Saliuitl: Ensemble Salience Guided Recovery of Adversarial Patches against CNNs

Supplementary Material

Saliuitl: further details

Pseudocode

In Section 3, we describe *Saliuitl* in detail. We now present the pseudocode for *Saliuitl* in Algorithm 1. Note that the algorithm involves the subroutines *GetAttributes*, *Aggregate*, *PreprocessMask*, *Inpaint*, *UpdateCondition*, *Update*, and *StopCondition*. *GetAttributes* refers to the attribute extraction process applied for each binary feature map during the detection stage, hence it takes as input a binary feature map and returns its n_f attributes. The *Aggregate* subroutine aggregates the ensemble feature set S into the ensemble feature vector $\mathbf{s} \in \mathbb{R}^{n_f \times |\mathcal{B}_D|}$. This process involves not only organizing the input such that each attribute across the ensemble represents one of the n_f channels of $\mathbf{s}$, but also normalizing each dimension of $\mathbf{s}$ before it is fed into *AD*. In particular, each $1 \times |\mathcal{B}_D|$ vector corresponding to an ensemble attribute is normalized between -1 and 1.

During the recovery stage, we use the current iteration threshold β_r to compute the binary feature map B_r. In the *PreprocessMask* subroutine we compute a binary pixel mask *Mask*$\in \{0,1\}^{W \times H}$ using B_r. This may involve, for example, potentially ignoring neurons in B_r that are isolated outliers, and ultimately determining which input space regions (pixels) correspond to the neurons of interest in B_r. Note that the receptive fields of the neurons in the feature map $\mathbf{m}_i$ (and therefore those in B_r) depend on the layers that are applied to the input $\mathbf{x}_i$ to produce $\mathbf{m}_i$, hence they depend on the shallow layers of the victim model h. The *Inpaint* subroutine takes as input an image $\mathbf{x}_i \in \mathbb{R}^{W \times H \times C}$ and a binary pixel mask *Mask*$\in \{0,1\}^{W \times H}$, where W, H, and C are width, height, and channels of the input image. The pixels corresponding to ones in the binary pixel mask are inpainted (e.g., using black pixels, biharmonic inpainting, etc.), and those corresponding to zeros retain their original values, the result is the inpainted image $\hat{\mathbf{x}}_i$.

The *UpdateCondition*, *Update*, and *StopCondition* subroutines depend on the task of the victim model h. In the *UpdateCondition* we compare $\hat{\mathbf{y}}_i$ to $h(\hat{\mathbf{x}}_i)$, to determine whether $\hat{\mathbf{y}}_i$ should be updated; recall this condition corresponds to detecting new objects in object detection (i.e., objects with an IoU below 0.5 for all detected objects in $\hat{\mathbf{y}}_i$) and to a label change in image classification. For image classification, *Update* merely sets $\hat{\mathbf{y}}_i$ to $h(\hat{\mathbf{x}}_i)$; in object detection *UpdateCondtion* and *Update* are performed implicitly and simultaneously on every iteration, by applying non-maximum suppression on objects labeled with the same class in $\hat{\mathbf{y}}_i \cup h(\hat{\mathbf{x}}_i)$, using an IoU threshold of 0.4 (see our code implementation for further details). Finally, in *StopCondition*, we determine if the inpainted region is too large and in the case of image classification we also determine whether the output of the victim model changed; we terminate *Saliuitl* if any of these conditions are met. Note that for the sake of readability we place *StopCondition* at the end of the loop in Algorithm 1 and Figure 2, but in practice we can check for the first stopping condition after the mask preprocessing step, before performing inpainting.

Algorithm 1 *Saliuitl*

Require: input $\mathbf{x}_i \in \mathbb{R}^{W \times H \times C}$, model h, attack detector AD, attack detection threshold $\alpha^* \in [0,1]$, sets of saliency thresholds $\mathcal{B}_D \in \mathbb{R}^{|\mathcal{B}_D|}$ and $\mathcal{B}_R \in \mathbb{R}^{|\mathcal{B}_R|}$

```
m_i ← h(x_i)                          ▷ m_i ∈ R^{m_x × m_y} is a feature map.
S := {}    ▷ Empty sequence to be filled (ensemble attributes)
for β_b ∈ B_D do
    B_b := m_i ≥ β_b ▷ Binary feature map B_b ∈ {0,1}^{m_x × m_y}.
    S ← S ∪ GetAttributes(B_b)
end for
s ← Aggregate(S) ▷ Attribute ensemble vector s ∈ R^{n_f × |B_D|}.
if AD(s) < α* then                          ▷ Attack detection score
    return h(x_i)
else                          ▷ Attack detected→ enter recovery stage
    ŷ_i ← h(x_i)
    for β_r ∈ B_R do
        B_r := m_i ≥ β_r                    ▷ B_r ∈ {0,1}^{m_x × m_y}.
        Mask← PreprocessMask(B_r)     ▷ Mask∈ {0,1}^{W × H}.
        x̂_i ← Inpaint(x_i, Mask)
        if UpdateCondition(ŷ_i, h(x̂_i)) then
            ŷ_i ← Update(ŷ_i, h(x̂_i))
        end if
        if StopCondition(B_r, ŷ_i, h(x̂_i)) then
            return ŷ_i
        end if
    end for
    return ŷ_i
end if
```

Architecture and training of AD

A full description of AD is presented in Table 3. When deploying *Saliuitl* in a given domain (dataset) we propose to train *AD* using a balanced dataset of attacked and clean images from said domain, akin to $\mathcal{X}_0$ and $\mathcal{X}_0'$ as described in Section 4.1. The generation of such data is straightforward when clean images from the target domain are available by using (or generating) adequate patch attacks de-

Table 3. Architecture of the attack detector AD.

Layer	Input Size	Output Size	Activation	Batch norm	Kernel size	Input channels	Output channels	Pooling
1D Conv	$(1, n_f, \lvert\mathcal{B}_D\rvert)$	$(1, 12, 12)$	ReLU	✓	2	4	12	Adaptive average
1D Conv	$(1, 12, 12)$	$(1, 12, 12)$	ReLU	✓	2	12	12	Adaptive average
Flatten	$(1, 12, 12)$	$(1, 144)$	-	-	-	-	-	-
Linear	$(1, 144)$	$(1, 576)$	ReLU	✗	-	-	-	-
Linear	$(1, 576)$	$(1, 576)$	ReLU	✗	-	-	-	-
Linear	$(1, 576)$	$(1, 1)$	Sigmoid	✗	-	-	-	-

pending on the victim model task; there are many publicly available patches from the literature [11, 24, 36, 39]. Since this is a binary classification task, we train *AD* using binary cross-entropy loss, labeling attacked images as 1 and clean images as 0. Recall that the input to *AD* are the ensemble attribute vectors corresponding to each image, thus the threshold set $\mathcal{B}_D$ for detection and the n_f attributes to be extracted per image must also be defined before training, and cannot be changed at inference time. We consider the optimization algorithm, number of epochs, batch size, train/validation split, and early stopping strategy to be implementation choices. We state our concrete choices for these hyper-parameters later in the Appendix when we describe in detail *Saliuitl*'s parameters for our evaluation.

Evaluation Setup Details

Mean Average Precision

A crucial metric in our evaluation of object detection scenarios is the mean average precision (mAP), which is the area under the precision-recall curve of detected objects. We provide further details on its computation. Objects detected by the model are grouped by class, and a descending confidence score order is imposed on each group. The precision-recall curve for each group (i.e., class) is computed following the order of the group, and it indicates how precision (accurate inferences) evolves at different levels of recall (detected ground truth objects), and the average precision (AP) is computed as the area under the curve; averaging the AP of all classes yields the mAP. In particular, we compute the mAP using the 11 point interpolation method of the Pascal VOC 2007 challenge [13]. Recall that we use the victim model's clean outputs as the ground truth; our victim model is YOLOv2 and we set its confidence and non-maximum suppression thresholds to 0.4 throughout our evaluation, including the computation of ground truth (i.e., clean) bounding boxes.

Patch Attack Models

In this section we describe the attack models employed throughout our evaluation. For object detection, we use evasion or hiding-attack patches, which are optimized to minimize the objectness score of the victim model. Patches are applied following the attack model by [36]. In short, adversarial patches are applied at the center of the bounding boxes of detected objects in the clean image. Each patch is always scaled to occupy 20% of the attacked object's total bounding box area; see the top row of Figure 3 for illustrations of our attacks on object detection. Note that with the exception of the multi-object patch attack, we limit the attack to a single object. To perform double patch attacks, we create two patches occupying 10% of the original bounding box area each, and place them reflected diagonally from each other w.r.t to the center of the attacked object's bounding box; as seen in Figure 3, the pattern of both patches in our double patch attack are identical. To perform triangular patch attacks, we apply a single patch attack, but use 40% of the attacked object's bounding box instead of 20%, and then we remove half of the patch diagonally. Multi-object patch attacks are essentially single-patch attacks that are applied to all objects detected by the victim model on the clean image. Note that in all our evaluations, except for our adaptive attack experiment, we do not perform patch optimization and instead use the publicly available patch from [24]. We choose to evaluate on this patch due to its availability and due to recent work showing that it is more challenging to detect than most other attacks in the literature [38]. For our adaptive attacker experiment in Section 4.3, we optimize single-patch attacks from scratch based on [36], using different stealthiness values. Moreover, note that during the training of *AD* we use only single-patch attacks with the publicly available *OBJ* patch from [36], so that the evaluation patch attack is not seen by *AD*.

For image classification, we use an untargeted classification attack, following the official implementation of [39]. Square patches with a fixed 32×32 pixel area are placed randomly on an image and the binary cross entropy of the victim model w.r.t to the ground truth label for the corresponding input is maximized. Our chosen adversarial attack model on image classification is displayed on the bottom row of Figure 3. As opposed to our attack on detection, this attack is input specific. To generate double patches, we merely halve the area of the randomly placed patch, and then place a new patch symmetrically reflected from the original patch location; for four patches, we repeat the process in two new locations (and of course, we halve the

Table 4. Patch attack effectiveness on all datasets.

Attack	Dataset			
	INRIA	VOC	ImageNet	CIFAR-10
Single	0.2292	0.2852	0.8959	0.5022
Double	0.2153	0.2601	0.9433	0.6880
Quadruple	-	-	0.9851	0.7918
Triangular	0.1979	0.2224	0.0890	0.0614
Multi-object	0.6215	0.5768	-	-

area again). For triangular patches, we follow the same procedure we use for object detection, we take the single patch attack, double its size, and remove half of it diagonally; note that unlike rectangular attacks with any number of patches, we do not perform optimization for triangular patches, and merely reshape the square patches.

We also report the attack effectiveness of each type of attack in Table 4. From the attacks on object detection (INRIA and Pascal VOC), we observe that modifying the patch attacks into double and triangular patches results in a drop in attack effectiveness. This follows from the fact that the patches are optimized according to the single and multi-object patch scenarios [24, 36], and hence changing the shape and location of the patches makes them less effective. Note that for image classification (ImageNet and CIFAR-10), where we optimize the patches from scratch, the attack effectiveness increases for multiple patches. This suggests that if an adversarial patch attack is optimized to attack more than one region, this may lead to a larger adversarial impact, without increasing the total area of the attack. The large drop in effectiveness for the triangular patches on image classification is congruent with the results for object detection, since triangular patches for classification are obtained by reshaping single patch attacks without performing any optimization; we conjecture the much larger drop in effectiveness for image classification compared to the drop for object detection follows from the input dependency of the attacks on image classification, which makes them more susceptible to deviations from the patch region they were optimized for.

Saliuitl (further) implementation details

We introduce *Saliuitl* in Section 3, and provide the detailed pseudocode in Algorithm 1. Here we describe our concrete implementation choices for the different components of *Saliuitl*, upon which we base our numerical results. The detailed pseudocode for our DBSCAN-based implementation of *Saliuitl* is presented in Algorithm 2.

Attribute Extraction via DBSCAN. Recall that we extract four attributes from each binarized feature map B_b in the ensemble defined by $\mathcal{B}_D$ during the detection stage, as in [3]. In Algorithm 2 the attributes extracted from each binary feature map B_b are placed into the set S_b, moreover, we also save the clusters found by DBSCAN for B_b in the set $\mathcal{C}_b$, as they might be used during the recovery stage as well.

Algorithm 3 shows how we compute the four attributes

Algorithm 2 *Saliuitl*-DBSCAN

Require: input $\mathbf{x}_i \in \mathbb{R}^{W\times H\times C}$, model h, attack detector AD, attack detection threshold $\alpha^* \in [0,1]$, sets of saliency thresholds $\mathcal{B}_D \in \mathbb{R}^{|\mathcal{B}_D|}$ and $\mathcal{B}_R \in \mathbb{R}^{|\mathcal{B}_R|}$

$\mathbf{m}_i \leftarrow h(\mathbf{x}_i)$ ▷ $\mathbf{m}_i \in \mathbb{R}^{m_x\times m_y}$ is a feature map.
$S := \{\}$ ▷ Empty sequence to be filled (ensemble attributes)
$\mathcal{C} := \{\}$ ▷ Empty sequence to be filled (clusters)
for $\beta_b \in \mathcal{B}_D$ **do**
  $B_b := \mathbf{m}_i \geq \beta_b$ ▷ Binary feature map $B_b \in \{0,1\}^{m_x\times m_y}$.
  $S_b, \mathcal{C}_b \leftarrow$ GetAttributesDBS(B_b)
  $S \leftarrow S \cup S_b$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}_b$
end for
$\mathbf{s} \leftarrow$ Aggregate(S) ▷ Attribute ensemble vector $\mathbf{s} \in \mathbb{R}^{4\times|\mathcal{B}_D|}$.
if $AD(\mathbf{s}) < \alpha^*$ **then** ▷ Attack detection score
  return $h(\mathbf{x}_i)$
else ▷ Attack detected→ enter recovery stage
  $\hat{\mathbf{y}}_i \leftarrow h(\mathbf{x}_i)$
  for $\beta_r \in \mathcal{B}_R$ **do**
    $B_r := \mathbf{m}_i \geq \beta_r$ ▷ $B_r \in \{0,1\}^{m_x\times m_y}$.
    Mask← PreprocessMaskDBS$(B_r, \beta_r, \mathcal{C}, \mathcal{B}_\mathcal{D})$
      ▷ *Mask*$\in \{0,1\}^{W\times H}$.
    $\hat{\mathbf{x}}_i \leftarrow$ Inpaint$(\mathbf{x}_i, Mask)$
    if UpdateCondition$(\hat{\mathbf{y}}_i, h(\hat{\mathbf{x}}_i))$ **then**
      $\hat{\mathbf{y}}_i \leftarrow$ Update$(\hat{\mathbf{y}}_i, h(\hat{\mathbf{x}}_i))$
    end if
    if StopCondition$(B_r, \hat{\mathbf{y}}_i, h(\hat{\mathbf{x}}_i))$ **then**
      return $\hat{\mathbf{y}}_i$
    end if
  end for
  return $\hat{\mathbf{y}}_i$
end if

Algorithm 3 GetAttributesDBS

Require: Bin. feature map $B \in \{0,1\}^{m_x\times m_y}$, threshold $\beta \in \mathbb{R}$
$n_{imp} := \sum_j \sum_k B(j,k)$ ▷ Number of important neurons
$\mathcal{C}_x \leftarrow$ DBSCAN(B) ▷ Important neuron clusters
$n_c \leftarrow |\mathcal{C}_x|$ ▷ Number of clusters.
$\mathcal{E} := \{\}$
  ▷ Empty sequence to be filled (set of intra-cluster distances)
for $c \in \mathcal{C}_x$ **do**
  $d_{ic} \leftarrow$ AvgICdistance(c)
    ▷ Average intra-cluster distance for c.
  $\mathcal{E} \leftarrow \mathcal{E} \cup d_{ic}$
end for
$\overline{d_{ic}} \leftarrow$ ArithmeticMean$(\mathcal{E})$ ▷ Mean over $\mathcal{E}$
$\sigma(d_{ic}) \leftarrow$ PopulationStdDev$(\mathcal{E})$ ▷ Std. dev. over $\mathcal{E}$
$S_x \leftarrow \{n_{imp}, n_c, \overline{d_{ic}}, \sigma(d_{ic})\}$
return $S_x, \mathcal{C}_x$

Algorithm 4 PreprocessMaskDBS

Require: Bin. feature map $B \in \{0,1\}^{m_x \times m_y}$, threshold $\beta \in \mathbb{R}$, set of cluster sets $\mathcal{C}$, set of det. thresholds $\mathcal{B}_D \in \mathbb{R}^{|\mathcal{B}_D|}$
Mask$\leftarrow \mathbf{0}$ ▷ Initialize *Mask*$\in \{0,1\}^{W \times H}$ to zero matrix
if $\beta \in \mathcal{B}_D$ **then** ▷ Was $\mathcal{C}_x$ computed in the det. stage?
$\mathcal{C}_x \leftarrow$ GetClusters$(\mathcal{C}, \beta)$
else
$\mathcal{C}_x \leftarrow$ DBSCAN(B)
end if
for $c \in \mathcal{C}_x$ **do**
Mask$\leftarrow$ UpdateMask$(c,$ *Mask*$)$
end for
return *Mask*

used in our evaluation. The first feature is the number of important neurons n_{imp}, i.e., the number of ones in B_b. The other three features characterize the spatial distribution of important neurons in B_b, and are obtained through density-based clustering using DBSCAN; the advantage of this clustering approach is that it allows arbitrarily shaped clusters [12]. We employ sci-kit learn's DBSCAN implementation [30]; the details of DBSCAN [12] are outside of our scope. We thus treat the binarized feature map as a two-dimensional $m_x \times m_y$ plane and use DBSCAN to identify the set of important neuron clusters $\mathcal{C}_b$. We then use as a second attribute the number of identified clusters $|\mathcal{C}_b|$. For the last two attributes, we compute for each cluster $c \in \mathcal{C}_b$ the average intra-cluster distance (distance between points in c) denoted d_{ic}, and we then compute its mean $\overline{d_{ic}}$ and its standard deviation $\sigma(d_{ic})$ over the set $\mathcal{C}_b$. Note that the average intra-cluster distance d_{ic} for a given cluster c involves computing the distance matrix for the elements in c, and then taking the average of its lower triangular elements. To put a limit on the computation time, if $|c| > 1000$, we take a random sample of 1000 elements and use only said sample to compute the distance matrix and d_{ic}. Once the four attributes are extracted for all members of the ensemble, we aggregate S into the ensemble attribute vector $\mathbf{s}$, normalizing across each of the four dimensions as described before. For further details refer to our code implementation.

Mask Preprocessing via DBSCAN. During the recovery stage, we use the threshold set $\mathcal{B}_R$ (that may overlap with the set $\mathcal{B}_D$) which is in strictly descending order. Recall that for each threshold $\beta_r \in \mathcal{B}_R$, we compute a binary feature map B_r from $\mathbf{m}_i$ to localize input regions that correspond to adversarial patches. In particular, for our DBSCAN-based implementation we find clusters of important neurons in B_r and only consider pixel regions corresponding to neurons that are within a cluster.

In Algorithm 4 we describe how *Mask* is obtained for a given B_r and its corresponding threshold β_r. After initializing *Mask* to zeros, we first check whether the clusters for B_r were already computed in the detection stage (i.e., we check if $\beta_r \in \mathcal{B}_D$), if they were, we need only to retrieve the clusters from the set of cluster sets $\mathcal{C}$ (already computed in the detection stage). Note that if the clusters are not available we need to compute them using DBSCAN. Then, important neurons within the clusters found by DBSCAN are used to update *Mask*; in short, we map each such neuron back to the pixels in input space that are within its receptive field. This operation depends on the victim model h and the particular shallow layer used to compute $\mathbf{m}_i$, our code implementation shows the details on how this is done for our choice of using the first max pooling layer in ResNet-50 and YOLOv2. We then feed the computed mask *Mask* and the original image $\mathbf{x}_i$ to the inpainting subroutine; in our implementation we use sci-kit image's biharmonic inpainting [37] (version 0.19.3), whose description is beyond our scope. The implementation of the update and stopping conditions, as well as the update of the recovered output $\hat{\mathbf{y}}_i$ follow our description for Algorithm 1.

Training of *AD*. We now turn to describe in detail how we trained AD to detect adversarial patches in the benchmark datasets used in our evaluation. In all four datasets, we select a random split, significantly smaller than the one we use for evaluation, to be used for training. For the object detection datasets (INRIA and Pascal VOC) we randomly select 20% of the training sets, resulting in a total of 123 images for INRIA, and 1003 for Pascal VOC, and for evaluation we use their entire test sets (288 and 4905 images, respectively). For CIFAR-10, we employ 2.5% of the validation set (250 images) as training data, and the rest of said set (9423 images) for evaluation. Finally, for ImageNet we use 2% of the validation set (2500 images) for training, and the remaining 98% (37065 images) for evaluation. Note that these splits used for image classification do not add to 100%, this is because we randomly extracted the training data before filtering out inputs where the victim model does not achieve a correct prediction in the clean setting. For each dataset, we generate an attacked version of its training split using the patch attack that corresponds to the dataset's task, and importantly, we do so using *exclusively* rectangular single-patch attacks, moreover note that for object detection we use the patch by [36], distinct from the patch used in our evaluation [24]. Afterwards, we label the original dataset and its attacked version as clean (0) and adversarial (1) examples, respectively, and proceed to train AD using binary cross-entropy loss; we employ the Adam optimizer with default parameters (betas=(0.9, 0.999) and learning rate $\gamma = 0.0001$). We use a batch size of one, and shuffle the dataset after each training epoch. Moreover, we select 20% of the training data as a validation split, to perform early stopping when the validation loss has no improvement after 200 epochs. AD is able to converge in only a few epochs (typically less than 10) in all four datasets. Regarding the specifics of weight initializations and simi-

lar hyper-parameters, we perform the entirety of our training using PyTorch 1.13.1 and we use the default parameters for all modules [29]. Recall the extracted attributes we use for our evaluations rely on DBSCAN clustering. To obtain the features fed into *AD* during training and inference time we employ sci-kit learn's DBSCAN implementation[30]; we use version 1.2.2 and use the DBSCAN parameters described in Section 4.1.

Parameters for baseline methods

In our evaluation, we varied the parameters of the baseline methods as follows. For *NutNet* [25], we vary the number of blocks between all three possible settings in the official implementation (8, 16, and 32), we vary the coarse threshold k_1 in the range $[0.04, 0.2]$ using a step size of 0.005, and we vary the fine threshold k_2 in the range $[0.1, 0.3]$ using a step size of 0.05. For *PAD* [22], we vary the threshold parameter (default value is 80 in [22]) in the range $[10, 90]$, using a step size of 10 (for ImageNet we use a step size of 20 in the range $[20, 80]$ due to the high computational cost of running *PAD* on such a large dataset and we also include results using a threshold parameter of 90, due to its superior performance in all datasets). For *Jedi* [35], we vary the threshold at the output of the autoencoder to an eighth, a quarter, and half of its original value. For *Patch Cleanser* [40] we vary the number of masks k, setting it to half, double, and triple of its default setting $k = 6$. Note that *Patch Cleanser* is not applicable to the object detection task. For *Themis* [18], we vary both neuron importance thresholds (by default fixed at $\beta = 0.75$, $\theta = 0.85$) in the range $[0.05, 0.95]$, using a step size of 0.05. Moreover, since *Themis* assumes a known patch size, we introduced *Themis-50*, which is a baseline representing the case where *Themis* expects patches that are half the size of the true bound on the patch size, that is, for *Themis* we provide the ground truth bound on patch size, which corresponds exactly to 32×32 pixels for image classification, and for object detection we use the maximum patch size across each evaluation dataset (INRIA and Pascal VOC), recall that for our attacks on object detection the patch size depends on the size of the attacked objects. For *FNS* [43], we vary the Gaussian decay rate λ in the range $[1, 3]$ using a step size of 0.4, and we vary the clipping parameter α in the range $[1, 2]$ using a step size of 0.05. Finally, for *Object Seeker* [41] we vary the pruning threshold τ (0.6 by default) and the masked confidence threshold γ_m (with no specific default value), both in the range $[0.1, 0.9]$ using a step size of 0.1. Note that *Object Seeker* is not applicable to the image classification task.

Complementary results

In the following, we present figures referenced in Section 4.2 regarding results we did not present in full detail due to spatial constraints.

Object Detection

Figures 5 and 6 show the detailed tradeoff between recovery rate and lost predictions corresponding to object detection on INRIA and Pascal VOC, respectively. For these figures, the four columns in each row represent the four object detection patch attack scenarios (single, double, triangular, and multi-object patches). In the top row we show the tradeoffs achieved by each method, where each point is a specific setting for its parameters. Note that not all settings are displayed in the figure; to ease visualization, we report only points corresponding to Pareto optimal settings (i.e., those for which not other setting has a higher recovery rate *and* a lower lost prediction rate) for *NutNet*, *Themis*, *Themis-50*, *FNS*, and *Object Seeker*. Moreover, we display results only for the relevant region that allows to visualize the best settings from all methods (e.g., if some settings have a lost prediction rate close to 1 they would not appear in our plots). In the middle row, we report the maximum recovery rate that each method achieves for a fixed limit on the lost prediction rate, while the bottom row shows the maximum recovery rate that each method achieves for a fixed limit on the inflicted attack rate (as defined in Section 4.1). When a method is not represented in the middle or bottom rows of these figures it means that it is unable to achieve a non-zero recovery rate for the largest limit on lost prediction rate or inflicted attack rate denoted in the corresponding figures.

Figures 7 and 8 display the tradeoff between clean and adversarial nmAP on INRIA and Pascal VOC, respectively. Once again, the columns represent the four object detection patch attack scenarios, each point in the figure represents the adversarial/clean nmAP tradeoff achieved by a parameter configuration of a recovery method, and in the case of *NutNet*, *Themis*, *Themis-50*, *FNS*, and *Object Seeker*, we report only Pareto optimal configurations.

Image Classification

We present the tradeoff between recovery rate and lost predictions achieved by each method on ImageNet and CIFAR-10 in Figures 9 and 10, respectively. As in Figures 5 and 6, each column corresponds to a patch attack scenario (i.e., for image classification, single, double, quadruple, and triangular patch attacks). As before, each point in the top row of Figures 9 and 10 represents a parameter configuration of a recovery method, we focus on Pareto optimal settings for *NutNet*, *Themis*, *Themis-50*, and *FNS*, and we focus on a region that allows to visualize only relevant configurations from all methods. The middle and bottom rows of the figures present recovery rates at fixed limits on lost prediction and inflicted attack rates as described for the object detection datasets.

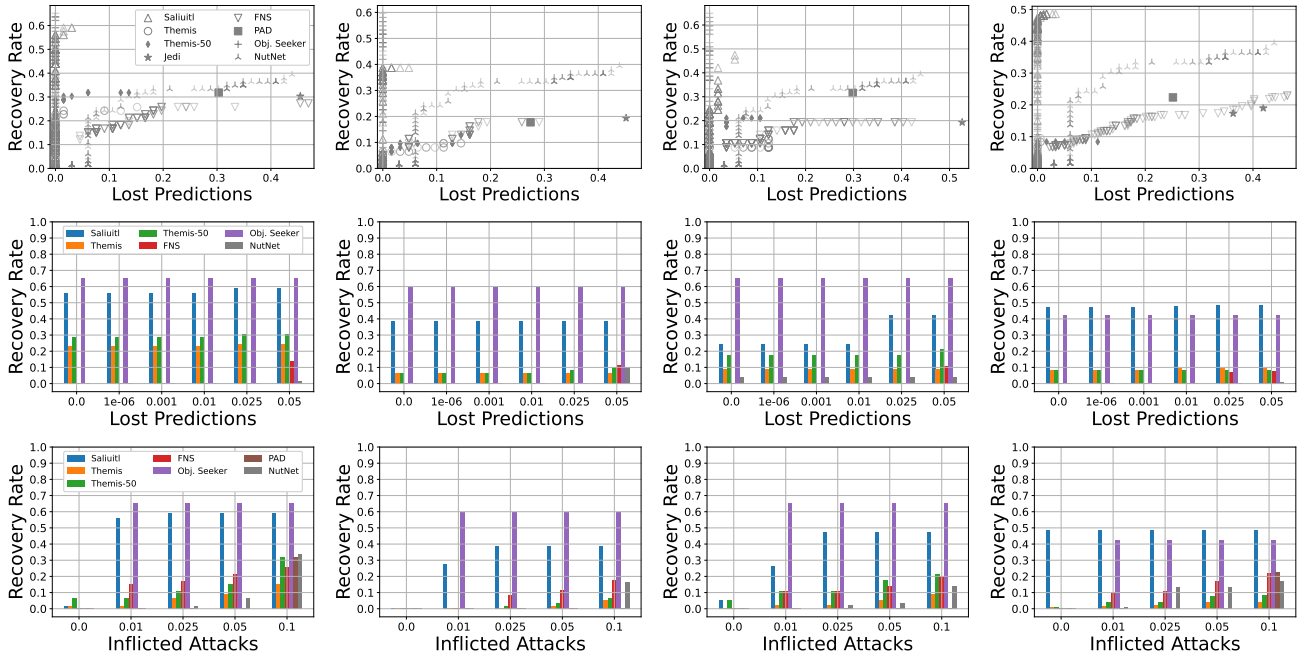


Figure 5. Recovery results for Object Detection (INRIA). From left to right: single, double, triangular, and multi-object patch attacks.

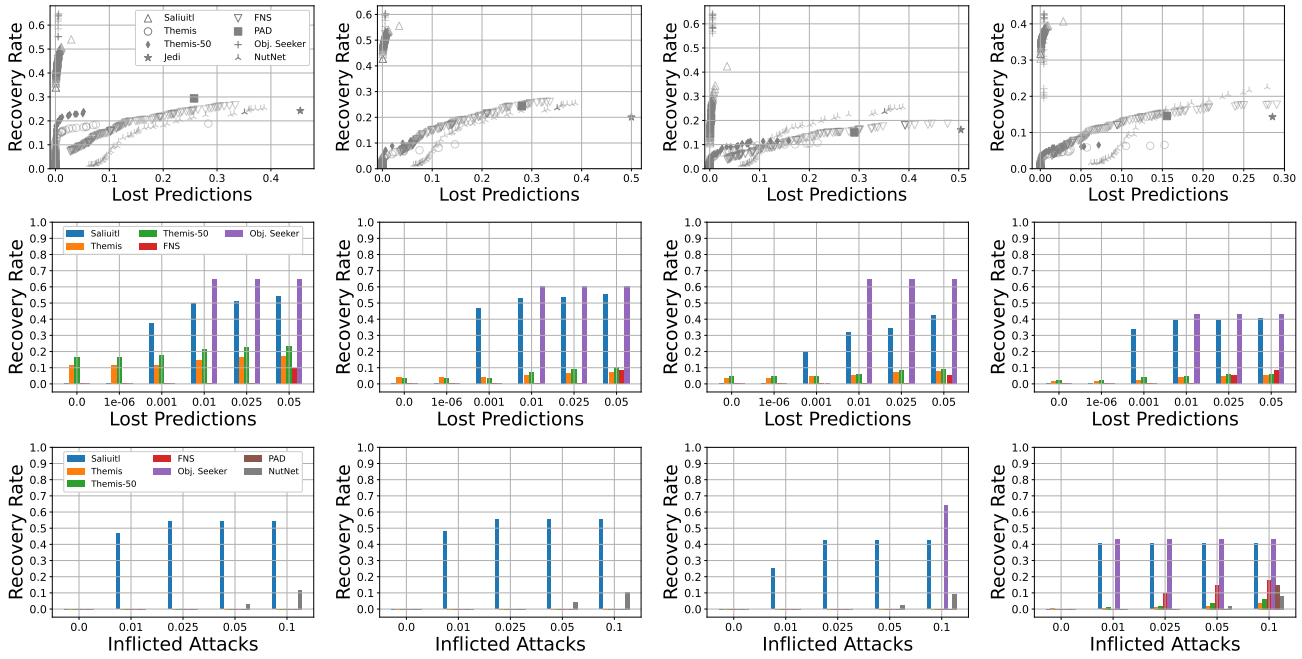


Figure 6. Recovery results for Object Detection (Pascal VOC). From left to right: single, double, triangular, and multi-object patch attacks.

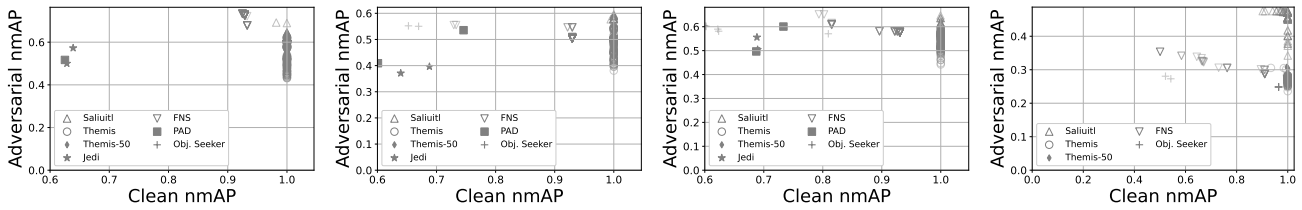


Figure 7. nmAP results for INRIA. From left to right: single, double, triangular, and multi-object patch attacks.

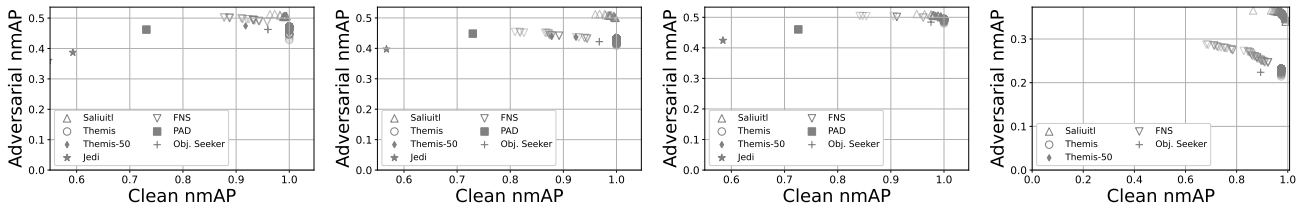


Figure 8. nmAP results for Pascal VOC. From left to right: single, double, triangular, and multi-object patch attacks.

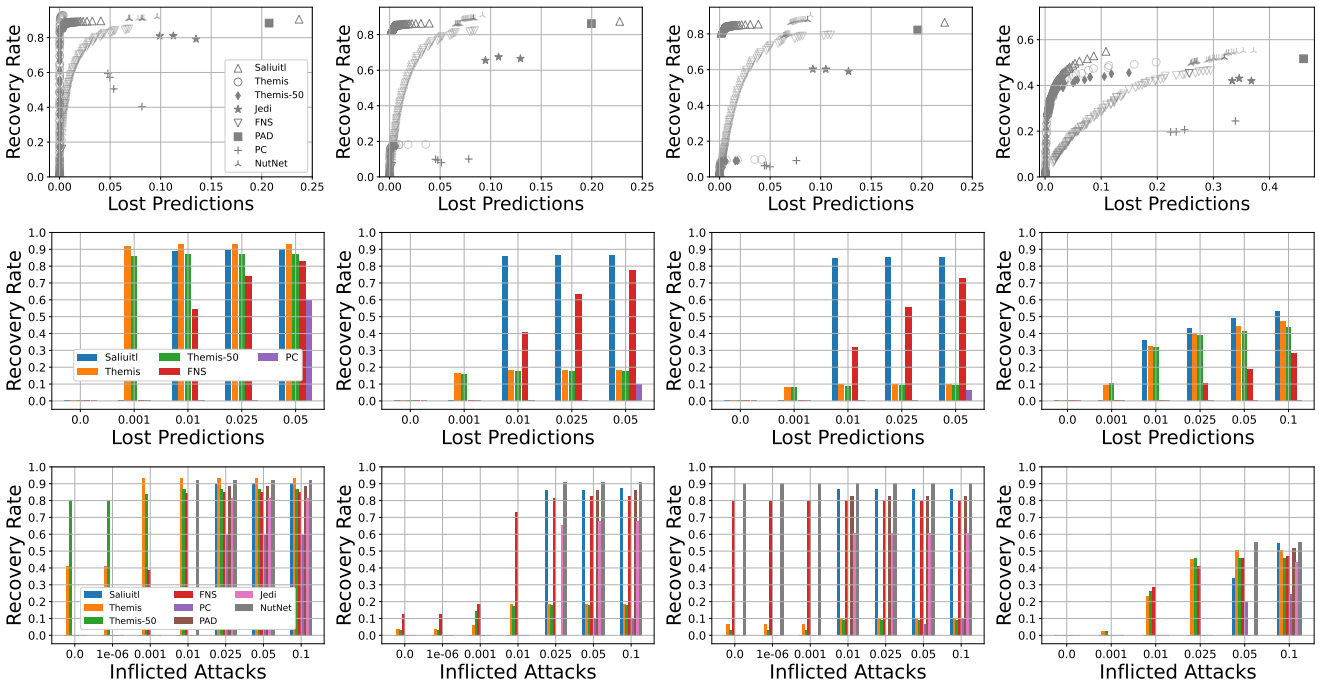


Figure 9. Recovery results for Image Classification (ImageNet). From left to right: single, double, quadruple, and triangular patch attacks.

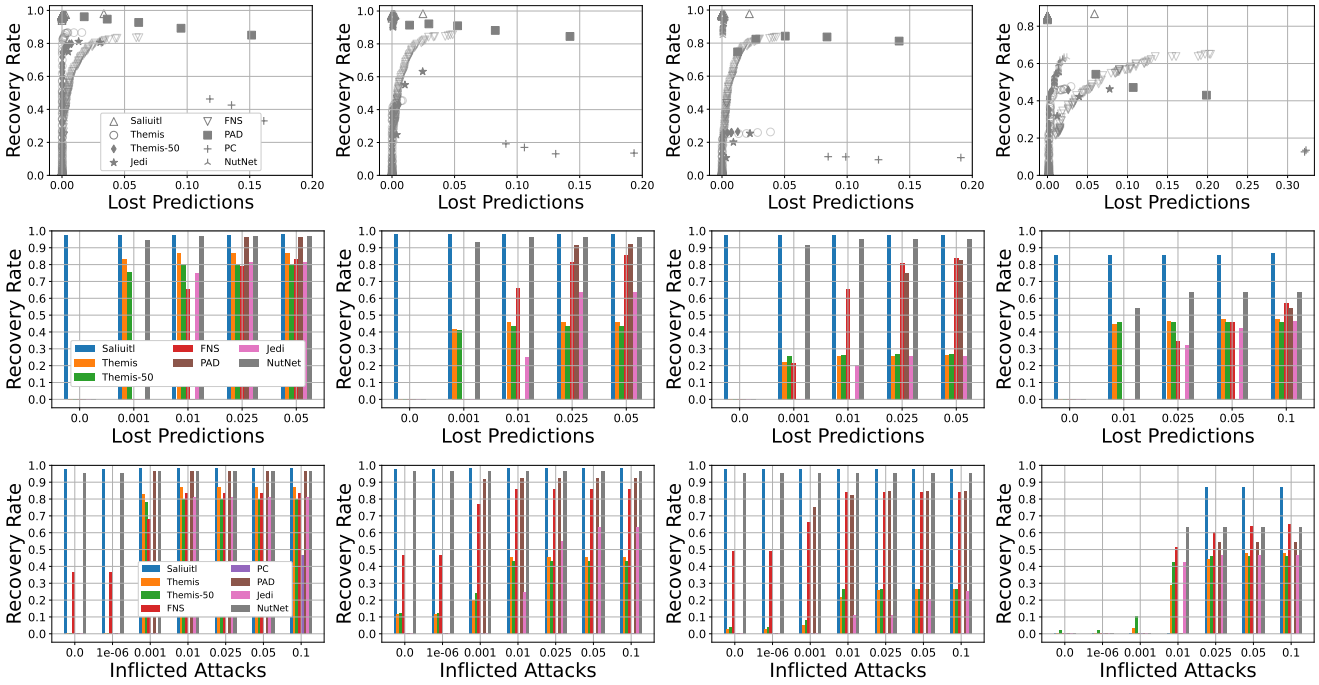
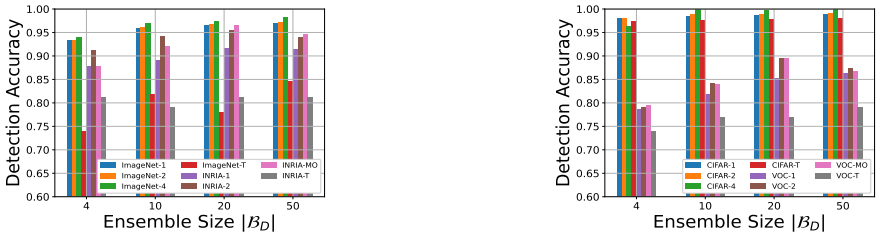
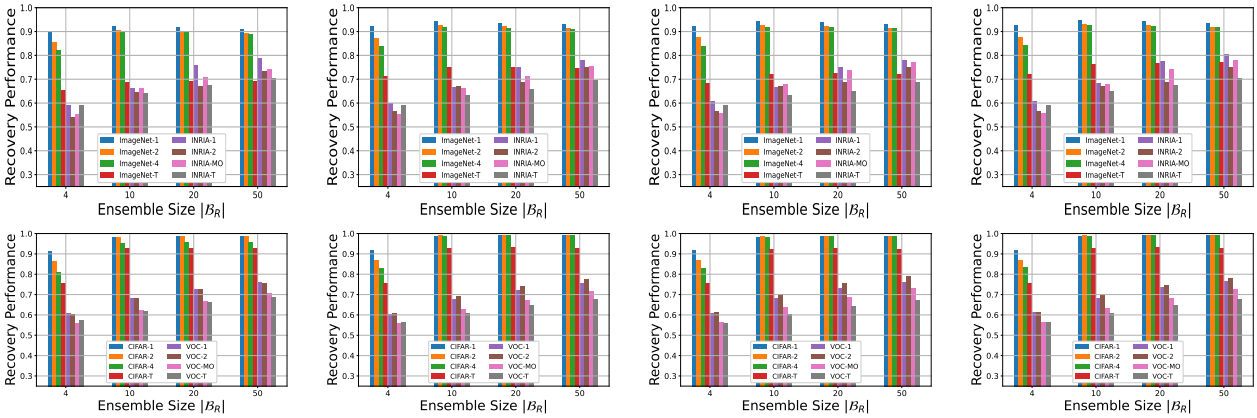


Figure 10. Recovery results for Image Classification (CIFAR-10). From left to right: single, double, quadruple, and triangular patch attacks.



(a) INRIA/ImageNet (b) VOC/CIFAR-10.

Figure 11. *Saliuitl*'s detection accuracy vs. the detection ensemble size $|\mathcal{B}_D|$, for all datasets.



(a) $|\mathcal{B}_D| = 4$ (b) $|\mathcal{B}_D| = 10$ (c) $|\mathcal{B}_D| = 20$. (d) $|\mathcal{B}_D| = 50$.

Figure 12. *Saliuitl*'s recovery performance on INRIA and ImageNet (top), and on Pascal VOC and CIFAR-10 (bottom) as a function of $|\mathcal{B}_R|$, illustrated for different choices of $|\mathcal{B}_D|$.

Ablation Studies

Impact of Ensemble Size

In Section 4.2 we assess the impact of the cardinality of the sets $\mathcal{B}_D$ and $\mathcal{B}_R$, i.e., that of the ensemble size on the performance and computational cost of *Saliuitl* when $\alpha^* = 0.5$. In what follows, we conduct an in-depth analysis of the impact of ensemble size on both performance and computational cost. As before, we consider four ensemble sets

$$\mathcal{B}_B := \left\{ \frac{x \cdot \max(\mathbf{m}_i)}{|\mathcal{B}_B|} \right\}_{x=0}^{|\mathcal{B}_B|-1}, \quad |\mathcal{B}_B| \in \{4, 10, 20, 50\}.$$

As in Section 4.2, when we state that one of $\mathcal{B}_D$ or $\mathcal{B}_R$ has cardinality $|\mathcal{B}_B|$, it means we choose that set to be $\mathcal{B}_B$ as described above.

Detection Accuracy and Recovery Performance

In Figure 11(a), we present results for attack detection accuracy on object detection (INRIA) and image classification (ImageNet) as a function of the ensemble size pertinent to attack detection $|\mathcal{B}_D|$. For each ensemble size and attack scenario, the reported detection accuracy is the best possible, i.e., instead of setting $\alpha^* = 0.5$, we tune α^* in each case to get the best possible accuracy; Figure 11(b) shows the corresponding results for Pascal VOC and CIFAR-10. Our results show that increasing the ensemble size can be beneficial for attack detection accuracy in both object detection (INRIA/Pascal VOC) and image classification (ImageNet/CIFAR-10). In particular, detection accuracy increases for all datasets when increasing $|\mathcal{B}_D|$ from 4 to 10. However, we also observe that increasing ensemble size is beneficial only up to a point, as a drop in attack detection accuracy can be observed for all object detection scenarios, with the exception of triangular patch attacks, when the cardinality of $\mathcal{B}_D$ is increased from 20 to 50. Interestingly, the detection accuracy for triangular patches on ImageNet decreases when increasing $|\mathcal{B}_D|$ from 10 to 20, but it then reaches its highest point when increasing $|\mathcal{B}_D|$ from 20 to 50. We conjecture that for all patch attacks, using an ensemble size that is too small leads to patterns that are too coarse in the ensemble attribute vector $\mathbf{s}$, leading to an accuracy reduction for our detector network *AD*; moreover, if the ensemble size is too large, this might lead to overfitting and a reduction in detection accuracy, as was observed for all non-triangular attacks on object detection.

In Figure 12 we display the recovery performance

achieved by each choice of $|\mathcal{B}_R|$ across all datasets and attack models, for different choices of $|\mathcal{B}_D|$. For a given choice of $|\mathcal{B}_D|$, we determine the value of α^* that achieves the best detection accuracy, and we use the corresponding detection results to determine when to perform the recovery stage. Recall the evaluation metrics described in Section 4.1; we now define recovery performance (RP) as the fraction of clean and effectively attacked inputs (i.e., images in $\mathcal{X}_1' \cup \mathcal{X}_1$) that yield a correct output upon applying $\mathcal{R}_\phi$. This can be expressed in terms of the recovery rate (RR) and lost prediction rate (LPR) as

$$RP = \frac{RR + 1 - LPR}{2}.$$

In general, our results show that for a given choice of $|\mathcal{B}_D|$, increasing $|\mathcal{B}_R|$ is beneficial for recovery performance. Beyond very small detriments for recovery performance for image classification scenarios (ImageNet/CIFAR) when $|\mathcal{B}_R| \geq 20$ and $|\mathcal{B}_D| \in \{4, 10, 20\}$, the clear trend is that increasing $|\mathcal{B}_R|$ allows *Saliuitl* to attain a better recovery performance. Note that we computed these results following our DBSCAN-based implementation for attribute extraction and mask preprocessing. The main takeaway is that while increasing the sizes of $\mathcal{B}_D$ and $\mathcal{B}_R$ is a sensible approach to improve recovery performance, larger sets of saliency thresholds might not always be desirable, even from a detection accuracy and recovery performance perspective with no regard for computational efficiency.

Computational Cost.

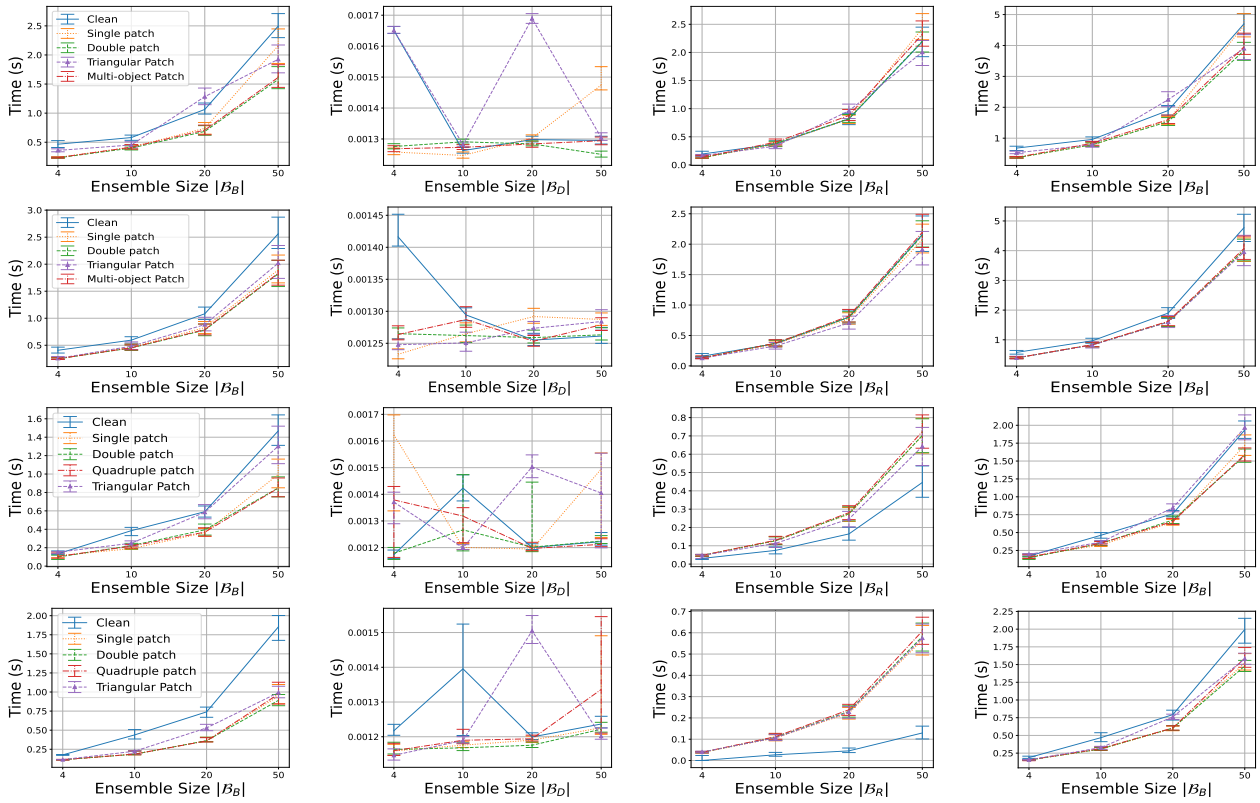
Next, we evaluate how the computational cost of our defense scales with the shape and number of patches, and with the cardinality of the saliency threshold sets $\mathcal{B}_D$ and $\mathcal{B}_R$. In Figure 13 we report the computational cost for different components of *Saliuitl* in all patch scenarios and all datasets, where each row in the figure corresponds to a dataset. All plots show median values across each dataset with error bars indicating the first and third quartiles. For any type of patch attack, we report statistics on all attacked inputs (i.e., images in $\mathcal{X}_0'$, as described in Section 4.1), and we report clean input statistics over all clean inputs (i.e., all images in $\mathcal{X}_0$).

For the attack detection stage, we show statistics for the time it takes to construct the clustering-based attribute ensemble vector $\mathbf{s}$ we use in our evaluation and the time it takes to compute the detection score for an input using *AD* in Figures 13(a) and 13(b), respectively. We present the computational cost as a function of both the size of $\mathcal{B}_D$, and the type of adversarial patch applied to the input.

The computational cost of clustering for clean images is larger than for attacked images for both tasks. This follows from the homogeneity of clean images, resulting in heavier clustering computations. Triangular patches also seem to have a higher cost than any number of rectangular patches. We conjecture that due to the use of square kernels in the convolutional layers, triangular patches result in sparser patterns of important neurons, leading to more homogeneous feature maps. While the cost of detection in Figure 13(b) shows no clear relationship to the type of attack or even the ensemble size, it is orders of magnitude smaller than the cost of clustering, hence we can consider that the cost of the detection stage *is* the cost of clustering. We can observe that this computational time is insensitive to the number of adversarial patches in the image, and finally, that the execution time increases with the ensemble size at a rate that is approximately linear, in line with the fact that the attributes used for detection are generated by performing $|\mathcal{B}_D|$ iterations of the first loop in Algorithm 1. It is important to emphasize that other choices of attributes are possible, and while our choice used for evaluation illustrates that *Saliuitl* does not need to increase computational cost to handle non-contiguous patches, this does not rule out that one might choose different attributes that scale differently depending on the number of adversarial regions in the input.

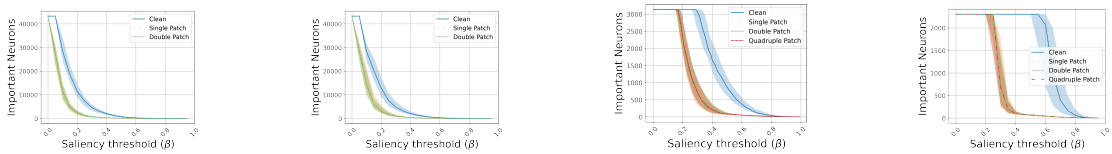
In Figure 13(c) we present the corresponding computational cost for recovery, note that in these results we assume $\mathcal{B}_D = \mathcal{B}_R = \mathcal{B}_B$, hence although our mask preprocessing scheme for evaluation relies on DBSCAN, no clustering is performed in the recovery stage. Moreover, to disregard abrupt input-dependent reductions in the computational time of recovery, we disable the second stopping condition that applies only to image classification, which terminates the recovery stage when the predicted label of the inpainted input changes. We observe that recovery is not dependent on the type of the adversarial patch attack, but for image classification, clean images have a notably lower cost. This is dependent on the first stopping condition for recovery, i.e., that the amount of important neurons suffices to cover at least half of the total input region. Our results are congruent with preliminary experiments indicating that in the case of image classification, the threshold for which this condition holds true is notably higher for clean images than for attacked images. The figure also shows that as the size of $\mathcal{B}_R$ increases, the computational cost again tends to increase at an approximately linear rate. Importantly, recovery is notably more computationally expensive than detection, which highlights the advantage of the two stage design of *Saliuitl*, which only performs recovery if an attack is detected. We present the total cost of *Saliuitl* in Figure 13(d). In general, the computational cost increases with ensemble size at an approximately linear rate, and inputs with triangular patches or no patches incur in a slightly higher cost due an increased cost during the construction of the ensemble attribute vector $\mathbf{s}$.

We mentioned that preliminary experiments show that the first stopping condition of the recovery stage occurs earlier for image classification than for object detection. In



(a) Clustering (b) Detection (c) Recovery (d) Total

Figure 13. *Saliuitl*'s computational cost as a function of the ensemble size and the number of patches, using all datasets. From top to bottom: INRIA, Pascal VOC, ImageNet, CIFAR-10. Error bars represent the first and third quartiles across the dataset.



(a) INRIA (b) Pascal VOC (c) ImageNet (d) CIFAR-10

Figure 14. Number of important neurons for different saliency thresholds for all datasets. Solid lines represent median values, and shaded regions represent the first and third quartiles.

Figure 14 we present the relevant results from said experiments. The figure shows how the number of important neurons changes for clean images and images with single and multiple rectangular patches on subsets of each dataset. In the main loop of the recovery stage, *Saliuitl* reduces the importance threshold at each iteration, which corresponds to going from right to left in the plots shown in Figure 14, and for each iteration we terminate the recovery stage if the region to be inpainted, dictated by the important neurons, corresponds to an image region at least half as large as the whole image. In Figure 14 we observe that as the number of important neurons increases from right to left, the threshold (iteration) at which the maximum value is reached for clean images, is higher (occurs earlier) than for attacked images, and in the case of ImageNet and CIFAR-10, the difference is particularly large. Hence, we attribute the lower recovery times for image classification to this phenomenon. Note that this is not a general statement regarding a fundamental difference for patch attacks on object detection and classification; our results are dependent on our choice of victim models, shallow layers, datasets, and patch attack models. This is further emphasized by the differences between ImageNet and CIFAR-10 in Figure 14, which are congruent with their differences in Figure 13(c).

Table 5. Detection stage ablation. Best Recovery Rate (*RR*) - Lost Prediction Rate (*LPR*) tradeoffs for single (1), double (2), quadruple (4), triangular (T), and multi-object (MO) patch attack scenarios. Best attribute extraction and attack detection methods per row are in bold and underlined, respectively. Note *Saliuitl* represents the default DBSCAN-based attributes and the 1D-CNN attack detector *AD*.

Attack		Ensemble Attributes		Attack Detector
	Saliuitl RR/LPR	*Saliuitl*-impneu RR/LPR	*Saliuitl*-alt RR/LPR	*Saliuitl*-svm RR/LPR
INRIA-1	**0.5909/0.0152**	0.5909/0.0303	0.5909/0.0303	0.5606/0.0
INRIA-2	**0.3871/0.0**	0.3548/0.0	0.3710/0.0	0.3710/0.0
INRIA-T	**0.4737/0.0526**	**0.4737/0.0526**	**0.4737/0.0526**	0.4743/0.0526
INRIA-MO	0.4749/0.0	0.4860/0.0335	**0.4804/0.0**	0.4860/0.0056
VOC-1	**0.5404/0.0293**	**0.5404/0.0293**	**0.5404/0.0293**	0.5382/0.0243
VOC-2	**0.5376/0.0125**	0.5556/0.0337	0.5556/0.0337	0.5541/0.0274
VOC-T	**0.4244/0.0348**	**0.4244/0.0348**	**0.4244/0.0348**	0.4244/0.0348
VOC-MO	**0.3955/0.0095**	0.4069/0.0283	0.3832/0.0028	0.3991/0.0095
ImageNet-1	**0.8869/0.0071**	0.8809/0.0129	0.8763/0.0135	0.8721/0.0128
ImageNet-2	**0.8535/0.0061**	0.8523/0.0117	0.8470/0.0093	0.8442/0.0125
ImageNet-4	0.8436/0.0086	0.8562/0.0089	**0.8544/0.0058**	0.8513/0.0099
ImageNet-T	0.5065/0.0612	0.5896/0.0834	**0.5959/0.0812**	0.5092/0.0427
CIFAR-1	**0.9738/0.0008**	0.9645/0.0034	0.9560/0.0044	0.9708/0.0126
CIFAR-2	**0.9789/0.0006**	0.9688/0.0	0.9835/0.0247	0.9764/0.0097
CIFAR-4	0.9747/0.0	**0.9778/0.0**	0.9673/0.0005	0.9778/0.0085
CIFAR-T	**0.8566/0.0**	0.8411/0.0017	0.8428/0.0086	0.8601/0.0190

Table 6. Detection stage ablation. Best adversarial-clean nmAP tradeoffs for object detection. For single (1), double (2), triangular (T), and multi-object (MO) patch attack scenarios. Best attribute extraction and attack detection methods per row are in bold and underlined, respectively.

Attack		Ensemble Attributes		Attack Detector
	Saliuitl Adv./Clean	*Saliuitl*-impneu Adv./Clean	*Saliuitl*-alt Adv./Clean	*Saliuitl*-svm Adv./Clean
INRIA-1	0.6897/0.9998	0.6362/1.0	**0.6933/1.0**	0.6922/1.0
INRIA-2	**0.5998/1.0**	0.5786/1.0	0.5786/1.0	0.5787/1.0
INRIA-T	**0.7034/0.9987**	0.7007/0.9566	0.7007/0.9566	0.7007/0.9566
INRIA-MO	0.4737/0.9999	0.4716/1.0	**0.4776/0.9999**	0.4767/0.9999
VOC-1	0.5094/0.9950	0.5145/0.9966	**0.5155/0.9993**	0.5167/0.9964
VOC-2	**0.5088/0.9940**	0.4977/0.9958	0.5035/0.9934	0.5012/0.9987
VOC-T	0.5043/0.9942	0.5182/0.9800	**0.5041/0.9994**	0.4999/0.9949
VOC-MO	0.3563/0.9877	0.3392/0.9958	**0.3634/0.9944**	0.3590/0.9997

Impact of Attribute Extraction and Detection

For our evaluation, the implementation of the detection stage relies on four DBSCAN-based attributes to construct the ensemble attribute vectors used for detection, moreover, we propose to use the one-dimensional CNN detector network *AD* to compute the detection score. In this section we evaluate alternative attributes for detection, and we also explore the possibility of using a binary classifier that is different from the proposed *AD*. Note that we still use the DBSCAN-based approach during the recovery stage regardless of the attribute extraction or attack detection scheme. Moreover, in all cases we use the default salience threshold sets $\mathcal{B}_D$ and $\mathcal{B}_R$ defined in Section 4.1, that is, $\mathcal{B}_D = \mathcal{B}_R = \{\frac{x \cdot \max(\mathbf{m}_i)}{20}\}_{x=0}^{19}$.

Alternative Attributes

We propose to use two sets of attributes that differ from our DBSCAN-based attributes proposed in Section 4.1. In *Saliuitl*-impneu we use only the first attribute used for evaluation, that is, the number of important neurons. This allows us to evaluate how well our scheme can maintain performance using a simpler ensemble attribute vector that has a significantly lower computational cost. In *Saliuitl*-alt we extract four attributes from each binary feature map: mean, variance, skewness, and kurtosis; these statistics are computed across all neurons in the binary feature map. Using these alternative attributes also results a reduction in execution time for our detection stage compared to the default DBSCAN-based attributes. We re-train *AD* using these alternative attributes. The training of *AD*, including the normalization across each attribute for $\mathbf{s}$ remains unchanged from our previous description; note that the training of *AD* can be applied to any choice of attributes as long as they are preprocessed adequately before being fed into *AD*.

Object Detection The first eight rows in Table 5 show the best tradeoffs in terms of recovery and lost prediction rates in object detection scenarios for our default implementation of *Saliuitl* and our proposed alternative attribute extraction approaches. The detailed tradeoffs can be found in Figures 15 and 16. We observe that the default implementation is as good or better than the alternative attribute extraction approaches for all object detection scenarios with the excep-

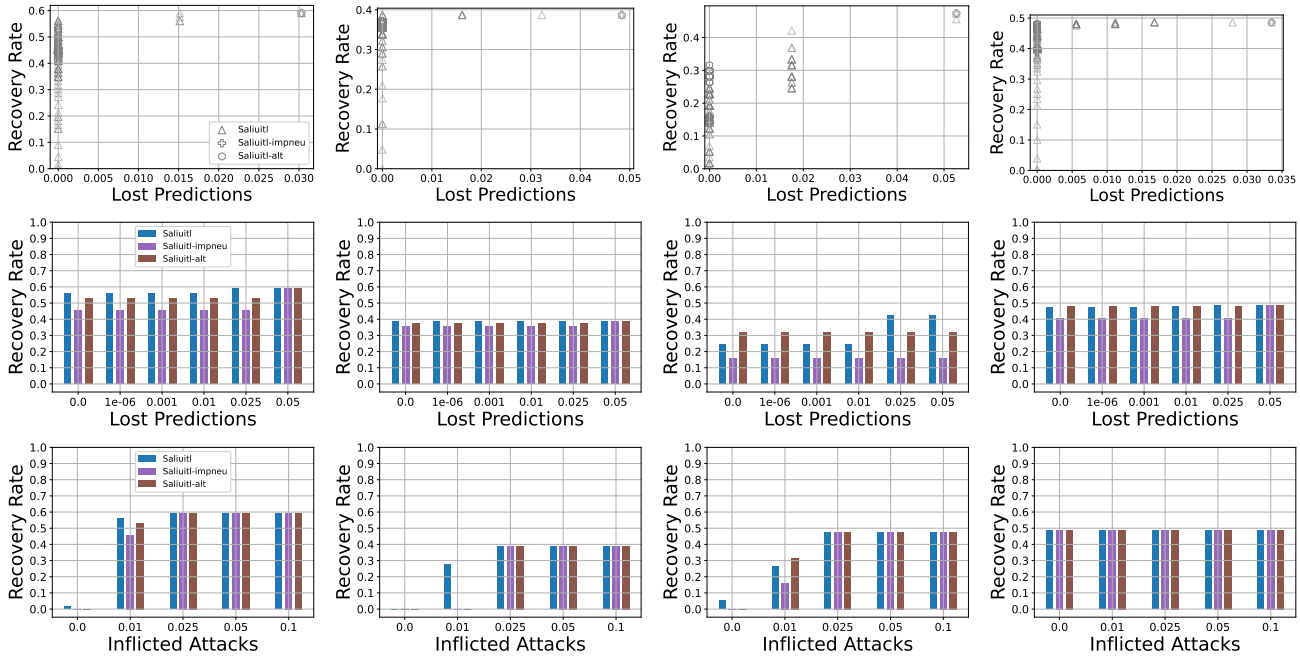


Figure 15. Attribute extraction ablation. Recovery results for Object Detection (INRIA). From left to right: single, double, triangular, and multi-object patch attacks.

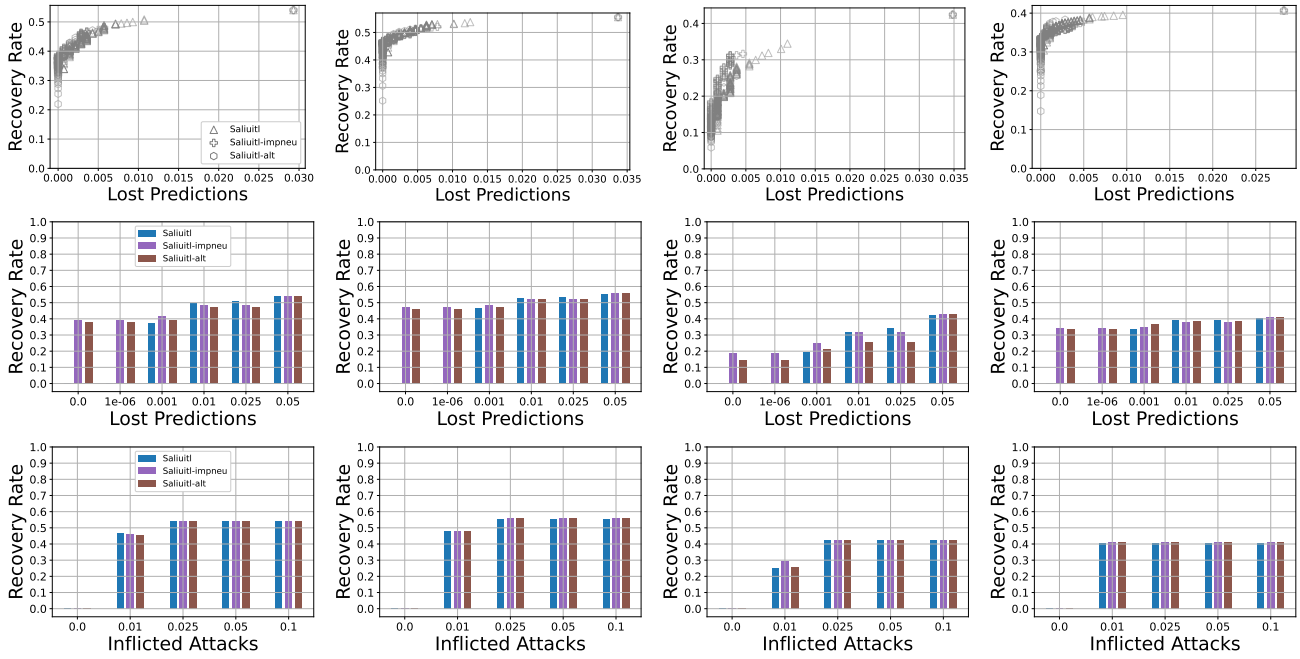


Figure 16. Attribute extraction ablation. Recovery results for Object Detection (Pascal VOC). From left to right: single, double, triangular, and multi-object patch attacks.

tion of multi-object patches for INRIA, where *Saliuitl*-alt achieves a better tradeoff. While all three instances of *Saliuitl* perform rather similarly, the detailed tradeoffs show that our default implementation is often able to achieve a higher recovery rate by trading off an increase in lost prediction rate that *Saliuitl*-impneu and *Saliuitl*-alt are unable to achieve (e.g, see the tradeoffs achieved by *Saliuitl* at a lost prediction rate of 0.015 for single patches in INRIA).

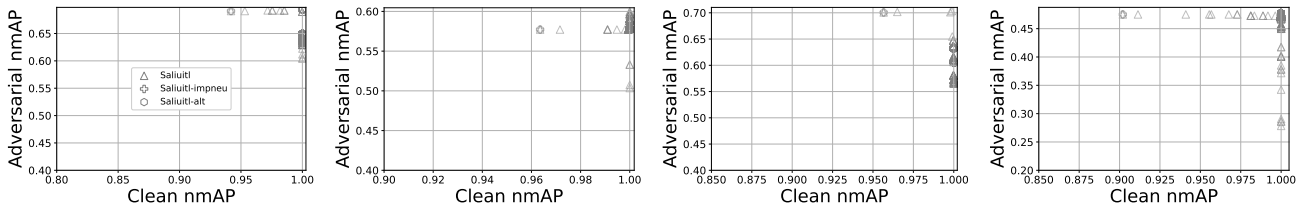


Figure 17. Attribute extraction ablation. nmAP results for INRIA. From left to right: single, double, triangular, and multi-object patch attacks.

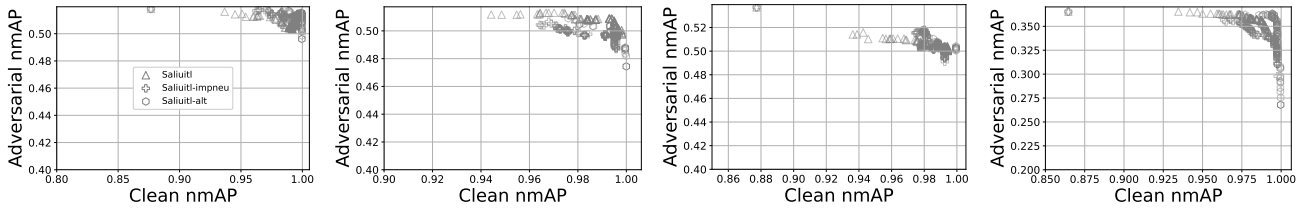


Figure 18. Attribute extraction ablation. nmAP results for Pascal VOC. From left to right: single, double, triangular, and multi-object patch attacks.

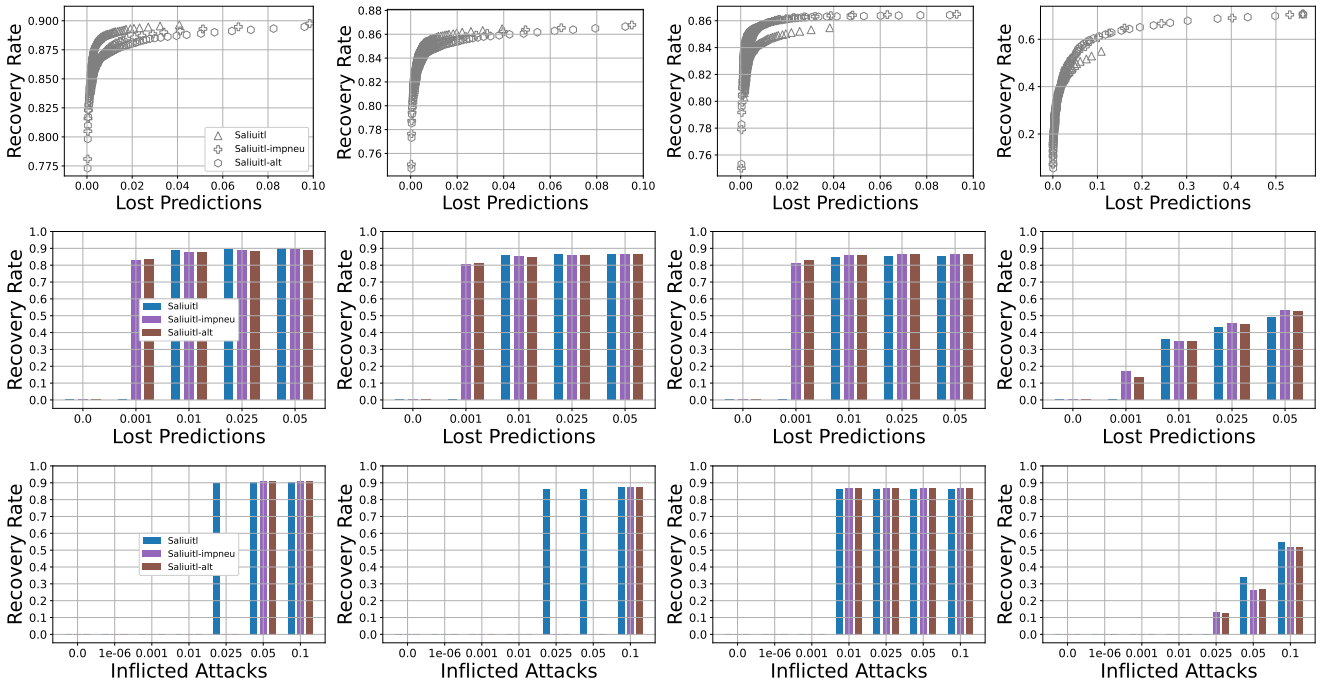


Figure 19. Attribute extraction ablation. Recovery results for Image Classification (ImageNet). From left to right: single, double, quadruple, and triangular patch attacks.

Regarding the performance of these methods at fixed thresholds on lost prediction and inflicted attack rates, we find that the default approach has the upper hand in most INRIA scenarios (except for triangular patches), yet *Saliuitl*-impneu and *Saliuitl*-alt achieve relatively high recovery rates at lost prediction rates that are not attainable for *Saliuitl* on Pascal VOC.

The best tradeoffs in terms of clean and adversarial nmAP are shown in Table 6; the detailed tradeoffs are shown in Figures 17 and 18. *Saliuitl*-alt is the best performing attribute extraction approach in most scenarios, although our default implementation is better for triangular patches on INRIA and double patches on both INRIA and Pascal VOC; notably we find that the superiority in the at-

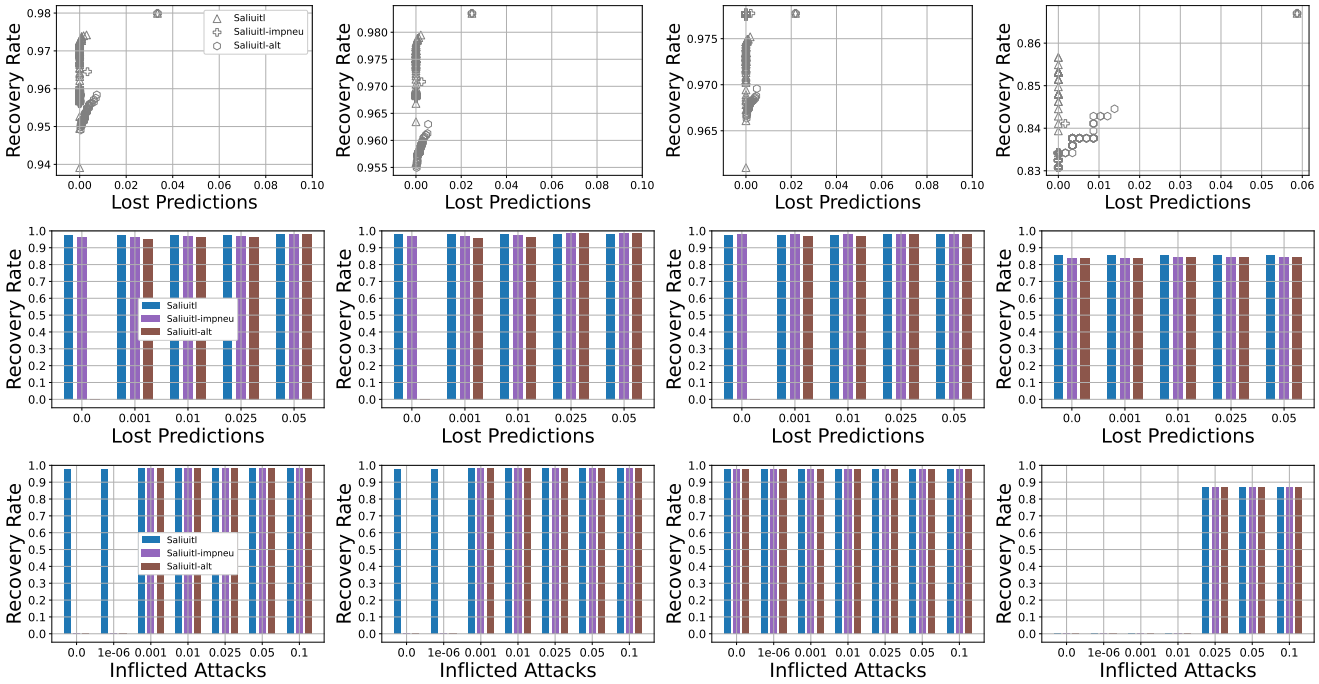


Figure 20. Attribute extraction ablation. Recovery results for Image Classification (CIFAR-10). From left to right: single, double, quadruple, and triangular patch attacks.

tainable adversarial nmAP for the default implementation for double patches on INRIA is the only case where any approach stands out with such a large margin. While *Saliuitl* and *Saliuitl*-alt have quite similar performances, *Saliuitl*-impneu has a notably lower performance in some scenarios (e.g. single patches on INRIA); this indicates that extracting multiple attributes that contribute with distinct information for each input image can result in notable benefits for overall performance.

Image Classification The last eight rows of Table 5 show the best recovery/lost prediction rate tradeoffs achieved by the three attribute extraction approaches. The detailed tradeoffs are shown in Figures 19 and 20. We find that the default approach performs best in most scenarios, except for quadruple and triangular patches on ImageNet, where *Saliuitl*-alt is better, and quadruple patches on CIFAR-10, where *Saliuitl*-impneu is better. Overall the three methods have similar performances, although *Saliuitl* has notably worse performance for triangular patches on ImageNet, it is the best performing approach for triangular patches on CIFAR-10. Overall, *Saliuitl*-alt has the upper hand regarding recovery rate at fixed thresholds on the lost prediction rate, while the default *Saliuitl* has the edge for fixed thresholds on inflicted attacks. The image classification results further support that using a single attribute (*Saliuitl*-impneu) will most likely hinder performance, whereas using different attributes (*Saliuitl*-alt) leads to a more moderate impact (that can be either negative or positive).

Alternative Detector

Instead of feeding the ensemble attribute vector $\mathbf{s}$ to *AD*, we explore the possibility of using an SVM classifier; we refer to this alternative implementation as *Saliuitl*-svm. We use the same training data for each dataset that we used to train *AD*, however, we do not use any data for validation and instead fit the SVM to all the training data. In particular we use sci-kit learn's support vector classification (SVC) implementation [30] (version 1.2.2) with default parameters (radial basis function kernel) and we use its class probability estimates based on Platt's scaling as the detection score. We employ the four default DBSCAN-based attributes described in Section 4.1.

Object Detection The first eight rows of Table 5 show the best recovery/lost prediction rate tradeoffs achieved by *Saliuitl* and *Saliuitl*-svm for object detection. Detailed tradeoffs are shown in Figures 21 and 22. Both detection approaches have similar performances, and in fact, *Saliuitl*-svm enjoys equal or better tradeoffs for all scenarios except for single and double patches on INRIA. Moreover, we noted that compared to other attribute extraction approaches, the default *Saliuitl* was able to achieve certain tradeoffs inaccessible to *Saliuitl*-impneu and *Saliuitl*-alt, while this is still true when comparing to *Saliuitl*-svm in some cases (e.g., single patches on INRIA), *Saliuitl*-svm is able to attain such tradeoffs in scenarios where *Saliuitl*-

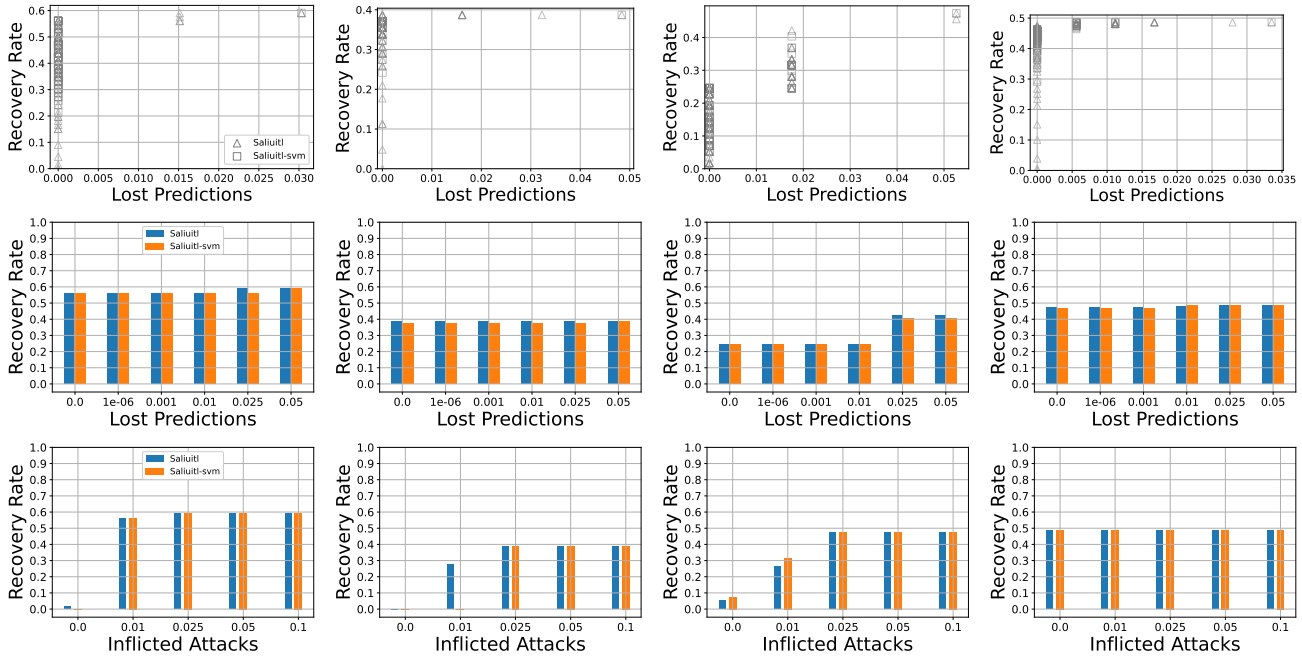


Figure 21. Attack detector ablation. Recovery results for Object Detection (INRIA). From left to right: single, double, triangular, and multi-object patch attacks.

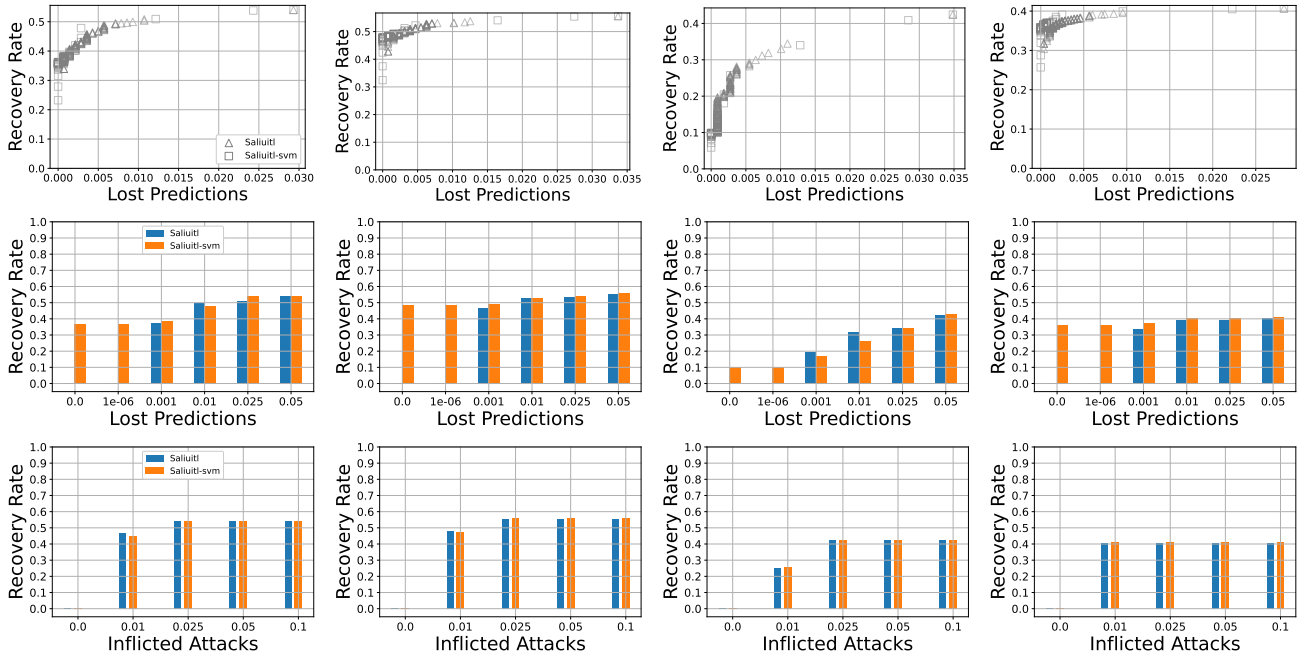


Figure 22. Attack detector ablation. Recovery results for Object Detection (Pascal VOC). From left to right: single, double, triangular, and multi-object patch attacks.

impneu and *Saliuitl*-alt are not (e.g., triangular patches on INRIA); this suggests that the choice of attributes is more determinant than the choice of attack detector for the precise tradeoffs achievable by *Saliuitl*. From the recovery rates at fixed thresholds on lost predictions we observe that *Saliuitl* and *Saliuitl*-svm are on par for INRIA, but *Saliuitl*-svm has the upper hand on Pascal VOC, and is even able to attain a relatively high recovery rate at zero lost prediction rates.

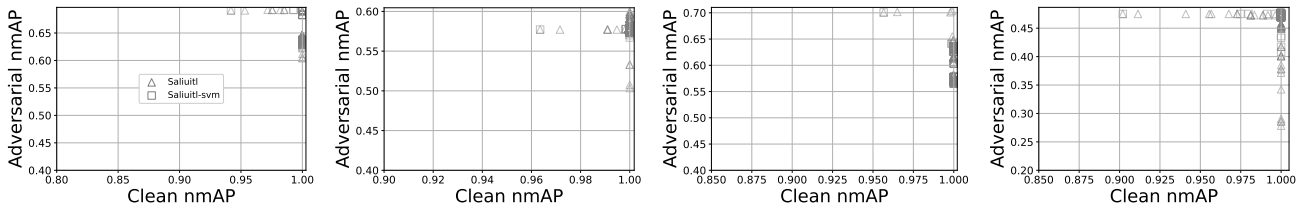


Figure 23. Attack detector ablation. nmAP results for INRIA. From left to right: single, double, triangular, and multi-object patch attacks.

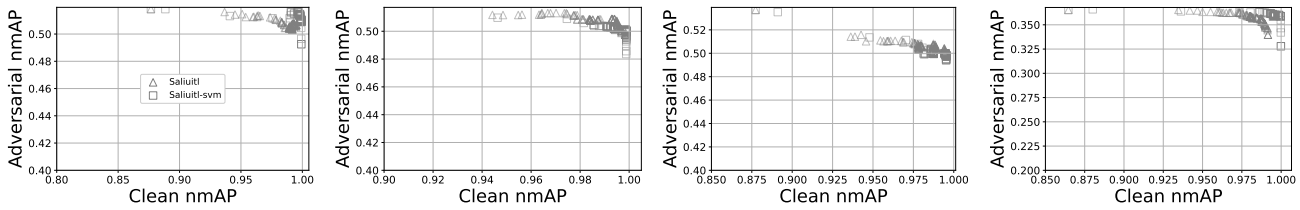


Figure 24. Attack detector ablation. nmAP results for Pascal VOC. From left to right: single, double, triangular, and multi-object patch attacks.

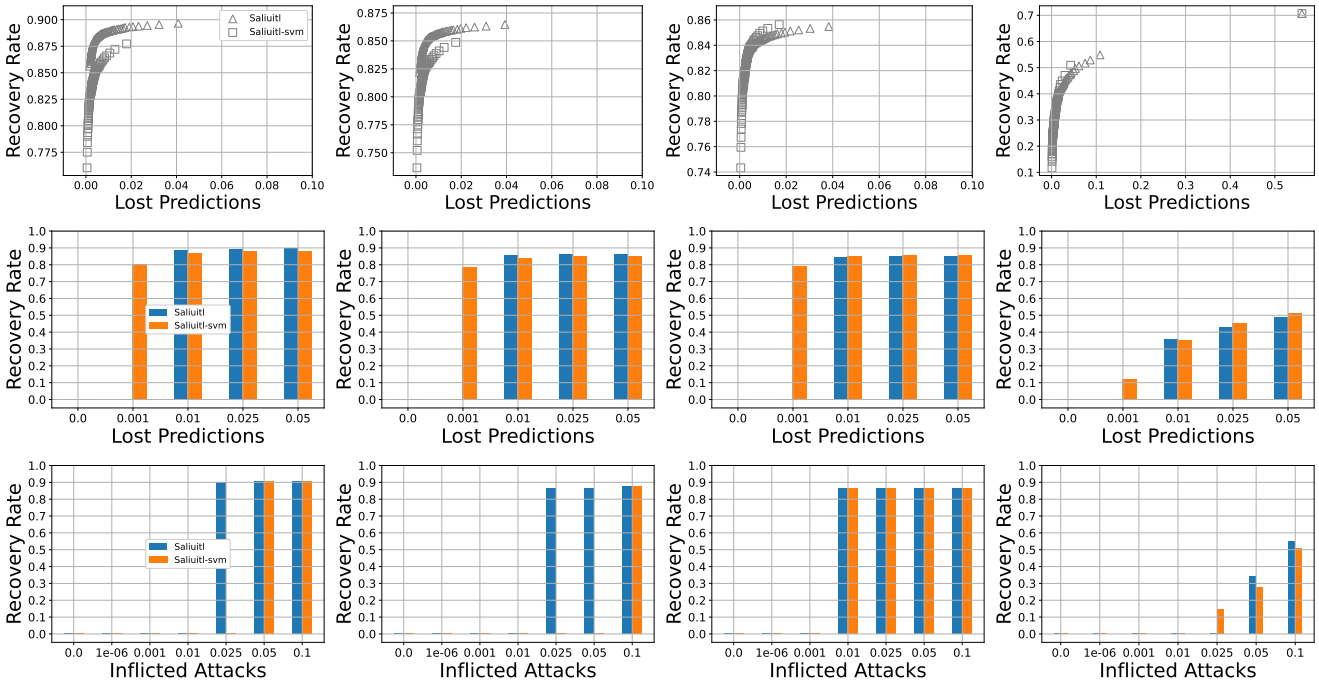


Figure 25. Attack detector ablation. Recovery results for Image Classification (ImageNet). From left to right: single, double, quadruple, and triangular patch attacks.

For inflicted attacks, *Saliuitl* outperforms *Saliuitl*-svm for double attacks on INRIA, but both methods are essentially similar on all other scenarios.

The comparison between *Saliuitl* and *Saliuitl*-svm in terms on clean and adversarial nmAP tradeoffs is shown in Table 6 and Figures 23 and 24. We observe that *Saliuitl* performs best for double and triangular patches, while *Saliuitl*-svm is better for single and multi-object patches. Both methods perform quite similarly across all scenarios; similar to *Saliuitl*-impneu and *Saliuitl*-alt, *Saliuitl*-svm is unable to achieve *Saliuitl*'s adversarial nmAP for double patches on INRIA, regardless of its clean nmAP reduction. Overall the results for object detection show that although the choice of *AD* as a 1D CNN is in line with our proposed use of ensemble attributes, alternative detectors can perform comparatively to *AD* when using our proposed ensemble attribute

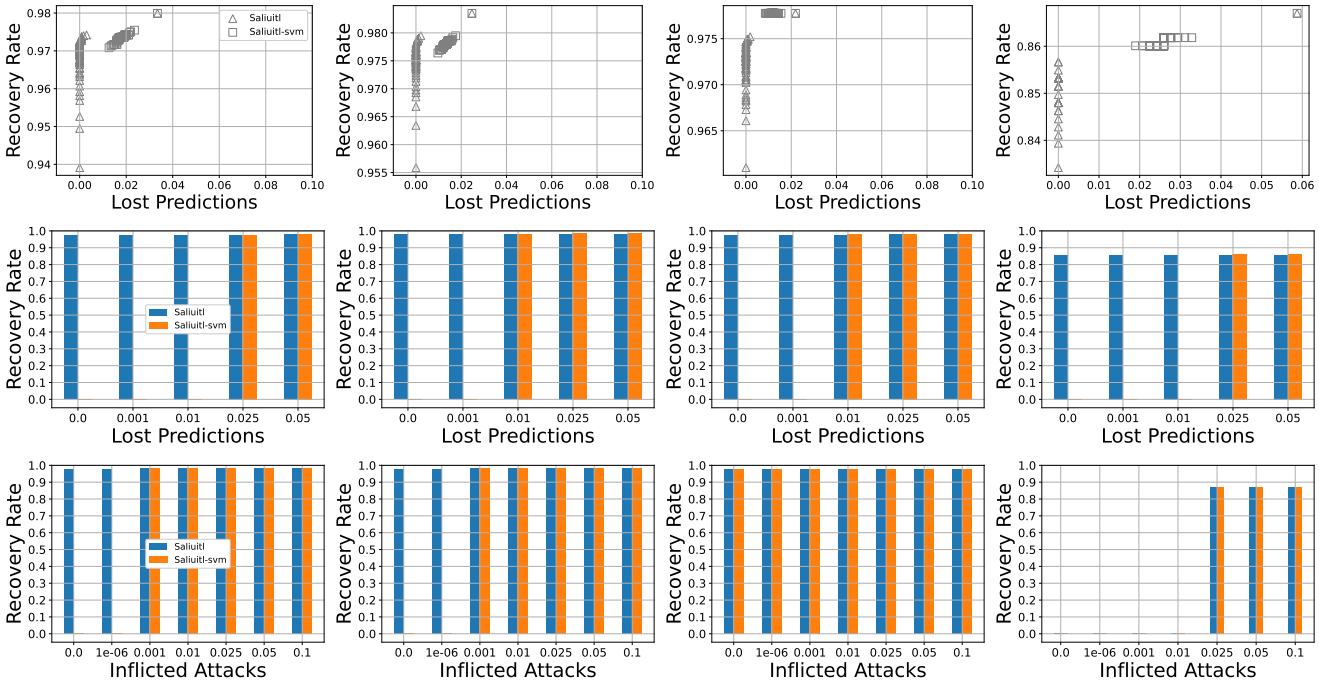


Figure 26. Attack detector ablation. Recovery results for Image Classification (CIFAR-10). From left to right: single, double, quadruple, and triangular patch attacks.

vectors as inputs. This is also congruent with the data efficiency demonstrated by the performance of both *AD* and the alternative SVM classifier, which need only a relatively small amount of data to achieve good performances on the evaluation datasets compared to existing defenses.

Image Classification The last eight rows of Table 5 show the comparison of *Saliuitl* and *Saliuitl*-svm in terms of the best recovery and lost prediction rate tradeoffs for image classification; the detailed tradeoffs are shown in Figures 25 and 26. *Saliuitl* is able to find better tradeoffs than *Saliuitl*-svm in most scenarios, with the exception of quadruple and triangular patches on ImageNet, yet we observe the performances of both methods are quite close to one another in these two scenarios. On CIFAR-10, while *Saliuitl* achieves the best tradeoffs, *Saliuitl*-svm is able to surpass its recovery rate at a higher lost prediction rate in most scenarios. Moreover, for fixed limits on the lost prediction rate, *Saliuitl*-svm tends to outperform *Saliuitl* on ImageNet, while the opposite is true for CIFAR-10. In terms of inflicted attacks, *Saliuitl* is better than *Saliuitl*-svm in most scenarios, with the exception of triangular patches on ImageNet. In line with the object detection results, these results indicate that an adequate choice of attributes allows *Saliuitl* to retain (and even improve) its performance when using alternative attack detection schemes. This represents a significant advantage for *Saliuitl*, as the potential flexibility for the attack detection scheme might be useful in applications with constrained resources for both training and inference. We consider the exploration of how different classifiers scale with the size of the ensemble $\mathcal{B}_D$ or the number of attributes n_f to be the scope of future work.

Table 7. Recovery stage ablation. Best Recovery Rate (*RR*) - Lost Prediction Rate (*LPR*) tradeoffs for single (1), double (2), quadruple (4), triangular (T), and multi-object (MO) patch attack scenarios. Best mask preprocessing and inpainting methods per row are in bold and underlined, respectively. Note *Saliuitl* represents the default methods for both mask preprocessing and inpainting.

Attack	Mask Preprocessing			Inpainting		
	Saliuitl RR/LPR	*Saliuitl*-direct RR/LPR	*Saliuitl*-mf RR/LPR	*Saliuitl* * RR/LPR	*Saliuitl*-z RR/LPR	*Saliuitl*-m RR/LPR
INRIA-1	**0.5909/0.0152**	0.5606/0.0303	0.5606 /0.0	0.6364/0.0	0.4848/0.0	0.5455/0.0
INRIA-2	0.3871/0.0	0.4194/0.0	**0.4516/0.0**	0.7258/0.0	0.4516/0.0	0.4677/0.0
INRIA-T	0.4737/0.0526	**0.5263/0.0702**	0.4737/0.0351	0.5439/0.0	0.3333/0.0351	0.4737/0.0
INRIA-MO	0.4749/0.0	**0.4860/0.0**	0.4749/0.0	0.6760/0.0	0.3464/0.0	0.3631/0.0
VOC-1	0.5404/0.0293	**0.5525/0.0343**	0.5232/0.0293	0.6455/0.0	0.4732/0.0293	0.4918/0.0264
VOC-2	0.5376/0.0125	**0.5408/0.0094**	0.5282/0.0094	0.6763/0.0	0.4334/0.0102	0.4843/0.0078
VOC-T	0.4244/0.0348	**0.4280/0.0376**	0.3951/0.0302	0.4904/0.0	0.3602/0.0321	0.3740/0.0275
VOC-MO	0.3955/0.0095	**0.4023/0.0078**	0.3821/0.0064	0.5811/0.0	0.2679/0.0085	0.3150/0.0081
ImageNet-1	**0.8869/0.0071**	0.8861/0.0067	0.8789/0.0067	0.9325/0.0001	0.8830/0.0093	0.9380/0.0095
ImageNet-2	**0.8535/0.0061**	0.8396/0.0052	0.8413/0.0064	0.9165/0.0001	0.8586/0.0096	0.8600/ 0.0097
ImageNet-4	**0.8436/0.0086**	0.8188/0.0074	0.8244/0.0091	0.9170/0.0001	0.8476/0.0095	0.8506/0.0096
ImageNet-T	0.5065/0.0612	**0.8424/0.2085**	0.6081/0.0491	0.9054/0.0003	0.5468/0.0721	0.5550/0.0706
CIFAR-1	0.9738/0.0008	**0.9907/0.0**	0.9837/0.0	0.9937/0.0004	0.9744/0.0	0.9746/0.0
CIFAR-2	0.9789/0.0006	**0.9872/0.0023**	0.9841/0.0008	0.9934/0.0003	0.9653/0.0011	0.9676/0.0015
CIFAR-4	0.9747/0.0	0.9811/0.0021	**0.9811/0.0**	0.9893/0.0003	0.9504/0.0001	0.9530/0.0003
CIFAR-T	0.8566/0.0	**0.9775/0.0035**	0.9689/0.0086	0.9275/0.0	0.9551/0.0501	0.9396/0.0380

Table 8. Recovery stage ablation. Best adversarial-clean nmAP tradeoffs for object detection. For single (1), double (2), triangular (T), and multi-object (MO) patch attack scenarios. Best mask preprocessing and inpainting methods per row are in bold and underlined, respectively. Note *Saliuitl* represents the default methods for both mask preprocessing and inpainting.

Attack	Mask Preprocessing			Inpainting		
	Saliuitl Adv./Clean	*Saliuitl*-direct Adv./Clean	*Saliuitl*-mf Adv./Clean	*Saliuitl* * Adv./Clean	*Saliuitl*-z Adv./Clean	*Saliuitl*-m Adv./Clean
INRIA-1	0.6897/0.9998	0.6866/0.9999	**7273/0.9999**	0.7213/1.0	0.6988/0.9963	0.6709/0.9984
INRIA-2	0.5998/1.0	**0.6707/0.9997**	0.6465/1.0	0.7191/1.0	0.6045/1.0	0.6879/0.9980
INRIA-T	**0.7034/0.9987**	0.6735/0.9980	0.6324/0.9986	0.6901/1.0	0.5731/1.0	0.6225/0.9939
INRIA-MO	**0.4737/0.9999**	0.4478/0.9999	0.4468/1.0	0.5505/1.0	0.3970/0.9773	0.4164/0.9999
VOC-1	0.5094/0.9950	0.5167/0.9941	**0.5188/0.9957**	0.5960/1.0	0.4439/0.9904	0.4809/0.9885
VOC-2	0.5088/0.9940	0.5094/0.9956	**0.5150/0.9973**	0.6196/1.0	0.4251/0.9889	0.4635/0.9943
VOC-T	0.5043/0.9942	0.5010/0.9931	**0.5172/0.9917**	0.5778/1.0	0.4645/0.9941	0.4872/0.9929
VOC-MO	0.3563/0.9877	0.3596/0.9907	**0.3809/0.9929**	0.4987/1.0	0.2570/0.9897	0.3246/0.9853

Impact of Mask Preprocessing and Inpainting

In our implementation, during the recovery stage *Saliuitl* uses DBSCAN results from the binary feature map corresponding to each threshold in $\mathcal{B}_R$ to determine potentially adversarial image regions (for an input $\mathbf{x}_i$ and its feature map $\mathbf{m}_i$); moreover, it uses biharmonic inpainting to produce $\hat{\mathbf{x}}_i$. Here we show results for alternative choices regarding these two components, namely mask preprocessing and inpainting. For any choice of mask preprocessing and inpainting methods in this section, the default DBSCAN approach with the proposed 1D CNN *AD* for attack detection is used in the detection stage. Once again we use the default saliency threshold sets for detection and recovery, that is, $\mathcal{B}_D = \mathcal{B}_R = \{\frac{x \cdot \max(\mathbf{m}_i)}{20}\}_{x=0}^{19}$.

Mask Preprocessing

We propose two alternatives for mask preprocessing. In *Saliuitl*-direct we skip mask preprocessing for all binary feature maps altogether, i.e., for each $\mathbf{m}_i$ and each threshold $\beta_r \in \mathcal{B}_R$, we inpaint all image regions corresponding to the neurons in the resulting binary feature map B_r. In *Saliuitl*-mf we apply uniform filtering on B_r and determine the inpainted regions using only neurons that remain after applying the uniform filter. We use sci-kit image's uniform filter implementation [37] (version 0.19.3) and set the kernel size to 2.

Object Detection In Table 7 (first eight rows) we show the best tradeoffs in terms of recovery and lost prediction rates achieved by *Saliuitl*, *Saliuitl*-direct, and *Saliuitl*-mf, on all datasets. The detailed tradeoffs are provided in Figures 27 and 28. We observe that in most cases *Saliuitl*-direct achieves the best tradeoffs, except for single and double patches on INRIA, where *Saliuitl* and *Saliuitl*-mf achieve the best performance, respectively. Despite their differences in performance, the three approaches show similar tradeoffs in the figures, distinct from the detection stage ablations where different methods seemed to find tradeoffs exclusive to themselves (e.g. *Saliuitl* for single patches on INRIA or *Saliuitl*-svm for quadruple patches on CIFAR-10); this suggests that the detection stage has an important role in determining what specific attacks are recoverable by *Saliuitl*. For fixed limits on the lost prediction rate, *Saliuitl* seems to

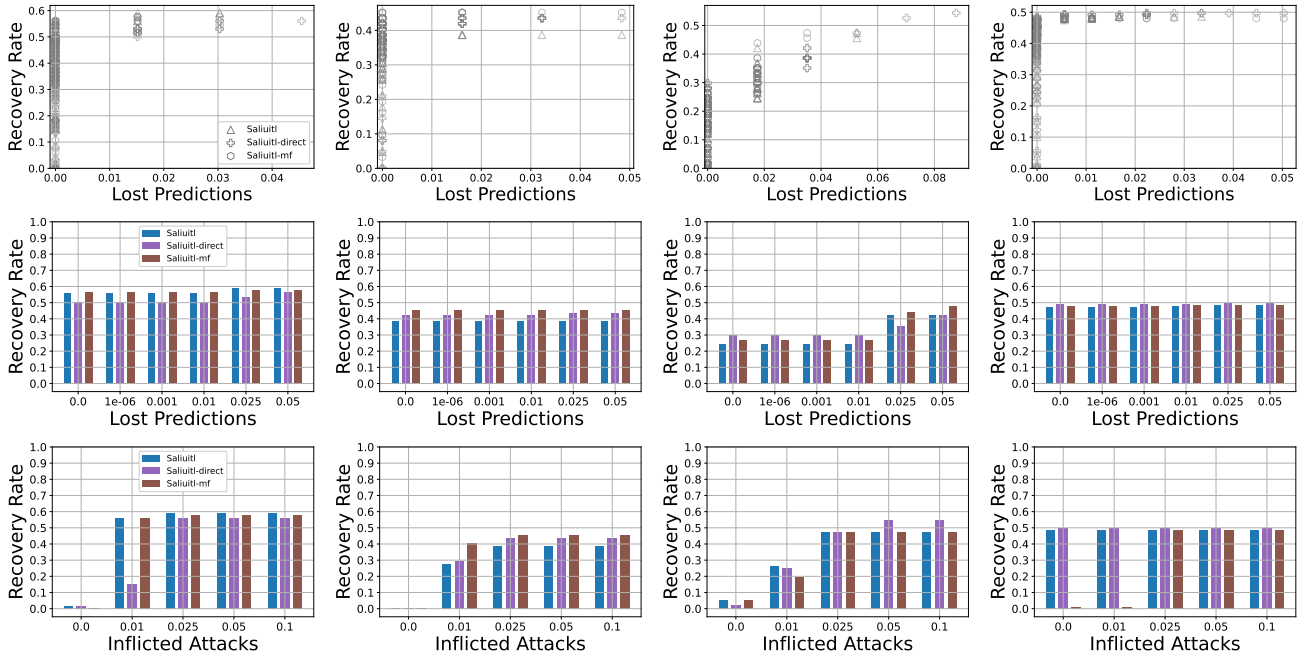


Figure 27. Mask preprocessing ablation. Recovery results for Object Detection (INRIA). From left to right: single, double, triangular, and multi-object patch attacks.

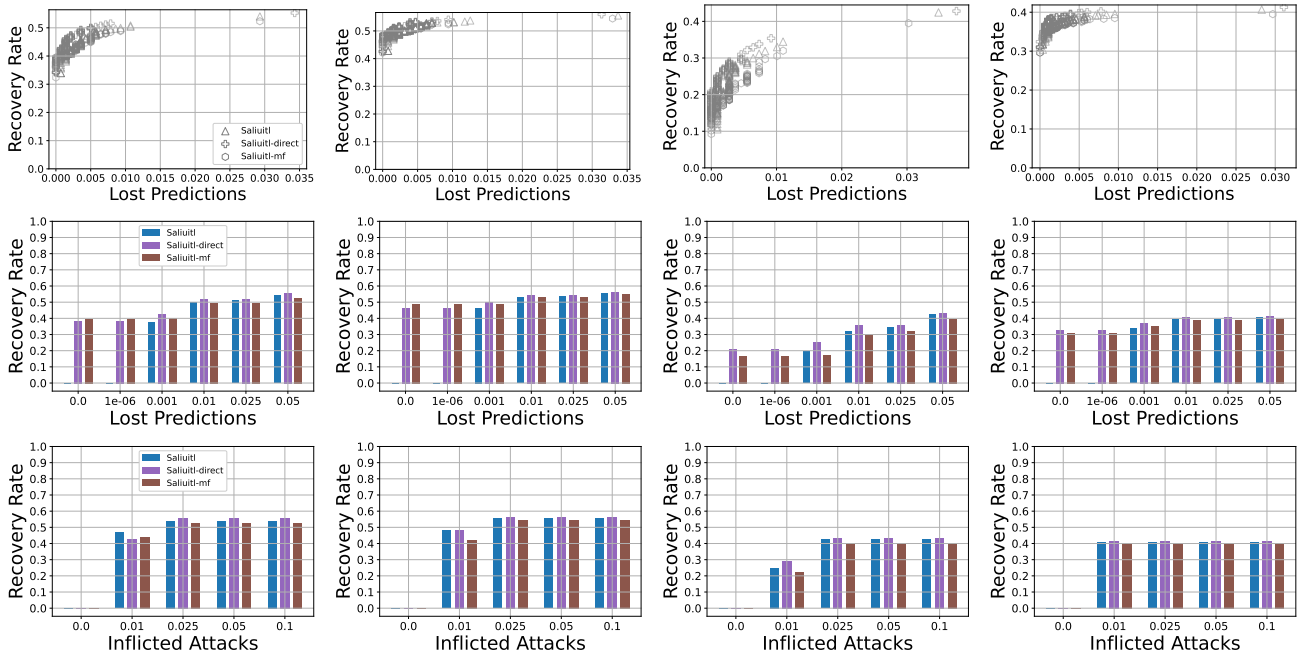


Figure 28. Mask preprocessing ablation. Recovery results for Object Detection (Pascal VOC). From left to right: single, double, triangular, and multi-object patch attacks.

perform worse than both *Saliuitl*-direct and *Saliuitl*-mf on Pascal VOC, but outperforms both of them for fixed limits on the inflicted attack rate on INRIA.

We further compare the different mask preprocessing methods in terms of clean and adversarial nmAP in Table 8 and Figures 29 and 30. In contrast with the results on recovery and lost prediction rates, *Saliuitl*-direct achieves the best tradeoff only for double patches on INRIA, while *Saliuitl* -

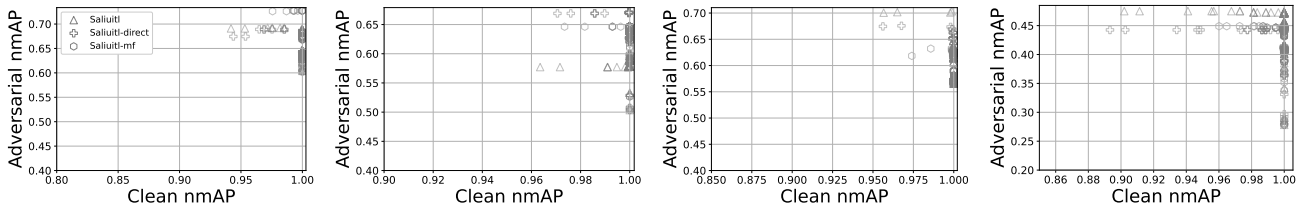


Figure 29. Mask preprocessing ablation. nmAP results for INRIA. From left to right: single, double, triangular, and multi-object patch attacks.

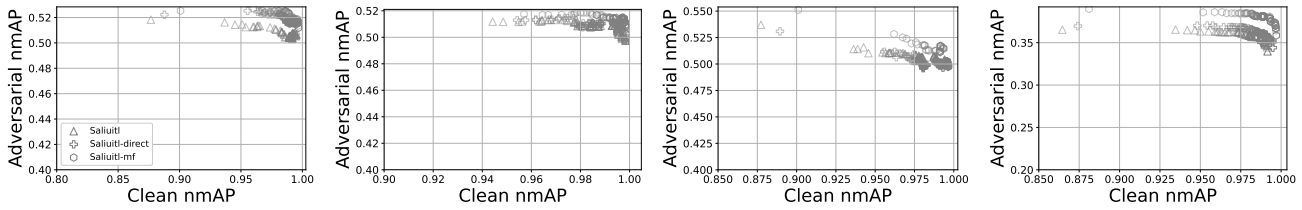


Figure 30. Mask preprocessing ablation. nmAP results for Pascal VOC. From left to right: single, double, triangular, and multi-object patch attacks.

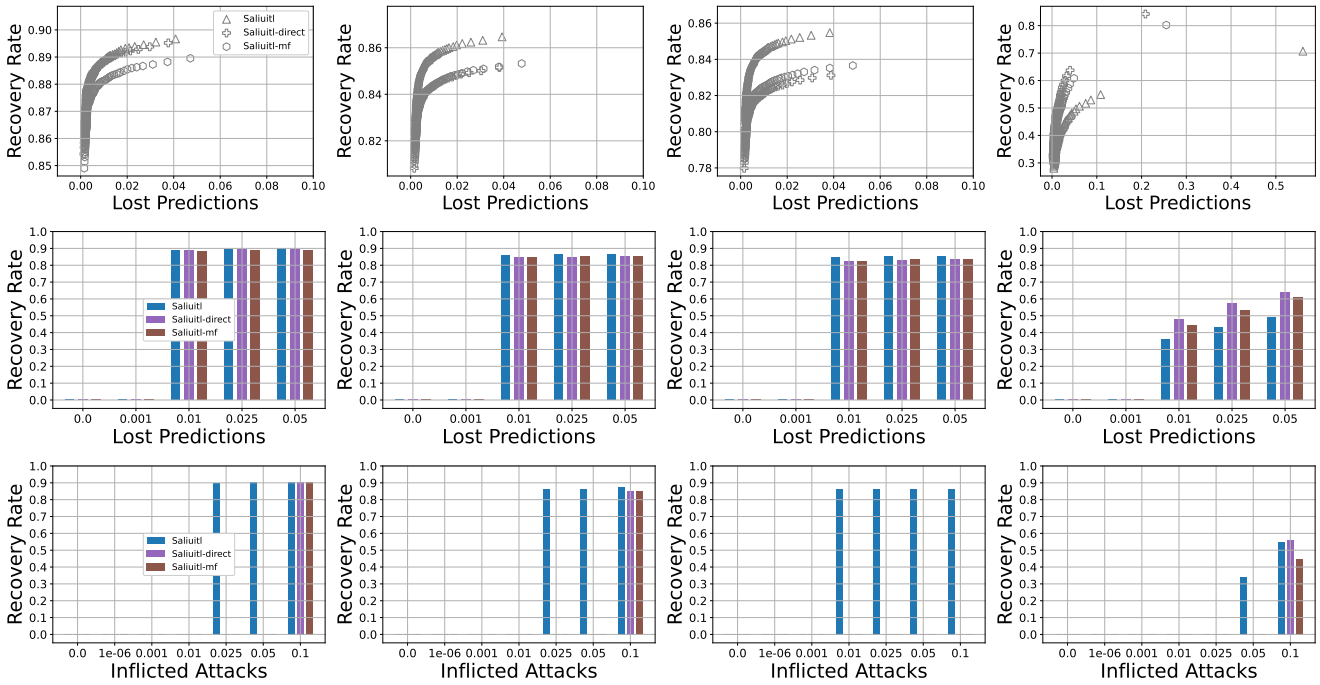


Figure 31. Mask preprocessing ablation. Recovery results for Image Classification (ImageNet). From left to right: single, double, quadruple, and triangular patch attacks.

mf achieves the best tradeoffs in most other scenarios, with the exception of triangular and multi-object patches, where the default *Saliuitl* is the best performing approach. These results suggest that *Saliuitl*-direct, by being more aggressive and considering that *any* neuron in B_r corresponds to a patch attack, is able to achieve a better tradeoff between recovery and lost prediction rates, but at the same time it may become more disruptive than *Saliuitl* and *Saliuitl*-mf in terms of clean and adversarial nmAP; this is similar to what was observed for *Object Seeker* in Section 4.2.

Image Classification The bottom half of Table 7 shows the best recovery-lost prediction rate tradeoffs, and the detailed tradeoffs are shown in Figures 31, and 32. We note that *Saliuitl* is the best performing approach for most Ima-

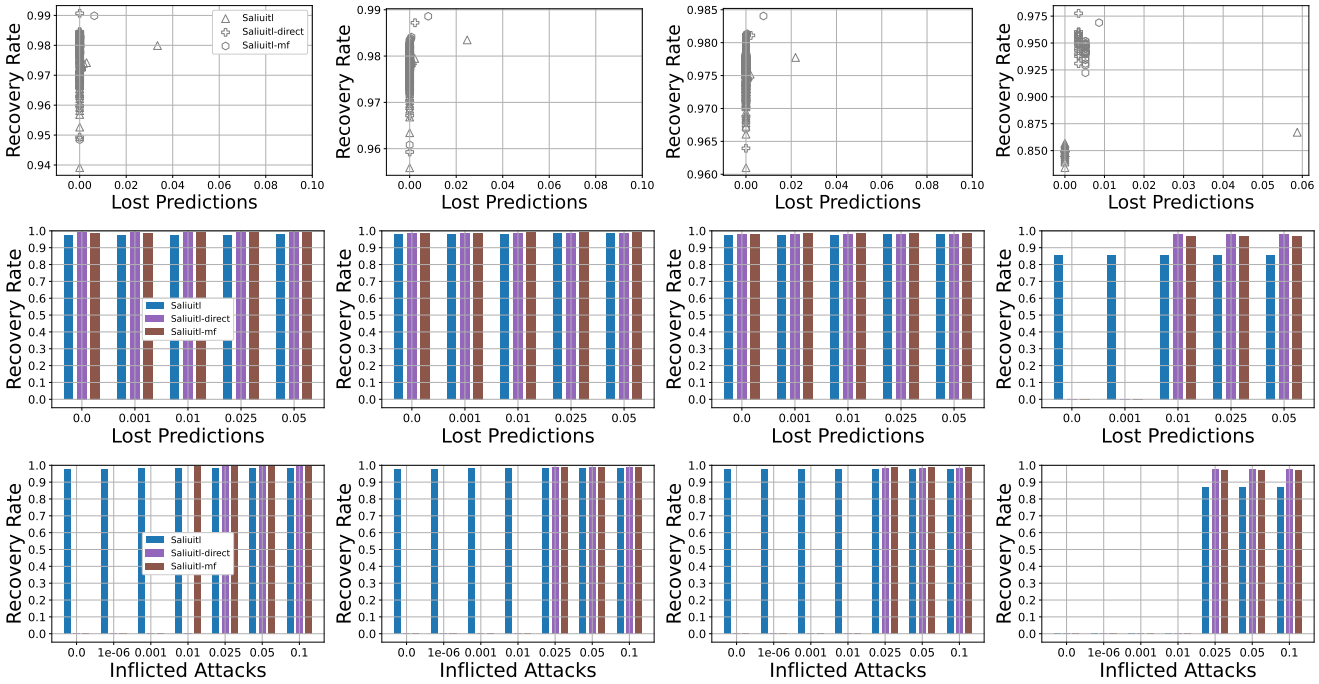


Figure 32. Mask preprocessing ablation. Recovery results for Image Classification (CIFAR-10). From left to right: single, double, quadruple, and triangular patch attacks.

geNet scenarios, except for triangular patches, where *Saliuitl*-direct is better. *Saliuitl*-direct is also the best method for most CIFAR-10 scenarios, except for quadruple patches, where *Saliuitl*-mf is better. All three approaches have similar performances in image classification scenarios. At fixed thresholds on lost predictions, the three approaches perform similarly in most scenarios, although *Saliuitl* has a significant advantage for triangular patches on CIFAR-10, interestingly, except for this very scenario, *Saliuitl* has a clear superiority for a fixed threshold on inflicted attacks. The results are also congruent with the results for object detection, in that *Saliuitl*-direct seems to be the most disruptive approach; in some cases, this pays off and *Saliuitl*-direct is able to find a better tradeoff than both *Saliuitl* and *Saliuitl*-mf, but this often comes at a higher rate of lost predictions, the much higher recovery *and* lost prediction rates for *Saliuitl*-direct for triangular patches on ImageNet in Table 7 illustrate this most clearly.

Inpainting

In Section 4, we mention that *Saliuitl* employs biharmonic inpainting for our evaluation. In particular, we use sci-kit image's implementation [37] (version 0.19.3) which does not require setting any hyper-parameters. To illustrate the importance of inpainting in *Saliuitl*'s recovery stage we consider three alternative inpainting approaches. *Saliuitl* * is an ideal version of *Saliuitl* where inpainting is performed by replacing "inpainted" pixels with their corresponding ground truth values from the original clean image; while this is not a realistic baseline, it provides an insightful reference for the limits on the performance of *Saliuitl* for any inpainting approach we might consider. *Saliuitl*-z is a simpler version of *Saliuitl*, where inpainting is performed by replacing the relevant pixels with zeros instead; finally, *Saliuitl*-m is another simple approach where the relevant pixels are replaced not with zeros but with the mean pixel value from pixels *outside* the inpainted region.

Object Detection The first eight rows of Table 7 show the best tradeoffs achieved by each approach in terms of recovery and lost prediction rates; the detailed tradeoffs are shown in Figures 33 and 34. As one might expect, *Saliuitl* * achieves the best performance in all scenarios. Interestingly, the default *Saliuitl* is second best in most scenarios except for double patches on INRIA, where *Saliuitl* has the worst performance, and triangular patches on INRIA, where *Saliuitl*-m performs better. The figures also show that when imposing fixed thresholds on lost predictions and inflicted attacks *Saliuitl* * has a clear advantage, with *Saliuitl* being second best in most scenarios, except for double and triangular patches on INRIA. The results suggest that more elaborate inpainting approaches usually lead to an increase in recovery performance, but not in all scenarios. In particular we find that simpler schemes might be preferable in scenarios where it is relatively straightforward to maintain a low lost prediction rate; for example, in the double patch

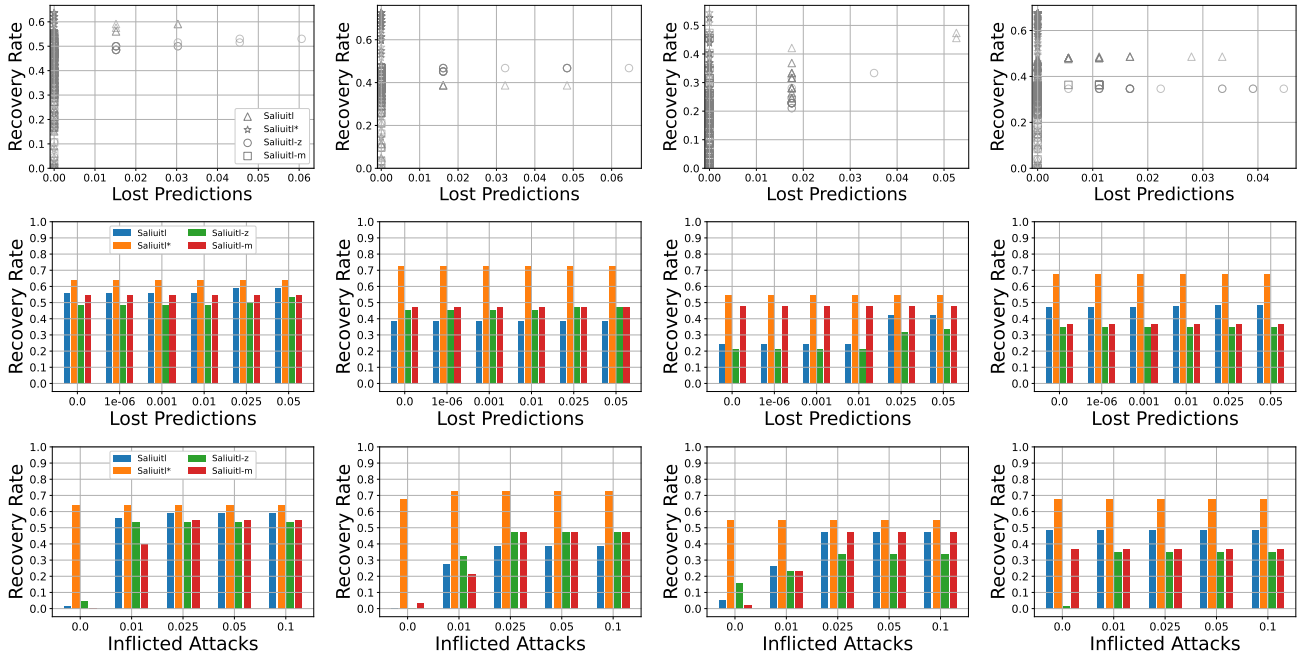


Figure 33. Inpainting ablation. Recovery results for Object Detection (INRIA). From left to right: single, double, triangular, and multi-object patch attacks.

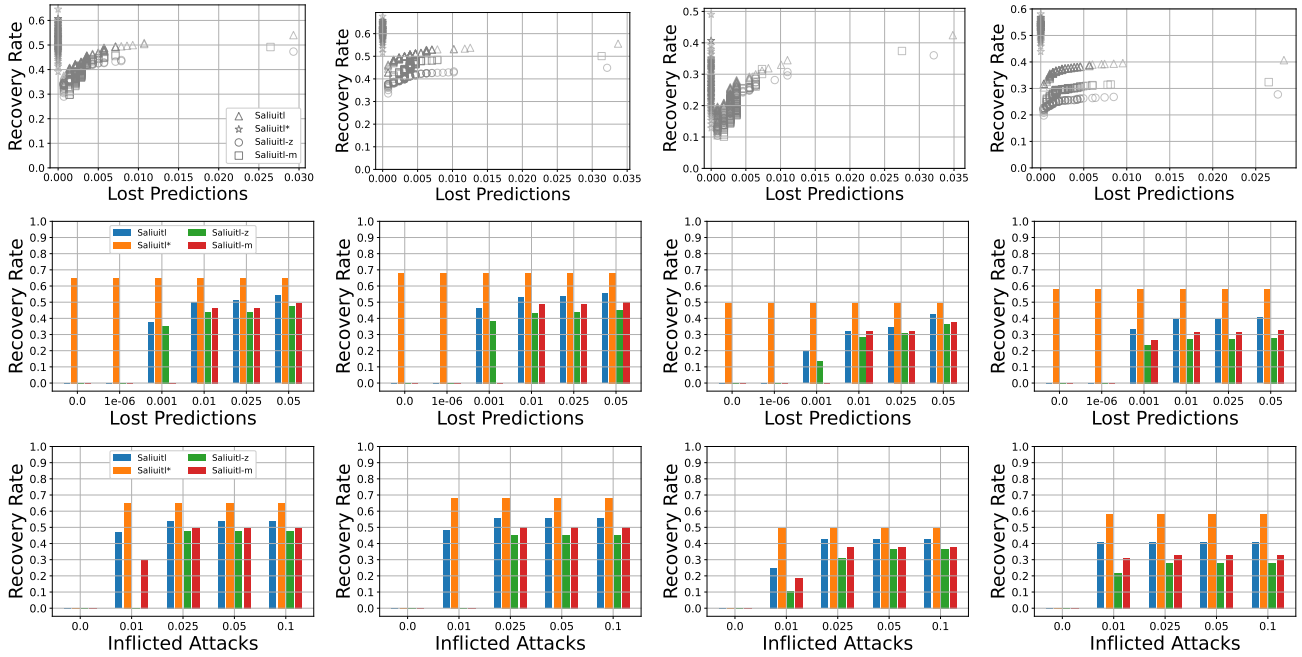


Figure 34. Inpainting ablation. Recovery results for Object Detection (Pascal VOC). From left to right: single, double, triangular, and multi-object patch attacks.

scenario on INRIA both *Saliuitl*-z and *Saliuitl*-m are able to match *Saliuitl*'s low lost prediction rate, but are capable of achieving a higher recovery rate (see Table 7 and Figure 33). Crucially, the performance of *Saliuitl* * shows that even with perfect inpainting, not all attacks are recoverable by *Saliuitl*.

We also show the best tradeoffs in terms of clean and adversarial nmAP in Table 8 and Figures 35 and 36. While

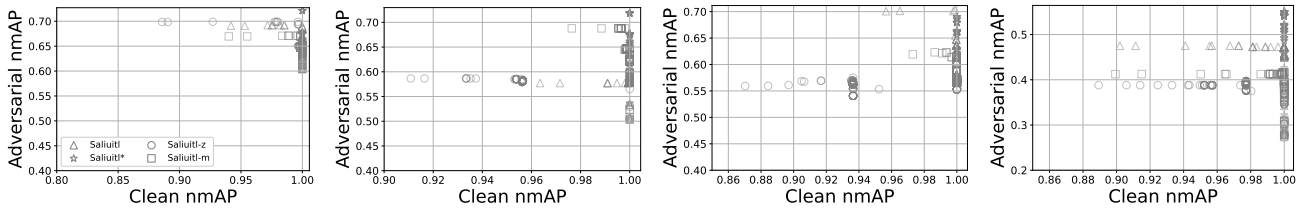


Figure 35. Inpainting ablation. nmAP results for INRIA. From left to right: single, double, triangular, and multi-object patch attacks.

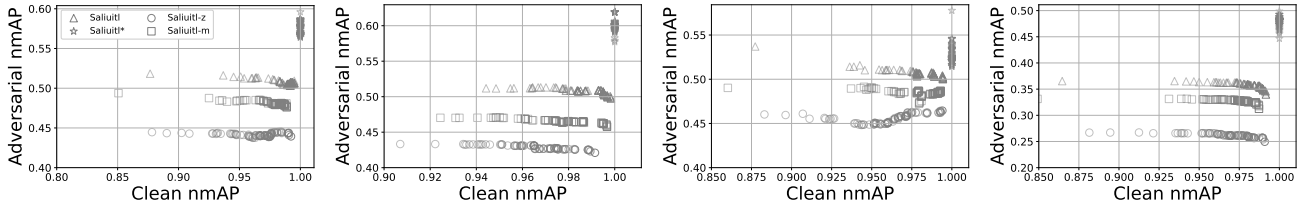


Figure 36. Inpainting ablation. nmAP results for Pascal VOC. From left to right: single, double, triangular, and multi-object patch attacks.

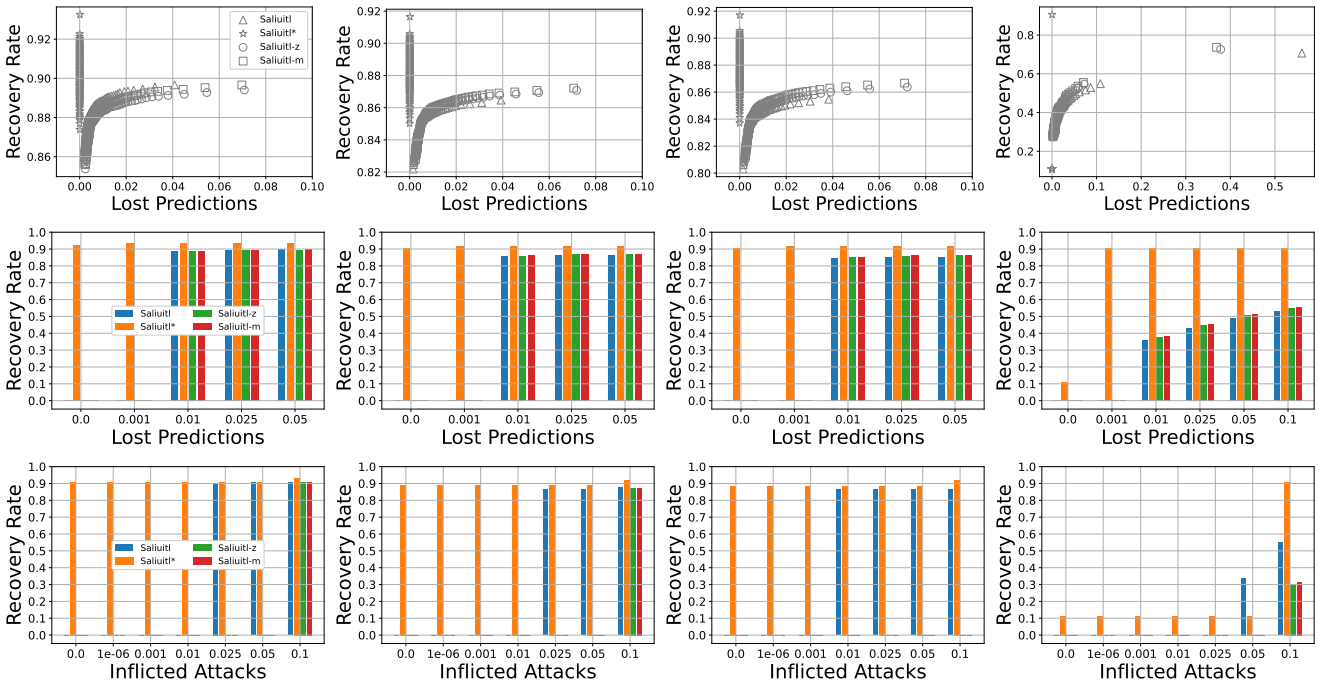


Figure 37. Inpainting ablation. Recovery results for Image Classification (ImageNet). From left to right: single, double, quadruple, and triangular patch attacks.

Saliuitl * retains its superiority in terms of nmAP for most scenarios, intriguingly *Saliuitl* has a better tradeoff for triangular patch attacks on INRIA. For most other cases *Saliuitl* is only second to *Saliuitl* *, except for double patches on INRIA, where it is once again the worst performing method. These results further support the notion that a better inpainting approach usually leads to a better performance, and that this need not be the case for all scenarios.

Image Classification The best achieved tradeoffs for the distinct inpainting approaches are shown in the last eight rows of Table 7, and the detailed tradeoffs are shown in Figures 37 and 38. Once again we observe that *Saliuitl* * performs best as one might expect. Interestingly, while *Saliuitl* outperforms *Saliuitl*-z and *Saliuitl*-m on CIFAR-10, these simpler inpainting approaches outperform *Saliuitl* for most ImageNet scenarios, except for single patch attacks. For fixed thresholds on the lost prediction rate, *Saliuitl* * performs best, while *Saliuitl* performs either similarly or bet-

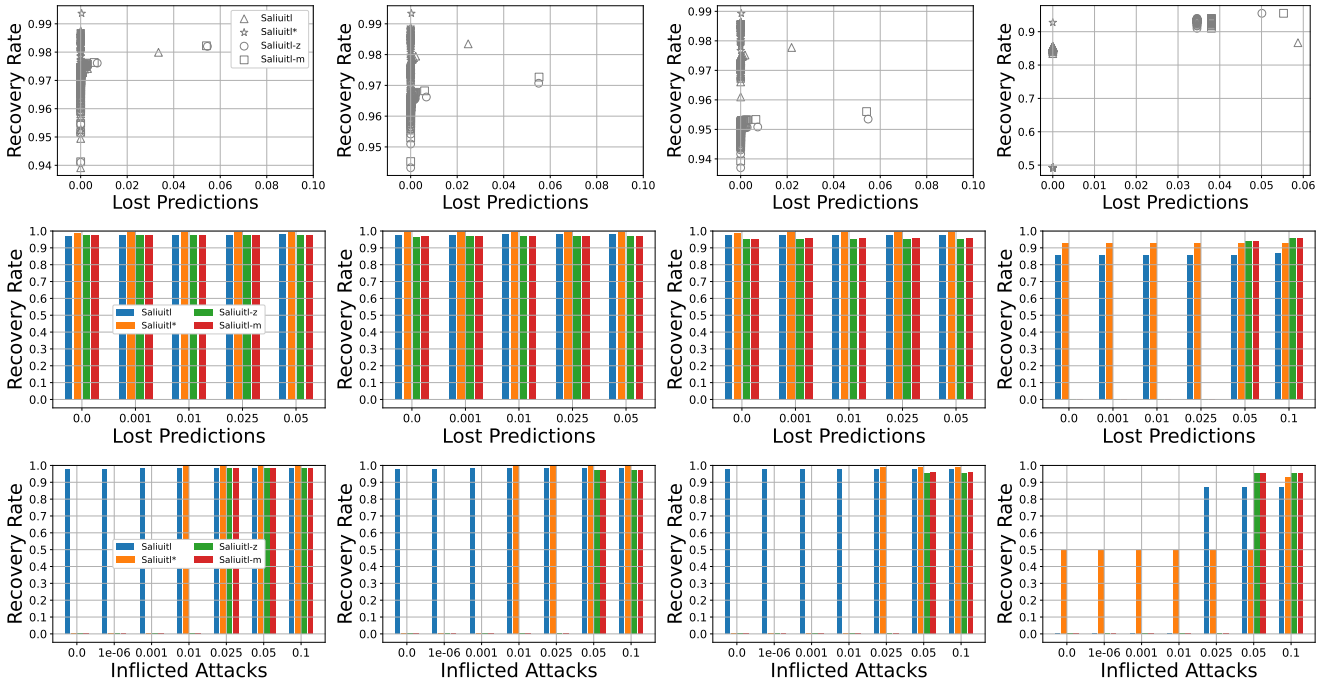
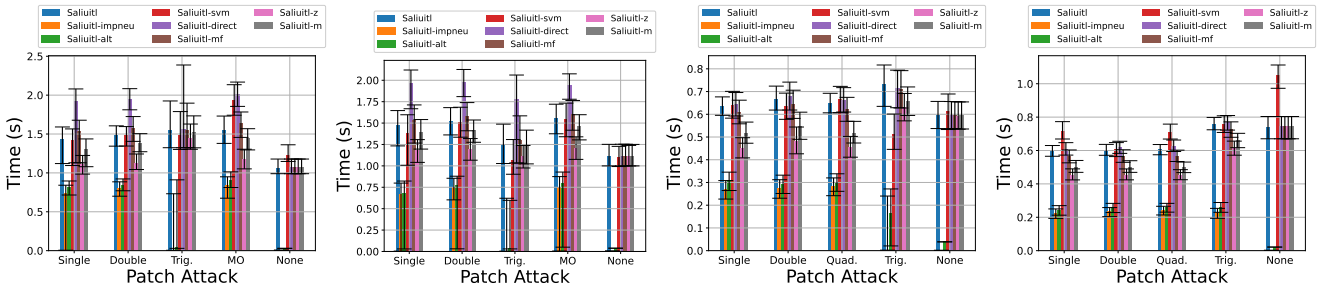


Figure 38. Inpainting ablation. Recovery results for Image Classification (CIFAR-10). From left to right: single, double, quadruple, and triangular patch attacks.

ter than *Saliuitl*-z and *Saliuitl*-m in all other scenarios. Intriguingly, for fixed thresholds on the inflicted attack rate, *Saliuitl* can even outperform *Saliuitl* * for all rectangular patch scenarios on CIFAR-10, while *Saliuitl* * is the best performing approach for all other scenarios. These results show that inpainting also has a relevant impact on recovery performance for image classification; most notably, a more complex and precise inpainting approach can result in remarkable improvements in terms of lost predictions and inflicted attacks. At the same time, these results confirm that a better inpainting approach does not always result in a better performance.

Balancing *Saliuitl*'s recovery performance and computational cost from the perspective of inpainting may be an interesting avenue to explore in future work. As shown in Figure 4(a), our default biharmonic inpainting can attain a reasonable computational cost, hence it could be valuable to explore more complex inpainting approaches. On the other hand, comparing Tables 7 and 8 to Tables 1 and 2 shows that simpler approaches like *Saliuitl*-m and *Saliuitl*-z are also competitive and even superior to most existing approaches, therefore it would also be relevant to explore the possibility of improving the performance of *Saliuitl*-m or *Saliuitl*-z (e.g., with a larger size for the ensemble $\mathcal{B}_R$).



(a) INRIA (b) VOC (c) ImageNet (d) CIFAR-10

Figure 39. Comparison of *Saliuitl*'s computational cost for the detection and recovery stage ablations, on all datasets. Error bars represent the first and third quartiles across the dataset.

Trading off computational cost and performance

In our analyses of Figure 4(a) in Section 4.2, we argue that *Saliuitl* can attain a low computational cost while maintaining an adequate attack recovery performance, based on the comparison to the baselines' at their default parameters in terms of execution time.

Now, to show explicitly the recovery performances that correspond to our reported execution times for *Saliuitl*, consider our results in Figure 12, which shows extensive combinations of $|\mathcal{B}_D|$ and $|\mathcal{B}_R|$, including the cases where $\mathcal{B}_D = \mathcal{B}_R$ on which we based our computational cost comparisons. In Figure 12 we display the recovery performance achieved by each choice of $|\mathcal{B}_R|$ across all datasets and attack models, for different choices of $|\mathcal{B}_D|$. Recall that to relate Figure 12 to Figure 4(a), one must focus on those cases where $|\mathcal{B}_D| = |\mathcal{B}_R| = |\mathcal{B}_B|$. Hence we can see that, e.g., when we select $|\mathcal{B}_B| = 10$ in the INRIA dataset, despite *Saliuitl*'s low computational cost, comparable to *Jedi* and not too far above that of *Themis* (see Figure 4(a)), the recovery performance attained (see Figure 12(b), top row) is superior to most baselines in all scenarios, and only below that of *Object Seeker* (see Table 1). Moreover, if recovery performance is more desirable than the remarkable efficiency gains, reverting to the default $|\mathcal{B}_D| = |\mathcal{B}_R| = 20$ can achieve a notable performance increase while still maintaining the upper hand on *Object Seeker* in terms of efficiency, with a computational cost that is twice as low (see Figure 4(a), and either Figure 12(c), top row, or Table 1).

We may also consider the ablation experiments regarding ensemble attributes, detection schemes, mask preprocessing, and inpainting. Figure 39 shows the comparison between the default DBSCAN-based instance of *Saliuitl* used for evaluation, along with all the different ablations for these components, whose performance in terms of recovery and lost prediction rates is summarized in Tables 5 and 7. Figure 39 shows the performance of these alternative implementations of *Saliuitl* using the default ensemble size of 20 for both $\mathcal{B}_D$ and $\mathcal{B}_R$, and setting $\alpha^* = 0.5$. We observe that simpler attribute extraction approaches can yield significant performance gains (as seen for *Saliuitl*-impneu and *Saliuitl*-alt), and the same can be said for simpler inpainting approaches (*Saliuitl*-z and *Saliuitl*-m). This allows *Saliuitl* to, for example, maintain superiority over all other baselines in terms of nmAP at much lower computational cost, by using different ensemble attributes and still outperform the best settings of existing defenses: comparing Table 6 with Table 2, it is evident that *Saliuitl*-alt outperforms all existing defenses, and comparing Figure 4(a) with Figure 39 one can see that the computational cost of *Saliuitl*-alt is on par with *Themis* and even below (e.g., notice their cost for clean images). Furthermore, one may also apply a simpler inpainting approach as in *Saliuitl*-z and *Saliuitl*-m, or even replace the entire recovery stage with existing defenses with negligible cost such as *FNS* or *NutNet*. We consider such explorations to tradeoff computational efficiency and performance to be within the scope of future work.

Additional Attack and Defense Baselines

For completeness, we present additional results considering defenses and attacks not included in our main evaluations from Section 4.2.

Defenses

Segment and Complete (*SAC*) is a recovery approach that performs segmentation to identify adversarial regions and then shape completion to obtain a refined mask for the input image. A U-Net architecture is trained to segment ground truth adversarial regions in patched images, self-adversarial training is also used to robustify the segmenter by using patches that are trained to bypass it. The segmenter assigns adversarial probability scores to each pixel in the input image, thus the mask from the segmenter consists of all pixels with a score above 0.5. For shape completion, the ground truth patch size or a set of feasible sizes is required to produce candidate masks that are aggregated into a final refined mask, which is finally applied to the input

image. Total Variation based image Resurfacing (*TVR*) is a recovery method designed to handle multiple patch scenarios [33]. It considers each input color channel separately and for a fixed block size of $k \times k$ pixels, it computes block-wise total variation using a $k \times k$ sliding window. Outlier blocks in the image are those with a total variation above $Q_3 + 1.5 \cdot (Q_3 - Q_1)$ where Q_1 and Q_3 are the first and third quartiles of the total variation values of all blocks in the image. The outlier blocks from each color channel are combined to produce a mask for the input image. Masked regions are inpainted by feeding the masked input to a generative adversarial network trained to reconstruct images.

SAC is a common point of comparison in the papers for most of our baselines [26]. We favored said baselines over SAC for evaluation in Section 4.2 because they are more recent and present results showing they generally outperform *SAC*. *TVR* is interesting due to its explicit treatment of multiple patch scenarios, however we were unaware of it when we performed our evaluations. Moreover our baselines already include more recent empirical approaches that in principle handle multiple patches as well.

For evaluation we vary the parameters of these two baselines as follows: for *SAC* we vary the thresholds on the segmenter's output probability and the threshold used for the shape completion mask in the range $[0.1, 0.9]$ in increments of 0.1. Interestingly, while the threshold for shape completion can be found in the official code for *SAC* (and set by default at 0.5), it was not mentioned in their paper [26]. For *TVR*, the feasible range of the block size k is determined by the image size, since the block size must be a factor of the image size. Thus we use block sizes from 13 to 208 for INRIA/VOC, from 7 to 112 for ImageNet, and from 6 to 96 for CIFAR-10, such that each increasing block size doubles the previous size. Due to the high computational cost of *TVR*, we only apply it on the full dataset for INRIA scenarios. For all other datasets, we evaluate only on 1000 images.

Table 9 shows the results for *SAC* and *TVR* in terms of recovery rate and lost prediction tradeoffs, we also include *Saliuitl* for reference. While *SAC* performs quite well for single patches, it clearly struggles to adapt to triangular and multiple patch scenarios. Interestingly, we found that while the original paper makes no mention of multiple patch attacks, the official implementation seems to contain some functionalities for that purpose, but they require *SAC* to know the number of patches beforehand, thus using them would lead to an unfair evaluation, moreover, for the multi-object (MO) scenarios, the number of patches depends on the ground truth objects in the scene, which emphasizes the flaws of assuming access to this information. We find that *TVR* performs quite poorly for attacks on object detection, and despite its explicit treatment of multiple patches, it performs even worse for multiple and triangular patches. For image classification *TVR* performs reasonably well and is able to retain a decent performance for multiple patches, although it experiences a very large drop in performance for triangular patches. Our results suggest that *TVR* can indeed handle multiple patches in certain scenarios, but our results in Table 1 show that other baselines are more resilient to an increase in the number of patches; we also note *TVR* seems to be mostly limited to scenarios where the victim model task is image classification. Most importantly, we point out that *Saliuitl* thoroughly dominates both *SAC* and *TVR* for all scenarios.

Table 9. Best Recovery Rate (*RR*) - Lost Prediction Rate (*LPR*) trade-offs for single (1), double (2), quadruple (4), triangular (T), and multi-object (MO) patch attack scenarios. The best method per row is in bold, according to *RR minus LPR*.

Attack	***Saliuitl*** RR/LPR	**SAC** RR/LPR	**TVR** RR/LPR
INRIA-1	**0.5909/0.0152**	0.3030/0.0	0.1970/0.0909
INRIA-2	**0.3871/0.0**	0.1129/0.0	0.1290/0.1129
INRIA-T	**0.4737/0.0526**	0.0526/0.0	0.0877/0.1228
INRIA-MO	**0.4749/0.0**	0.1229/0.0	0.0838/0.0838
VOC-1	**0.5404/0.0293**	0.3102/0.0071	0.1964/0.1236
VOC-2	**0.5376/0.0125**	0.1426/0.0063	0.1299/0.1339
VOC-T	**0.4244/0.0348**	0.0174/0.0816	0.0188/0.1408
VOC-MO	**0.3955/0.0095**	0.0813/0.0046	0.0654/0.1166
ImageNet-1	**0.8869/0.0071**	0.8667/0.0013	0.9150/0.0507
ImageNet-2	**0.8535/0.0061**	0.1812/0.0013	0.8310/0.0452
ImageNet-4	**0.8436/0.0086**	0.1429/0.0012	0.8260/0.1928
ImageNet-T	**0.5065/0.0612**	0.3695/0.0033	0.3421/0.1842
CIFAR-1	**0.9738/0.0008**	0.9174/0.0002	0.9560/0.0549
CIFAR-2	**0.9789/0.0006**	0.3892/0.0002	0.9500/0.0462
CIFAR-4	**0.9747/0.0**	0.2258/0.0001	0.9380/0.0705
CIFAR-T	**0.8566/0.0**	0.4801/0.0017	0.6290/0.1613

Table 10. Best Recovery Rate (*RR*) - Lost Prediction Rate (*LPR*) trade-offs for INRIA/VOC single patch attacks using different patches. Best performances in bold, according to *RR minus LPR*.

Attack	***Saliuitl*** (*Princess*) RR/LPR	***Saliuitl*** (*TSEAv3*) RR/LPR	***Saliuitl*** (*Dogv2*) RR/LPR
INRIA-1	0.5909/0.0152	**0.6471/0.0**	0.5976/0.0122
VOC-1	0.5404/0.0293	**0.5504/0.0025**	0.5157/0.0264

Attacks

There is a considerable amount of patch attacks that one could consider for evaluation. In our description of the patch attack models we justify our choices, and it is also important to note that our adaptive attack scenario presents a patch that is more challenging than any we could choose from the wide collection of publicly available patches in the literature. To further demonstrate *Saliuitl*'s effectiveness for different types of attacks, we present results on two more patch attacks for object detection, the naturalistic *Dogv2* patch from [20] and the *TSEAv3* patch from [21]. In Table 10 we show the recovery rate and lost prediction tradeoffs achieved by *Saliuitl* for single patch scenarios using these new patches, and for reference we include the results we obtained in our evaluation using the *Princess* patch, note

no further training is performed. *Saliuitl* is able to retain its performance for these new patches, and in fact performs better than it does for the *Princess* patch in most cases, confirming that we chose a reasonably challenging patch for evaluation; the only exception to this trend is *Saliuitl*'s slightly lower recovery rate for VOC with the *Dogv2* patch.