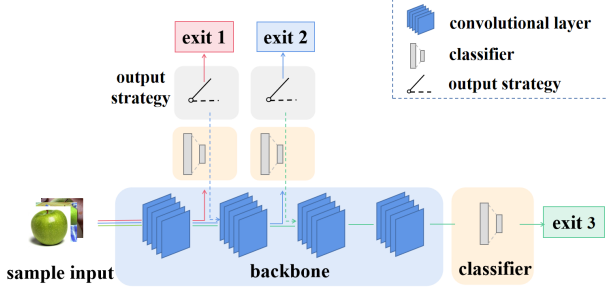


Fig. 1: Multi-Exit Model Structure.

dataset to query the victim model and fine-tune a pre-trained model to obtain the extracted model. It is worth mentioning that the structure of the extracted model and the victim model do not need to be the same. For example, the victim model's backbone can be based on VGG, while the extracted model can be built on ResNet.

2.2. Estimation Stage

To steal the classification functionality and output strategy of the victim model, we first employ a query dataset to probe the victim model. The output probabilities and exit numbers obtained from the victim model are then used as supervision to train our substitute model. While the output probability of a query sample can be directly obtained, the corresponding exit number cannot be observed from the output results. Therefore, we propose to infer the exit number based on the sample's execution time on the victim model. Since multi-exit networks always output at the first exit that satisfies the output strategy, we can conclude that the runtime of the multi-exit networks is positively correlated with the depth of the exit. Given the significant variations in runtimes across different exits, we propose using the Kernel Density Estimation (KDE) algorithm to analyze the runtime series of all samples. The local minima of the density curve can be interpreted as the temporal boundaries separating different exits.

The KDE algorithm proceeds as follows: First, the entire query dataset is used to query the victim model, obtaining n runtime measurements. These runtimes are then sorted, and the probability density at each runtime point is calculated using the Equation 1. Subsequently, we fit n probability density values to obtain a probability density curve. The m local minima of this curve serve as the runtime thresholds for different exits. By comparing the runtime of each sample to these thresholds, we can determine the corresponding exit number.

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right) \quad (1)$$

Here, $\hat{f}(x)$ denotes the estimated probability density function at point x , n represents the size of query dataset, K is the Gaussian kernel function, and h is the bandwidth.

2.3. Training Stage

As mentioned earlier, our attack not only aims to steal the victim model's classification functionality but also its output strategy. Therefore, our loss function consists of two parts as well. The performance loss guides the extracted model to learn the victim model's classification functionality. The strategy loss helps the extracted model to mimic the output strategy of the victim model, ensuring that query samples are mapped to same exits.

Performance Loss. The performance loss is used to constrain the classification probabilities of the extracted model to approximate that of the victim model. We achieve this by minimizing the KL-divergence distance across the entire query dataset, as shown in Equation 2.

$$L_P = \min_{\theta_S} \mathbb{E}_{x \sim D_{query}} \sum_{i=1}^K D_{KL}[S_i(x; \theta_S), V(x; \theta_V)] \quad (2)$$

where i represents the exit number, and there are a total of K exits in the multi-exit network. $S_i(x; \theta_S)$ denotes the classification probability from the i -th exit of the extracted model. $V(x; \theta_V)$ denotes the classification probability of the victim model's final output. D_{query} represents the entire dataset used for querying. θ_S and θ_V represent the parameters of the extracted model and the victim model, respectively. D_{KL} refers to the KL-divergence distance.

Strategy Loss. Given a query sample that is output at the i -th exit of the victim model, we aim to ensure that the same sample is also output at the i -th exit of the substitute model. Considering that multi-exit networks output at the first exit where confidence exceeds the threshold, we must ensure that the confidence of the given sample is greater than the threshold at the i -th exit, while being less than the threshold at all $i - 1$ previous exits. This guarantees that the sample will be output by the i -th exit of the substitute model and will not be prematurely output at any earlier exit. Our strategy loss is defined by Equation 3.

$$L_S = \min_{\theta_S} \sum_{i=1}^K \left[\max_{x \sim D_i} (0, \varphi_1 - \max(S_i(x; \theta_S))) + \sum_{j=1}^{i-1} \max_{x \sim D_i} (0, \max(S_j(x; \theta_S)) - \varphi_2) \right] \quad (3)$$

$S_i(x; \theta_S)$ denotes the classification probability from the i -th exit of the extracted model. $\max(S_i(x; \theta_S))$ denotes the maximum value in the classification probability, indicating the model's confidence in the current prediction. D_i represents the query samples output from the i -th exit of the victim model, and our goal is to train our extracted model to output D_i at the i -th exit as well. The first term in the equation aims to maximize the confidence of D_i at the i -th exit, ensuring that they meet the requirements of the output strategy and are correctly

output from the i -th exit. The second term in the equation aims to prevent the samples in D_i from being prematurely output from any earlier exit. Otherwise, these samples would fail to reach the exits where they should be output. φ_1 and φ_2 are two pre-defined constants¹.

2.4. Output Strategy Search.

In practice, simultaneously optimizing performance loss and strategy loss during training can lead to mutual influences. As a result, the final outputs of the substitute model cannot perfectly align with the victim model on all the query samples. Therefore, we aim to search for an optimal set of thresholds as the output strategy, thereby maximizing the output consistency between the extracted model and the victim model.

Specifically, we first use the substitute model to predict all query samples without early output. This means the substitute model outputs multiple predictions for each sample, with each prediction corresponding to one exit. Assuming there are n samples in the query dataset, then each exit has n corresponding confidences. We divide these n confidences for each exit into two sets, $Conf_i$ and $Conf_{\bar{i}}$. $Conf_i$ includes the confidence scores of samples that are expected to be output at the i -th exit, as they were output from the same exit in the victim model. $Conf_{\bar{i}}$ contains the confidences of samples that should not be output from the i -th exit. The final output strategy $T = \{T_1, T_2, \dots, T_K\}$ consists of a set of thresholds, one for each of the K exits. The candidate values for T_i are the n confidence values associated with the i -th exit. Consequently, there are up to n^K candidate output strategies. In practice, the number of candidate output strategies generated is significantly reduced. This is because T_i exclusively selects confidence scores within the interval $[\min(Conf_i), \max(Conf_{\bar{i}})]$ as candidate threshold. Any threshold outside this range would inevitably lead to a higher error rate.

$$T = \underset{T_{cand}}{argmax} \sum_{i=1}^K \mathbb{I}\{(Conf_i > T_i) \cup (Conf_{\bar{i}} < T_i)\} \quad (4)$$

Here, T_{cand} denote the set of all candidate output strategies. The indicator function \mathbb{I} is used to count the number of elements satisfying subsequent conditions. Equation 4 enables us to search for an output strategy that maximizes the output consistency between the substitute model and the victim model.

3. EVALUATION

Model Structure. We conduct experiments using the widely adopted multi-exit networks SDN [7]. Furthermore, since SDN

¹Through extensive experiments, we find that the values of φ_1 and φ_2 have a minimal impact on the final performance of the model, as long as φ_1 is greater than or equal to φ_2 . In our experiments, we set φ_1 to 0.95 and φ_2 to 0.9.

is implemented by placing multiple classifiers on the backbone network, we consider three different backbone network architectures for it: VGG [18], ResNet [19], and MobileNet [20]. In our experiments, we followed the structure and training process set forth in the original paper for the victim model. Moreover, since the attacker is unaware of the exact locations of the exits on the backbone network, they first estimate the number of exits using KDE and then evenly partition the backbone network of the extracted model to place the classifiers.

Datasets. We employ five mainstream datasets, including CIFAR10 [21], GTSRB [22], SVHN [23], FashionMNIST [24] and TinyImageNet. Among these, the first four are used to train victim models, while the last one is utilized as an unrelated dataset for querying purposes. As stated in the threat model, attackers do not have access to the victim model’s training or testing dataset and can only launch attacks using task-irrelevant datasets.

Baselines. In addition to directly querying with auxiliary datasets, model stealing can also be achieved using random noise and synthetic data. The construction of the query dataset has a significant impact on the effectiveness of model stealing. For a thorough comparison, we select the most representative works from each type of auxiliary dataset as baselines, including KnockOff (Knock)[25], Noise[26], and ES attack (ESA)[27].

Metrics. We utilized three metrics to evaluate the effectiveness of our method: Accuracy (ACC), Closeness (CLO), and Computation Cost (CC).

- **ACC** represents the model’s accuracy on the test dataset and is primarily used to measure the model’s classification performance.
- **CC** quantifies the computational cost of the model across the entire test dataset and is expressed in 10^9 FLOPs (floating-point operations), serving as an indicator of computational efficiency.
- **CLO** is a metric used to assess the output consistency between the substitute model and the victim model. Specifically, CLO measures the proportion of samples in the query dataset that produce identical outputs in both models.

3.1. Performance Evaluation

As shown in Table 1, we first evaluated the effectiveness of baselines and our approach on four datasets. Through testing on the victim model’s test dataset, we found that the extracted model trained by our method always achieves the closest accuracy to the victim model. This implies that our method can effectively learn the classification functionality of the victim model. In terms of computational efficiency, our method incurs a comparable computational cost to the victim model, significantly lower than all baseline methods. For instance, with the VGG on CIFAR10, our method incurs only a 4% additional computational overhead compared to the victim model, while the three baseline methods introduce computational overheads

Table 1: Model Performance.

Models		CIFAR10			GTSRB			SVHN			FMNIST		
		ACC	CLO	CC	ACC	CLO	CC	ACC	CLO	CC	ACC	CLO	CC
VGG	Victim	91.75%	—	3133	97.13%	—	2823	95.68%	—	8022	94.40%	—	2771
	Knock	88.25%	31.09%	4432(1.41X)	95.78%	24.40%	4868(1.72X)	91.99%	23.01%	11787(1.47X)	91.48%	29.75%	3760(1.36X)
	Noise	33.74%	21.34%	4887(1.56X)	45.21%	16.78%	4008(1.42X)	50.03%	38.12%	13156(1.64X)	38.56%	30.45%	3657(1.32X)
	ESA	82.43%	27.45%	4542(1.45X)	86.17%	18.76%	4742(1.68X)	87.29%	32.89%	10990(1.37X)	90.92%	20.34%	3574(1.29X)
	Ours	88.33%	63.27%	3258(1.04X)	95.42%	71.76%	3257(1.15X)	91.41%	64.85%	7848(0.98X)	91.69%	57.49%	3214(1.16X)
ResNet	Victim	89.39%	—	1519	96.66%	—	1141	95.11%	—	3633	93.34%	—	1158
	Knock	86.22%	34.69%	1937(1.28X)	92.19%	18.18%	2050(1.80X)	92.34%	32.77%	4871(1.34X)	90.81%	18.87%	1650(1.42X)
	Noise	31.89%	12.89%	2263(1.49X)	53.12%	25.67%	1848(1.62X)	42.45%	35.21%	5340(1.47X)	34.28%	18.56%	1598(1.38X)
	ESA	83.74%	38.12%	2369(1.56X)	85.65%	15.67%	1837(1.61X)	89.11%	23.98%	5122(1.41X)	88.02%	30.21%	1655(1.43X)
	Ours	86.88%	58.90%	1558(1.03X)	92.00%	64.30%	1374(1.20X)	92.53%	56.39%	3826(1.05X)	90.26%	51.65%	1330(1.15X)
Mobile	Victim	88.75%	—	3737	96.77%	—	3203	93.72%	—	9178	93.08%	—	3346
	Knock	83.35%	41.92%	5199(1.39X)	91.35%	30.77%	4788(1.59X)	90.52%	41.86%	12952(1.41X)	89.43%	27.19%	4339(1.29X)
	Noise	49.76%	31.09%	5717(1.53X)	52.01%	14.23%	4676(1.46X)	36.45%	29.76%	12390(1.35X)	30.87%	33.89%	4918(1.47X)
	ESA	82.79%	17.54%	5381(1.44X)	80.34%	25.87%	4932(1.54X)	86.44%	19.23%	12482(1.36X)	84.58%	28.65%	5019(1.50X)
	Ours	83.39%	59.72%	3914(1.05X)	91.87%	69.42%	4095(1.28X)	89.37%	62.44%	10212(1.11X)	89.37%	59.60%	3580(1.07X)

Table 2: The Prediction Accuracy of Exit Numbers

Extracted	CIFAR10	GTSRB	SVHN	FMNIST
VGG	0.998±1e-6	0.999±2e-5	0.999±1e-4	0.998±3e-6
ResNet	1.0±0.0	0.998±3e-5	1.0±0.0	0.999±1e-6
Mobile	0.998±2e-5	1.0±0.0	0.999±6e-7	0.998±1e-5

that are 10 times, 14 times, and 11 times greater than ours. Moreover, our method’s CLO also significantly exceeds all baseline methods, indicating a stronger consistency in output behavior between our extracted model and the victim model. These experimental results fully demonstrate that our method can successfully learn the output strategies of the victim model.

3.2. Accuracy on Exit Prediction

As shown in Table 2, our approach achieves nearly 100% accuracy in predicting query sample’s exits across various datasets and network architectures. This indicates that the runtime of a sample is indeed positively correlated with the depth of the corresponding exit, and KDE can effectively capture the temporal differences between different exits.

3.3. Sensitivity Analysis of Architectures and parameters

Numbers of Exits. The difficulty of model stealing may vary depending on the number of exits in the victim model. Here, we conducted experiments on four datasets, CIFAR10, GTSRB, SVHN, and FashionMNIST, and observed the attack effectiveness of our method when the victim models have 3, 4, 5, and 6 exits. Our experiments reveal that the number of exits has a very small impact on the attack effectiveness of our method, and the extracted models with different numbers of exits always exhibit very similar accuracy and closeness. Additionally, as the number of exits increases, we observe a slight decrease in the total computational cost. This is mainly because more samples output at the shallow exits, with only a few complex samples reaching the deep exits.

Table 3: Extracted Models with Different Backbone Architectures

Extracted	Victim	ACC	CLO	CC
VGG	VGG	88.33%	63.27%	3258(1.04X)
	ResNet	86.47%	58.86%	3383(1.08X)
	MobileNet	86.54%	61.70%	3289(1.05X)
ResNet	VGG	87.70%	53.59%	1610(1.06X)
	ResNet	88.68%	64.24%	1558(1.03X)
	MobileNet	86.76%	55.52%	1595(1.05X)
Mobile	VGG	82.68%	50.04%	3961(1.06X)
	ResNet	82.49%	57.81%	4036(1.08X)
	MobileNet	83.39%	59.72%	3923(1.05X)

Backbone Architectures. Because attackers are unaware of the victim model’s network architecture in black-box scenarios, it is necessary to evaluate the impact of using different structures in extracted models on attack effectiveness. We conducted experiments using a 4-exit model structure on the CIFAR10 dataset. As shown in Table 3, When using the same backbone structure as the victim model, the extracted model often achieves the highest accuracy and closeness. When using a different structure, the accuracy and computational cost of the extracted model averagely decrease by 1.36% and 2.16% respectively, still achieving good performance. This indicates that the choice of the extracted model’s structure has little impact on our method.

4. CONCLUSION

In this paper, we propose a novel model stealing attack targeting multi-exit networks, which extracts both the model functionality and output strategy. We employ KDE to analyze the target model’s output strategy and use performance loss and strategy loss to guide the training of the extracted model. Furthermore, we designed a novel output strategy search algorithm that can find the optimal output strategy to maximize the consistency between the victim model and the extracted model’s outputs. Various experimental results thoroughly demonstrate the effectiveness of our method.

5. REFERENCES

- [1] Raghuraman Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper,” *ArXiv*, vol. abs/1806.08342, 2018.
- [2] Antonio Polino, Razvan Pascanu, and Dan Alistarh, “Model compression via distillation and quantization,” *arXiv preprint arXiv:1802.05668*, 2018.
- [3] Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard, “Adaptive quantization for deep neural network,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, vol. 32.
- [4] Shunpu Tang, Lunyuan Chen, Ke He, Junjuan Xia, Lisheng Fan, and Arumugam Nallanathan, “Computational intelligence and deep learning for next-generation edge-enabled industrial iot,” *ArXiv*, vol. abs/2110.14937, 2021.
- [5] Xuan-Ha Nguyen, Xuan-Duong Nguyen, Hoang-Hai Huynh, and Kim-Hung Le, “Realguard: A lightweight network intrusion detection system for iot gateways,” *Sensors (Basel, Switzerland)*, vol. 22, 2022.
- [6] Gawsalyan Sivapalan, Koushik Kumar Nundy, Soumyabrata Dev, Barry Cardiff, and Deepu John, “Annet: A lightweight neural network for ecg anomaly detection in iot edge sensors,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 16, pp. 24–35, 2022.
- [7] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras, “Shallow-deep networks: Understanding and mitigating network overthinking,” in *International Conference on Machine Learning*, 2018.
- [8] Mary Phuong and Christoph H Lampert, “Distillation-based training for multi-exit architectures,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1355–1364.
- [9] Alexandros Kouris, Stylianos I Venieris, Stefanos Laskaridis, and Nicholas Lane, “Multi-exit semantic segmentation networks,” in *European Conference on Computer Vision*. Springer, 2022, pp. 330–349.
- [10] Chuang Hu, Wei Bao, Dan Wang, and Fengming Liu, “Dynamic adaptive dnn surgery for inference acceleration on the edge,” in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1423–1431.
- [11] Weiwen Jiang, Edwin H-M Sha, Xinyi Zhang, Lei Yang, Qingfeng Zhuge, Yiyu Shi, and Jingtong Hu, “Achieving super-linear speedup across multi-fpga for real-time dnn inference,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–23, 2019.
- [12] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang, “Neurosurgeon: Collaborative intelligence between the cloud and mobile edge,” *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.
- [13] Tian Dong, Han Qiu, Tianwei Zhang, Jiwei Li, Hewu Li, and Jialiang Lu, “Fingerprinting multi-exit deep neural network models via inference time,” *arXiv preprint arXiv:2110.03175*, 2021.
- [14] Ting-Kuei Hu, Tianlong Chen, Haotao Wang, and Zhangyang Wang, “Triple wins: Boosting accuracy, robustness and efficiency together by enabling input-adaptive inference,” *arXiv preprint arXiv:2002.10025*, 2020.
- [15] Alexandros Kouris, Stylianos I Venieris, Stefanos Laskaridis, and Nicholas Lane, “Multi-exit semantic segmentation networks,” in *European Conference on Computer Vision*. Springer, 2022, pp. 330–349.
- [16] Yen-Chi Chen, “A tutorial on kernel density estimation and recent advances,” *Biostatistics & Epidemiology*, vol. 1, no. 1, pp. 161–187, 2017.
- [17] Stanisław Węglarczyk, “Kernel density estimation and its application,” in *ITM web of conferences*. EDP Sciences, 2018, vol. 23, p. 00037.
- [18] Karen Simonyan and Andrew Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2014.
- [19] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun, “Deep residual learning for image recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.
- [20] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *ArXiv*, vol. abs/1704.04861, 2017.
- [21] Hongmin Li, Hanchao Liu, Xiangyang Ji, Guoqi Li, and Luping Shi, “Cifar10-dvs: an event-stream dataset for object classification,” *Frontiers in neuroscience*, vol. 11, pp. 309, 2017.
- [22] Shehan P Rajendran, Linu Shine, R Pradeep, and Sajith Vijayaraghavan, “Real-time traffic sign recognition using yolov3 based detector,” in *2019 10th international conference on computing, communication and networking technologies (ICCCNT)*. IEEE, 2019, pp. 1–7.
- [23] Pierre Sermanet, Soumith Chintala, and Yann LeCun, “Convolutional neural networks applied to house numbers digit classification,” in *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*. IEEE, 2012, pp. 3288–3291.
- [24] Mohammed Kayed, Ahmed Anter, and Hadeer Mohamed, “Classification of garments from fashion mnist dataset using cnn lenet-5 architecture,” in *2020 international conference on innovative trends in communication and computer engineering (ITCE)*. IEEE, 2020, pp. 238–243.
- [25] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz, “Knockoff nets: Stealing functionality of black-box models,” *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4949–4958, 2018.
- [26] Nicholas Roberts, Vinay Uday Prabhu, and Matthew McAteer, “Model weight theft with just noise inputs: The curious case of the petulant attacker,” *arXiv preprint arXiv:1912.08987*, 2019.
- [27] Xiaoyong Yuan, Leah Ding, Lan Zhang, Xiaolin Li, and Dapeng Oliver Wu, “Es attack: Model stealing against deep neural networks without data hurdles,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 6, no. 5, pp. 1258–1270, 2022.