Appendix

We provide details of the algorithm for searching the optimal stealing rate in Appendix A and extended experimental evaluations in Appendix B.

A. Algorithm Descriptions

For the stealing rate, we discuss how it scales stolen features from the branch and evaluate the knockoff quality for a given stealing rate in Section 3.4. Moreover, we assume $\alpha \in \mathbb{R}$, the high accuracy knockoff should achieve a stealing rate of 1.0 and the sub-optimal knockoff is between 0.0 to 1.0. However, determining the optimal stealing rate is expensive since we have to test all possible values on each test set. We further assume that the accuracy curve which varies by different stealing rates in the same evaluations is a smooth concave curve.

We provide a concise version of the algorithm for determining the best virtual knockoff in Algorithm 1, which involves the following steps: 1) Sampling phase: It samples numbers evenly for a given range as the stealing rate list L_α. 2) Evaluation phase: It evaluates each value from L_α and records the result in L_c. 3) Next iteration phase: It sets a new sampling range (s, e) for the next iteration. In prior experiments, we set fixed iterations to 3, which compute the number of decimals to 3 for all stealing rates.

Algorithm 1 Searching the best virtual knockoff

```
Require: Dataset $D$, Knockoff $K$, number of iterations $i$,
      number of samples $n$, start value $s$, end value $e$.
 1: Find optimal stealing rate $\alpha^*$ by evaluating $L_\alpha$ in the
    range of $s$ to $e$ on $D$.
 2: for each iteration 1, 2, ..., $i$ do
 3:     $L_\alpha$ = getSamples$(s, e, n)$
 4:     Create empty list $L_c$ for $L_\alpha$
 5:     for each stealing rate $\alpha$ in $L_\alpha$ do
 6:         $acc =$ evaluate$(D, K, \alpha)$
 7:         Append $acc$ to $L_c$
 8:     end for
 9:     $j =$ argmax$(L_c)$
10:     $s = L_\alpha[j-1]$
11:     $e = L_\alpha[j+1]$
12: end for
13: return $L_\alpha[j]$
```

B. Additional Experiments

We present an experiment for the possible limitations of ImageNet pretrained models, training the model from scratch for the victim model. We use publicly available ResNet-34 models from the timm library for CIFAR10 and CIFAR100 while training the models for Caltech256 and Indoor67 by ourselves. Other experimental settings remain consistent as prior.

Method	Caltech256	CIFAR10	CIFAR100	Indoor67
F_V	69.56% (1×)	95.40% (1×)	77.06% (1×)	65.07% (1×)
KnockoffNets [15]	67.76% (0.974×)	82.20% (0.862×)	57.38% (0.745×)	59.92% (0.921×)
Transformer-B [2]	68.62% (0.986×)	84.62% (0.887×)	63.07% (0.818×)	62.01% (0.953×)
Ours/base	67.81% (0.975×) (*72.68%, 0.579)	87.59% (0.918×) (*89.90%, 0.679)	59.47% (0.772×) (*62.18%, 0.679)	58.88% (0.905×) (*62.68%, 0.517)
Ours/alpha	73.42% (1.055×) (*73.45%, 1.052)	90.64% (0.950×) (*90.75%, 1.073)	64.31% (0.835×) (*64.32%, 1.032)	63.58% (0.977×) (*63.58%, 1.0)

Table 4. Experimentation Results.

The accuracy of the victim model and the adversary model reported in Table 4. We find the victim influences the adversary model's performance. The victim model provided by timm uses randomly initialized weight but is well-tuned. It performs with higher accuracy than the pretrained version, increases 8.46% on CIFAR10 and 14.92% on CIFAR100. For the adversary model, our approach and baselines show an improvement of 3%-8% on CIFAR10 and 6%-10% on CIFAR100. For the Caltech256 and Indoor67, the victim model is slightly lower than the pretrained version, and the accuracy of each adversary model also slightly decreases. Notably, the recovery rates $(Acc(F_V)/Acc(F_A))$ are similar to those reported in Table 2. The alpha branch still performs the improved accuracies of 4.8%, 6.02%, 1.24%, and 1.57% compared to baselines.