Appendix

Algorithm 2 Meta-algorithm for Anomaly Detection

Inputs: Trained DNN $\mathbf{F}(\cdot)$, Dataset $\mathcal{D}$, Test input $\mathbf{x}$, FPR α or detection threshold τ.
Output: Detector decision – normal 0 or anomaly 1.

Preprocessing:
Calculate the detection threshold τ (if not specified).
Class prediction of $\mathbf{x}$: $\hat{c} = \arg\max_{c\in[m]} F_c(\mathbf{x})$.
Layer representations of $\mathbf{x}$: $\mathbf{x}^{(\ell)} = \mathbf{f}_\ell(\mathbf{x})$, $\forall \ell \in \mathcal{L}$.
Data subsets: $\widehat{\mathcal{D}}_a(\ell, \hat{c})$ and $\mathcal{D}_a(\ell, c)$, $\forall\, \ell \in \mathcal{L},\ c \in [m]$.

I. Test statistics:
  for each layer $\ell \in \mathcal{L}$:
    Calculate $t^{(\ell)}_{p\,|\,\hat{c}} = T(\mathbf{x}^{(\ell)}, \hat{c}, \widehat{\mathcal{D}}_a(\ell, \hat{c}))$.
    Calculate $t^{(\ell)}_{s\,|\,c} = T(\mathbf{x}^{(\ell)}, c, \mathcal{D}_a(\ell, c)),\ \forall c \in [m]$.
  Compile the vectors: $\mathbf{t}_{p\,|\,\hat{c}}, \mathbf{t}_{s\,|\,1}, \cdots, \mathbf{t}_{s\,|\,m}$.

II. Normalizing transformations:
  if multivariate normalization:
    Calculate $q(\mathbf{t}_{p\,|\,\hat{c}}), q(\mathbf{t}_{s\,|\,1}), \cdots, q(\mathbf{t}_{s\,|\,m})$.
  else
    for each layer $\ell \in \mathcal{L}$:
      Calculate $q(t^{(\ell)}_{p\,|\,\hat{c}}), q(t^{(\ell)}_{s\,|\,1}), \cdots, q(t^{(\ell)}_{s\,|\,m})$.
    for each layer pair $(\ell_1, \ell_2) \in \mathcal{L}^2$: **[optional]**
      Calculate $q(t^{(\ell_1)}_{p\,|\,\hat{c}}, t^{(\ell_2)}_{p\,|\,\hat{c}})$.
      Calculate $q(t^{(\ell_1)}_{s\,|\,1}, t^{(\ell_2)}_{s\,|\,1}), \cdots, q(t^{(\ell_1)}_{s\,|\,m}, t^{(\ell_2)}_{s\,|\,m})$.

III. Layerwise aggregation and scoring:
  if multivariate normalization:
    Set $q_{\text{agg}}(\mathbf{t}_{p\,|\,\hat{c}}) = q(\mathbf{t}_{p\,|\,\hat{c}})$.
    Set $q_{\text{agg}}(\mathbf{t}_{s\,|\,c}) = q(\mathbf{t}_{s\,|\,c}),\ \forall c \in [m]$.
  else
    Create the sets $Q_{p\,|\,\hat{c}}$ and $Q_{s\,|\,c},\ \forall c \in [m]$.
    Calculate $q_{\text{agg}}(\mathbf{t}_{p\,|\,\hat{c}}) = r(Q_{p\,|\,\hat{c}})$.
    Calculate $q_{\text{agg}}(\mathbf{t}_{s\,|\,c}) = r(Q_{s\,|\,c}),\ \forall c \in [m]$.
  Calculate the final score:
  $S(q_{\text{agg}}(\mathbf{t}_{p\,|\,\hat{c}}), q_{\text{agg}}(\mathbf{t}_{s\,|\,1}), \cdots, q_{\text{agg}}(\mathbf{t}_{s\,|\,m}), \hat{c})$.

IV. Detection decision:
  if $S(q_{\text{agg}}(\mathbf{t}_{p\,|\,\hat{c}}), q_{\text{agg}}(\mathbf{t}_{s\,|\,1}), \cdots, q_{\text{agg}}(\mathbf{t}_{s\,|\,m}), \hat{c}) \geq \tau$:
    Return anomaly (1).
  else
    Return normal (0).

The appendices are organized as follows.

- Appendix A provides a background on adversarial attacks and defenses, and discusses prior works in more detail.
- Appendix B describes the harmonic mean p-value (HMP) method for aggregating p-values.
- Appendix C discusses details of the adaptive attack method that were left out of § 5.
- Appendix D discusses the implementation details such as computing platform, datasets, DNN architectures, experimental setup, and the method implementation details, hyperparameters etc.
- Appendix E discusses additional experiments that we performed including ablation studies, performance on attack transfer and attacks of varying strength, and a comparison of running times.

A. Background and Related Works

We provide a background on adversarial attacks and defense methods, followed by a detailed discussion of related works.

A.1. Adversarial Attacks

Adversarial attacks may be broadly classified into training-time or data poisoning attacks, and test-time or evasion attacks (Biggio & Roli, 2018). Data poisoning attacks focus on tampering with the training set of a learning algorithm by introducing malicious input patterns that steer the learning algorithm to a sub-optimal solution, causing degradation in performance (Biggio et al., 2012; Muñoz-González et al., 2017; Steinhardt et al., 2017). Evasion attacks focus on tampering with test inputs to an already-trained ML model such that the model predicts incorrectly on them. In both cases, an adversary aims to modify the inputs in such a way that the changes are not easily perceived by a human or flagged by a detector. For example, a test image of the digit 1 may be modified by introducing minimal perturbations to the pixels such that a classifier is fooled into classifying it as the digit 7, while the image still looks like the digit 1 to a human. In this work, we focus on detecting test-time evasion attacks.

Given a test input $\mathbf{x} \in \mathcal{X}$ from class c, adversarial attack methods aim to create a minimally-perturbed input $\mathbf{x}' = \mathbf{x} + \boldsymbol{\delta}$ that is mis-classified by the classifier either into a specific class (targeted attack) or into any class other than c (untargeted attack). This is formulated as an optimization problem, which in its most general form looks like

$$\begin{aligned} &\min_{\boldsymbol{\delta}} \ \|\boldsymbol{\delta}\|_p \quad \text{s.t.} \\ &\mathbf{x} + \boldsymbol{\delta} \ \in \ \mathcal{X} \\ &\widehat{C}(\mathbf{x} + \boldsymbol{\delta}) \ = \ c' \quad \text{(targeted)} \\ \text{or } &\widehat{C}(\mathbf{x} + \boldsymbol{\delta}) \ \neq \ c \quad \text{(untargeted)} \end{aligned}$$

A number of adversarial attack methods have been proposed based on this general formulation (Szegedy et al., 2014; Goodfellow et al., 2015; Madry et al., 2018; Carlini & Wagner, 2017b; Moosavi-Dezfooli et al., 2016; Papernot et al., 2016; Kurakin et al., 2017; Moosavi-Dezfooli et al., 2017). Some of the well-known methods for generating adversarial data include the fast gradient sign method (FGSM) (Goodfellow et al., 2015), projected gradient descent (PGD) attack (Madry et al., 2018), Carlini-Wagner attack (Carlini & Wagner, 2017b), and DeepFool attack (Moosavi-Dezfooli et al., 2016). These attack methods can be categorized into *white-box* or *black-box* depending on the extent of their knowledge about the classifier's structure, parameters, loss function, and learning algorithm. Most of the commonly

Table 1: Notations and Definitions

Term	Description
$[m] := \{1, \cdots, m\}$	Set of classes.
$\mathcal{L} = \{0, \cdots, L\}$, $\mathcal{L}^2 = \{(\ell_1, \ell_2) \in \mathcal{L} \times \mathcal{L} : \ell_2 > \ell_1\}$	Set of layers and distinct layer pairs.
$\|\cdot\|_p$	ℓ_p norm of a vector.
$d(\mathbf{x}, \mathbf{y})$	Distance between a pair of vectors; cosine distance unless specified otherwise.
$\mathbb{1}[\cdot]$	Indicator function mapping an input condition to 1 if true and 0 if false.
$h_\sigma(\mathbf{x}, \mathbf{y}) = e^{-\frac{1}{\sigma^2} d(\mathbf{x}, \mathbf{y})^2}$	Gaussian or RBF kernel.
$\mathbf{F}(\mathbf{x}) = [F_1(\mathbf{x}), \cdots, F_m(\mathbf{x})]$, $\mathbf{F} : \mathcal{X} \mapsto \Delta_m$	Input-output mapping learned by the DNN classifier.
$\Delta_m = \{(p_1, \cdots, p_m) \in [0,1]^m : \sum_i p_i = 1\}$	Space of output probabilities for the m classes.
$\widehat{C}(\mathbf{x}) = \arg\max_{c \in [m]} F_c(\mathbf{x})$	Class prediction of the DNN.
$\mathbf{x}^{(\ell)} := \mathbf{f}_\ell(\mathbf{x}) \in \mathbb{R}^{d_\ell}$, $\ell = 0, 1, \cdots, L$	Flattened layer representations of the DNN ($\ell = 0$ refers to the input).
C and $\widehat{C}$	Random variables corresponding to the true class and DNN-predicted class.
$\mathcal{D}_a = \{(\mathbf{x}_n^{(0)}, \cdots, \mathbf{x}_n^{(L)}, c_n, \hat{c}_n),\ n = 1, \cdots, N\}$	Labeled dataset with the layer representations, true class, and predicted class.
$\widehat{\mathcal{D}}_a(\ell, \hat{c})$	Subset of $\mathcal{D}_a$ corresponding to layer ℓ and predicted class $\hat{c}$.
$\mathcal{D}_a(\ell, c)$	Subset of $\mathcal{D}_a$ corresponding to layer ℓ and true class c.
$T(\mathbf{x}^{(\ell)}, \hat{c}, \widehat{\mathcal{D}}_a(\ell, \hat{c})) = T^{(\ell)}_{p \mid \hat{c}}$	Test statistic from layer ℓ conditioned on the predicted class $\hat{c}$.
$T(\mathbf{x}^{(\ell)}, c, \mathcal{D}_a(\ell, c)) = T^{(\ell)}_{s \mid c}$, $c \in [m]$	Test statistic from layer ℓ conditioned on a candidate true class c.
$\mathbf{t}_{p \mid \hat{c}} = [t^{(0)}_{p \mid \hat{c}}, \cdots, t^{(L)}_{p \mid \hat{c}}]$	Vector of test statistics from the layers conditioned on the predicted class $\hat{c}$.
$\mathbf{t}_{s \mid c} = [t^{(0)}_{s \mid c}, \cdots, t^{(L)}_{s \mid c}]$, $c \in [m]$	Vector of test statistics from the layers conditioned on a candidate true class c.
$q_{\text{agg}}(\mathbf{t}_{p \mid \hat{c}}), q_{\text{agg}}(\mathbf{t}_{s \mid 1}), \cdots, q_{\text{agg}}(\mathbf{t}_{s \mid m})$	Aggregate normalized test statistic from the predicted class and candidate true classes.
$(k_1^{(\ell)}, \cdots, k_m^{(\ell)}) \in \{0, 1, \cdots, k\}^m$ s.t. $\sum_i k_i^{(\ell)} = k$	Class counts from the k-nearest neighbors of a representation vector from layer ℓ.
$N_k^{(\ell)}(\mathbf{x}^{(\ell)}) \subset \{1, \cdots, N\}$	Index set of the k-nearest neighbors of a layer representation $\mathbf{x}^{(\ell)}$ relative to $\mathcal{D}_a$.

used test-time attacks on DNN classifiers are strongly white-box in that they assume a complete knowledge of the system. For a detailed discussion and taxonomy of adversarial attack methods, we refer readers to the recent surveys (Akhtar & Mian, 2018; Yuan et al., 2019).

A.2. Adversarial Defenses

On the defense side of adversarial learning, the focus can be broadly categorized into (1) *adversarial training* - where the objective is to train robust classifiers that generalize well in the face of adversarial attacks (Madry et al., 2018; Fawzi et al., 2016; 2018; Goodfellow et al., 2015), (2) *adversarial detection* - where the objective is to detect inputs that are adversarially manipulated by identifying signatures or patterns that make them different from natural inputs seen by the classifier at training time (Feinman et al., 2017; Xu et al., 2018; Lee et al., 2018; Ma et al., 2018). One approach to adversarial training involves augmenting the training set of the classifier with adversarial samples created from one or multiple attack methods along with their true labels, and retraining the classifier on the augmented training set (possibly initialized with parameters from a prior non-adversarial training) (Goodfellow et al., 2015). A limitation of this approach is that the resulting classifier may still fail on attacks that were not seen by the classifier during training. This has lead to research in the direction of robust optimization, where the learning objective of the classifier (usually an empirical risk function) is modified into a min-max optimization problem, with the inner maximization performed on a suitably chosen perturbation set (*e.g.*, an ℓ_∞ norm ball) (Madry et al., 2018). Adversarial detection, on the other hand, does not usually involve special training or modification of the underlying classifier to predict accurately on adversarial inputs. Instead, the focus is on methods that can detect adversarial inputs while operating at low false positive (natural inputs detected as adversarial) rates using ideas from the anomaly detection literature. The detector flags inputs that are suspicious and likely to be misclassified by the classifier so that they may be analyzed by an expert (possibly another ML system) for decision making down the line. There have been a plethora of works on the defense side adversarial learning. Recent surveys on this topic can be found in (Biggio & Roli, 2018; Miller et al., 2020).

A.3. Adversarial Samples as Anomalies

Adversarial detection is closely related to the problem of anomaly or outlier detection (Chandola et al., 2009; Zhao & Saligrama, 2009) with some important distinctions. The objective of anomaly detection is to determine if an input follows a pattern that is significantly different from that observed on a given data set. Stated differently, an input is said to be anomalous if it has a very low probability under the reference marginal distribution $p_0(\mathbf{x})$ underlying a given data set. On the other hand, adversarial in-

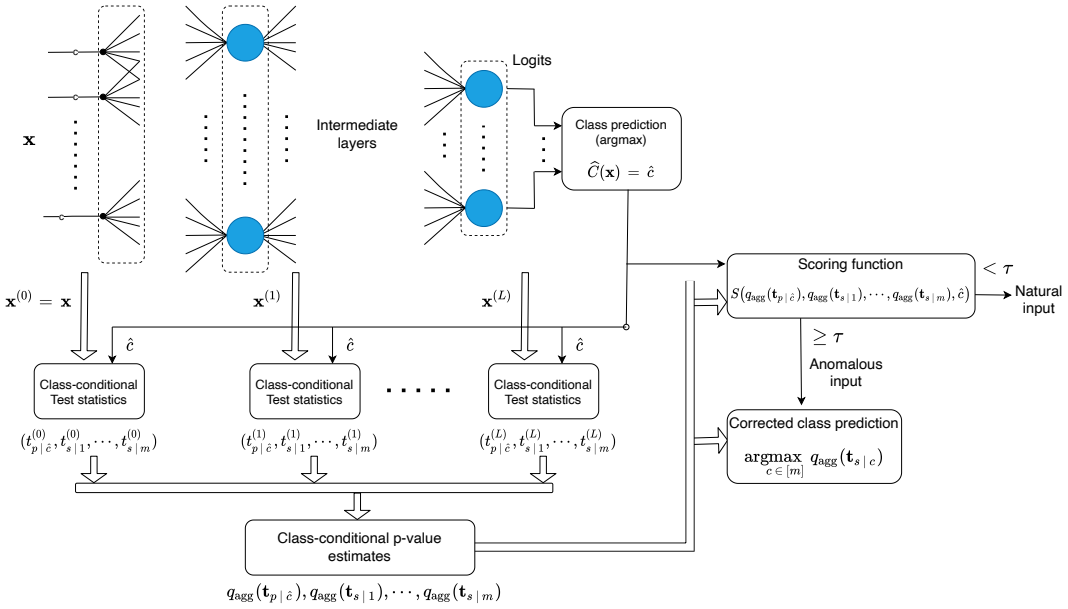


Figure 5: Overview of the proposed detection framework. The DNN classifies a test input $\mathbf{x}$ into one of the m classes. JTLA uses the predicted class and the layer representations to calculate $m+1$ class-conditional test statistics at each layer. The test statistics from the layers are aggregated into normalized p-value estimates, which are then used by the scoring function to determine whether the input is natural or anomalous.

puts from a test-time attack are not necessarily anomalous with respect to the marginal data distribution because of the way they are created by minimally perturbing a valid input $\mathbf{x} \sim p_0(\cdot \,|\, c)$ from a given class c. It is useful to consider the following notion of adversarial inputs. Suppose a clean input $\mathbf{x}$ from class c (*i.e.*, $\mathbf{x} \sim p_0(\cdot \,|\, c)$) is perturbed to create an adversarial input $\mathbf{x}'$ that appears to be from the same class c according to the true (Bayes) class posterior distribution, *i.e.*, $c = \arg\max_k p_0(k \,|\, \mathbf{x}')$. However, it is predicted into a different class c' by the classifier, *i.e.*, $\widehat{C}(\mathbf{x}') = \arg\max_k F_k(\mathbf{x}') = c'$. From this standpoint, we hypothesize that an adversarial input $\mathbf{x}'$ is likely to be a typical sample from the conditional distribution $p_0(\cdot \,|\, c)$ of the true class, while it is also likely to be an anomalous sample from the conditional distribution $p_0(\cdot \,|\, c')$ of the predicted class. We note that (Miller et al., 2019) use a similar hypothesis to motivate their detection method.

A.4. Related Works

We categorize and review some closely-related prior works on adversarial and OOD detection. For works that are based on multiple layer representations of a DNN, we discuss how the methods fit into the proposed meta-algorithm for anomaly detection.

Supervised Methods

In (Lee et al., 2018), the layer representations of a DNN are modeled class-conditionally using multivariate Gaussian densities, which leads to the Mahalanobis distance confidence score being used as a test statistic (feature) at the layers. The feature vector of Mahalanobis distances from the layers is used to train a binary logistic classifier for discriminating adversarial (or OOD samples) from natural samples. While this method uses the class-conditional densities of the layer representations, it uses a rather simple parametric model based on multivariate Gaussians, which may not be suitable for the intermediate layer representations (issues include high dimensionality, non-negative activations, and multimodality). Also, this method does not use the predicted class of the DNN; instead it finds the "closest" class corresponding to each layer representation. In the context of the meta-algorithm, this method does not explicitly apply a normalizing transformation to the test statistics. However, the Mahalanobis distance can be interpreted as the negative-log-density, which follows the Chi-squared distribution (at each layer) under the Gaussian density assumption. Finally, we note that using a binary logistic classifier is equivalent to using a weighted linear combination of the test statistics for scoring as shown in Appendix A.5.

(Ma et al., 2018) propose using the local intrinsic dimensionality (LID) of the layer representations as a test statistic for characterizing adversarial samples. Similar to the approach of (Lee et al., 2018), they calculate a test statistic vector of LID estimates from the layer representations, which is used for training a logistic classifier for discriminating adversarial samples from natural samples. In the context of the meta-algorithm, this method does not calculate class-

conditional test statistics. The LID is estimated based on the marginal distribution of the layer representation manifold. The LID test statistics are not normalized in any way, and use of the logistic classifier implies that this method also uses a weighted linear combination of the test statistics for scoring.

In (Yang et al., 2020), feature attribution methods are used to characterize the input and intermediate layer representations of a DNN. They find that adversarial inputs drastically alter the feature attribution map compared to natural inputs. Statistical dispersion measures such as the interquartile range (IQR) and median absolute deviation (MAD) are used to quantify the dispersion in the distribution of attribution values, which are then used as test statistics (features) to train a logistic classifier for discriminating adversarial samples from natural samples. In the context of the meta-algorithm, this method does not calculate class-conditional test statistics. The test statistics based on IQR or MAD are not normalized in any way. Similar to (Lee et al., 2018) and (Ma et al., 2018), this method uses a weighted linear combination of the test statistics for scoring.

Unsupervised Methods

(Roth et al., 2019) show that the log-odds ratio of inputs to a classifier (not necessarily a DNN) can reveal some interesting properties of adversarial inputs. They propose test statistics based on the expected log-odds of noise-perturbed inputs from different source and predicted class pairs. These test statistics are z-score normalized and thresholded to detect adversarial inputs. Their method does not use multiple layer representations for detection. They also propose a reclassification method for correcting the classifier's prediction on adversarial inputs. This is based on training logistic classifiers (one for each predicted class) that use the expected noise-perturbed log-odds ratio as features.

In (Zheng & Hong, 2018), the class-conditional distributions of fully-connected intermediate layer representations are modeled using Gaussian mixture models [12]. Inputs with a likelihood lower than a (class-specific) threshold under each class-conditional mixture model are rejected as adversarial. A key limitation of this method is that it is challenging to accurately model the high-dimensional layer representations of a DNN using density models such as Gaussian mixtures.

In (Miller et al., 2019), an anomaly detection method focusing on adversarial attacks is proposed, which in its basic form can be described as follows. The class-conditional density of the layer representations of a DNN are modeled using Gaussian mixture models, and they are used to estimate a Bayes class posterior at each layer. The Kullback-Leibler divergence between the class posterior of the DNN (based on the softmax function) and the estimated Bayes class posterior from each layer are used as test statistics for detecting adversarial samples. A number of methods are proposed for combining these class-conditional test statistics from the layers (*e.g.*, maximum and weighted sum across the layers). Their method does not apply any explicit normalization of the test statistics from the layers.

(Sastry & Oore, 2020) propose a method for detecting OOD samples based on analyzing the Gram matrices of the layer representations of a DNN. The Gram matrices of different orders (order 1 corresponds to the standard definition) capture the pairwise correlations between the elements of a layer representation. At training time, their detector records the element-wise minimum and maximum values of the Gram matrices (of different orders) from a training set of natural inputs, conditioned on each predicted class. The extent of deviation from the minimum and maximum values observed at training time is used to calculate a class-specific deviation test statistic at each layer. The final score for OOD detection is obtained by adding up the normalized deviations from the layers, where the normalization factor for a layer is the expected deviation observed on a held-out validation dataset. We note that the deviation statistic based on Gram matrices from the layer activations proposed in (Sastry & Oore, 2020) can be used as a test statistic for `JTLA` as well.

Confidence Metrics for Classifiers

Works such as the trust score (Jiang et al., 2018) and deep KNN (Papernot & McDaniel, 2018) have explored the problem of developing a confidence metric that can be used to independently validate the predictions of a classifier. Inputs with a low confidence score are expected to be misclassified and can be flagged as potentially adversarial or OOD. Deep KNN (Papernot & McDaniel, 2018) uses the class labels of the k-nearest neighbors of DNN layer representations to calculate a non-conformity score corresponding to each class. Large values of non-conformity corresponding to the predicted class indicate that an input may not have a reliable prediction. The method calculates empirical p-values of the non-conformity scores to provide a confidence score, credibility score, and an alternate (corrected) class prediction for test inputs. We note that the deep KNN test statistic based on the kNN class counts from the predicted class $k_{\hat{c}}^{(\ell)}$ and its complement $k - k_{\hat{c}}^{(\ell)}$ can be considered as a binomial specialization of the multinomial test statistic (§ 4.1). The trust score (Jiang et al., 2018) estimates the α-high density (level) set for each class, and calculates the distances from a test point to the α-high density sets from the classes to define a confidence metric. These methods can also be categorized as unsupervised.

[12] Convolutional layers are not modeled in their approach.

A.5. Scoring with a Logistic classifier

Consider a binary logistic classifier that takes a vector of test statistics $\mathbf{t}$ as input and produces an output probability for class 1 given by

$$P(Y = 1 \,|\, \mathbf{t}) \;=\; \frac{1}{1 \,+\, \exp(-\mathbf{w}^T\,\mathbf{t} \,-\, b)},$$

where $\mathbf{w}$ and b are weight vector and bias parameters. A detection decision of 1 (anomaly) is made when the output probability exceeds a threshold τ. It is easy to see that this results in the following decision rule:

$$\psi(\mathbf{x}^{(0)}, \cdots, \mathbf{x}^{(L)}, \hat{c}) \;=\; \begin{cases} 1 & \text{if } \mathbf{w}^T\,\mathbf{t} \,\geq\, \log\frac{\tau}{1-\tau} \,-\, b \\ 0 & \text{otherwise} \end{cases}$$

In other words, the score function is a weighted linear combination of the test statistics from the layers.

B. Harmonic Mean p-value Method

The harmonic mean p-value (HMP) (Wilson, 2019) is a recently proposed method for combining p-values from a large number of dependent tests. It is rooted in ideas from Bayesian model averaging and has some desirable properties such as robustness to positive dependency between the p-values, and an ability to detect small statistically-significant groups from a large number of p-values (tests). The main result of (Wilson, 2019) can be summarized as follows. Given a set of p-values $p_1, \cdots, p_m$ from m hypothesis tests, the weighted harmonic p-value of any subset $\mathcal{R} \subset \{1, \cdots, m\}$ of the p-values is given by

$$p_{\text{agg}}^{-1} \;=\; \frac{\sum\limits_{i \in \mathcal{R}} w_i p_i^{-1}}{\sum\limits_{i \in \mathcal{R}} w_i},$$

where the weights w_i are non-negative and satisfy $\sum_{i=1}^m w_i \,=\, 1$. In our problem, we apply the HMP method with the weights all set to the same value, resulting in the p-value aggregation function $r(\cdot)$ defined in Eq. (11).

In our experiments, we found the HMP method to have comparable or slightly worse performance than Fisher's method of combining p-values. Results from these ablation experiments can be found in Appendix E.2.

C. Details and Extensions of the Adaptive Attack

We first discuss the method we used for setting the scale of the Gaussian kernel per layer in the adaptive attack method of § 5. We then discuss an untargeted variant of the proposed adaptive attack, followed by an alternate formulation for the attack objective function.

C.1. Setting the Kernel Scale

For a given clean input $\mathbf{x}$, the scale of the Gaussian kernel for each layer σ_ℓ determines the effective number of samples that contribute to the soft count that approximates the kNN counts in Eq. (14). Intuitively, we would like the kernel to have a value close to 1 for points within a distance of η_ℓ (the kNN radius centered on $\mathbf{f}_\ell(\mathbf{x})$), and decay rapidly to 0 for points further away. Let $\mathcal{N}_k(\mathbf{f}_\ell(\mathbf{x}))$ denote the index set of the k-nearest neighbors of $\mathbf{f}_\ell(\mathbf{x})$ from the ℓ-th layer representations of the dataset $\mathcal{D}_a$. The probability of selecting the k-nearest neighbors from the set of N samples in $\mathcal{D}_a$ can be expressed as

$$s_1(\sigma_\ell) \;=\; \frac{\sum\limits_{n \,\in\, \mathcal{N}_k(\mathbf{f}_\ell(\mathbf{x}))} h_{\sigma_\ell}(\mathbf{f}_\ell(\mathbf{x}), \mathbf{f}_\ell(\mathbf{x}_n))}{\sum\limits_{n=1}^{N} h_{\sigma_\ell}(\mathbf{f}_\ell(\mathbf{x}), \mathbf{f}_\ell(\mathbf{x}_n))}.$$

We could choose σ_ℓ to maximize this probability and push it close to 1. However, this is likely to result in a very small value for σ_ℓ, which would concentrate all the probability mass on the nearest neighbor. To ensure that the probability mass is distributed sufficiently uniformly over the k-nearest neighbors we add the following normalized entropy [13] term as a regularizer

$$s_2(\sigma_\ell) \;=\; -\,\frac{1}{\log k} \sum_{i \,\in\, \mathcal{N}_k(\mathbf{f}_\ell(\mathbf{x}))} p_i(\sigma_\ell)\, \log p_i(\sigma_\ell),$$

where

$$p_i(\sigma_\ell) \;=\; \frac{h_{\sigma_\ell}(\mathbf{f}_\ell(\mathbf{x}), \mathbf{f}_\ell(\mathbf{x}_i))}{\sum\limits_{j \,\in\, \mathcal{N}_k(\mathbf{f}_\ell(\mathbf{x}))} h_{\sigma_\ell}(\mathbf{f}_\ell(\mathbf{x}), \mathbf{f}_\ell(\mathbf{x}_j))}.$$

Both terms $s_1(\sigma_\ell)$ and $s_2(\sigma_\ell)$ are bounded to the interval $[0, 1]$. We find a suitable σ_ℓ by maximizing a convex combination of the two terms, *i.e.*,

$$\max_{\sigma_\ell \,\in\, \mathbb{R}_+} \;(1 - \alpha)\, s_1(\sigma_\ell) \;+\; \alpha\, s_2(\sigma_\ell). \tag{15}$$

We set α to 0.5 in our experiments, and used a simple line search to find the approximate maximizer of Eq. (15).

C.2. Untargeted Attack Formulation

The formulation in § 5, where a specific class $c' \neq c$ is chosen, is used to create a targeted attack. Alternatively, a simple modification to Eq. (13) that considers the log-odds of class c, $\log\frac{p_c}{1-p_c}$, can be used to create an untargeted attack, resulting in the following objective function to be minimized:

[13] The entropy is divided by the maximum possible value of $\log k$ to scale it to the range $[0, 1]$.

$$J(\boldsymbol{\delta}) \;=\; \|\boldsymbol{\delta}\|_2^2 \;+\; \lambda \log \sum_{\ell=0}^{L} \sum_{\substack{n=1: \\ c_n=c}}^{N} h_{\sigma_\ell}(\mathbf{f}_\ell(\mathbf{x}+\boldsymbol{\delta}), \mathbf{f}_\ell(\mathbf{x}_n)) \\ -\; \lambda \log \sum_{\ell=0}^{L} \sum_{\substack{n=1: \\ c_n\neq c}}^{N} h_{\sigma_\ell}(\mathbf{f}_\ell(\mathbf{x}+\boldsymbol{\delta}), \mathbf{f}_\ell(\mathbf{x}_n)). \qquad (16)$$

C.3. Alternate Attack Loss Function

Starting from equation Eq. (13), but now assuming that the probability estimate for each class based on the kNN model factors into a product of probabilities across the layers of the DNN (*i.e.*, independence assumption), we get

$$\log \frac{p_c}{p_{c'}} \;=\; \log \frac{k_c}{k_{c'}} \;=\; \log \frac{\prod_\ell k_c^{(\ell)} \,/\, k}{\prod_\ell k_{c'}^{(\ell)} \,/\, k} \\ =\; \sum_{\ell=0}^{L} \log k_c^{(\ell)} \;-\; \sum_{\ell=0}^{L} \log k_{c'}^{(\ell)}.$$

Using the soft count approximation based on the Gaussian kernel (as before) leads to the following alternative loss function for the targeted adaptive attack

$$J(\boldsymbol{\delta}) \;=\; \|\boldsymbol{\delta}\|_2^2 \;+\; \lambda \sum_{\ell=0}^{L} \log \sum_{\substack{n=1: \\ c_n=c}}^{N} h_{\sigma_\ell}(\mathbf{f}_\ell(\mathbf{x}+\boldsymbol{\delta}), \mathbf{f}_\ell(\mathbf{x}_n)) \\ -\; \lambda \sum_{\ell=0}^{L} \log \sum_{\substack{n=1: \\ c_n=c'}}^{N} h_{\sigma_\ell}(\mathbf{f}_\ell(\mathbf{x}+\boldsymbol{\delta}), \mathbf{f}_\ell(\mathbf{x}_n)). \qquad (17)$$

In contrast to Eq. (14), this loss function considers the class probability estimates from each layer, instead of a single class probability estimate based on the cumulative kNN counts across the layers. A special case of the loss function Eq. (17) that includes only the final (logit) layer of the DNN $\mathbf{f}_L(\mathbf{x})$ can be directly compared to the Carlini-Wagner attack formulation (Carlini & Wagner, 2017b). With this formulation, the logit layer representations of an adversarial input will be guided closer to the neighboring representations from class c', and away from the neighboring representations from class c.

D. Additional Implementation Details

D.1. Computing Platform

Our experiments were performed on a single server running Ubuntu 18.04 with 128 GB memory, 4 NVIDIA GeForce RTX 2080 GPUs, and 32 CPU cores.

D.2. Datasets & DNN Architectures

A summary of the image classification datasets we used, the architecture and test set performance of the corresponding DNNs are given in Table 2. Each dataset has 10 image classes. We followed recommended best practices for training DNNs on image classification problems (using techniques like Dropout). We did not train a DNN on the NotMNIST dataset because this dataset is used only for evaluation in the OOD detection experiments.

D.3. Method Implementations

The code associated with our work can be accessed here [14]. We utilized the authors original implementation for the following methods: (i) deep mahalanobis detector [15], (ii) the odds are odd detector [16]. We implemented the remaining detection methods, viz. `LID`, `DKNN`, and `Trust`, and this is available as part of our released code. All implementations are in `Python3` and are based on standard scientific computing libraries such as `Numpy`, `Scipy`, and `Scikit-learn` (Pedregosa et al., 2011). We used `PyTorch` as the deep learning and automatic differentiation backend (Paszke et al., 2017). We perform approximate nearest neighbor search using the `NNDescent` method (Dong et al., 2011) to efficiently construct and query from kNN graphs at the DNN layer representations. We used the implementation of NNDescent provided by the library `PyNNDescent` [17]. Our implementation of `JTLA` is highly modular, allowing for new test statistics to be easily plugged in to the existing implementation. We provide implementations of the following test statistics: (i) multinomial class counts, (ii) binomial class counts, (iii) trust score, (iv) local intrinsic dimensionality, and (v) average kNN distance.

In our experiments with `JTLA`, where Fisher's method or HMP method are used for combining the p-values, we included p-values estimated from the individual layers (listed in Tables 3, 4, and 5) and from all distinct layer pairs. The number of nearest neighbors k is the **only hyperparameter** of the proposed method. This is set to be a function of the number of in-distribution training samples n using the heuristic $k = \lceil n^{0.4} \rceil$. For methods like `DKNN`, `LID`, and `Trust` that also depend on k, we found that setting k in this way produces comparable results to that obtained over a range of k values. Therefore, to maintain consistency, we set k for all the (applicable) methods using the rule $k = \lceil n^{0.4} \rceil$.

For the method `Trust`, we applied the trust score to the input, logit (pre-softmax) layer, and the fully-connected layer prior to the logit (pre-logit) layer. Since it was reported in (Jiang et al., 2018) that the trust score works well mainly in low-dimensional settings, we applied the same dimension-

[14] https://github.com/jayaram-r/adversarial-detection

[15] https://github.com/pokaxpoka/deep_Mahalanobis_detector

[16] https://github.com/yk/icml19_public

[17] https://github.com/lmcinnes/pynndescent

Table 2: Datasets & DNN Architectures.

Dataset	Input dimension	Train size	Test size	Test accuracy (%)	DNN architecture
MNIST (LeCun et al., 1998)	$28 \times 28 \times 1$	50,000	10,000	99.12	2 Conv. + 2 FC layers (LeCun et al., 1998)
SVHN (Netzer et al., 2011)	$32 \times 32 \times 3$	73,257	26,032	89.42	2 Conv. + 3 FC layers
CIFAR-10 (Krizhevsky et al., 2009)	$32 \times 32 \times 3$	50,000	10,000	95.45	ResNet-34 (He et al., 2016)
Not-MNIST (Bulatov, 2011)	$28 \times 28 \times 1$	500,000	18,724	N/A	N/A

ality reduction that was applied to JTLA (see Tables 3, 4, and 5) on the input and pre-logit layer representations. We found the pre-logit layer to produce the best detection performance in our experiments. Hence, we report results for Trust with the pre-logit layer in all our experiments. The constant α, which determines the fraction of samples with lowest empirical density to be excluded from the density level sets, is set to 0 in our experiments. We explored a few other values of α, but did not find a significant difference in performance. This is consistent with the observation in Section 5.3 of (Jiang et al., 2018).

The methods Mahalanobis and LID train a logistic classifier to discriminate adversarial samples from natural samples. The regularization constant of the logistic classifier is found by searching (over a set of 10 logarithmically spaced values between 0.0001 to 10000) for the value that leads to the largest average test-fold area under the ROC curve, in a 5-fold stratified cross-validation setup.

For the method Odds, the implementation of the authors returns a binary (0 / 1) detection decision instead of a continuous score that can be used to rank adversarial samples. The binary decision is based on applying z-score normalization to the original score, and comparing it to the 99.9-th percentile of the standard Gaussian density. Instead of using the thresholded decision, we used the z-score-normalized score of Odds in order to get a continuous score that is required by the metrics average precision and pAUC-α.

Details on the DNN Layer Representations

The DNN layers used in our experiments and their raw dimensionality are listed for the three datasets in Tables 3, 4, and 5. An exception to this is the LID method, for which we follow the implementation of (Ma et al., 2018) and calculate the LID features from all the intermediate layers. For DKNN and LID, we did not apply any dimensionality reduction or pre-processing of the layer representations to be consistent with the respective papers. For Mahalanobis, the implementation of the authors performs global average pooling at each convolutional layer to transform a $C \times W \times H$ tensor (with C channels) to a C-dimensional vector.

For JTLA, we use the neighborhood preserving projection (NPP) method (He et al., 2005) to perform dimensionality reduction on the layer representations. We chose NPP because it performs an efficient linear projection that attempts to preserve the neighborhood structure in the original space as closely as possible in the projected (lower dimensional) space. Working with the lower dimensional layer representations mitigates problems associated with the curse of dimensionality, and significantly reduces the memory utilization and the running time of JTLA. The original dimension, intrinsic dimension estimate, and the projected dimension of the layer representations for the three datasets are listed in Tables 3, 4, and 5. We did not apply dimension reduction to the logit layer because it has only 10 dimensions.

Table 3: Layer representations of the DNN trained on MNIST. The output of the block listed in the first column is used as the layer representation.

Layer block	Layer index	Original dimension	Intrinsic dimension	Projected dimension
Input	0	784	13	31
Conv-1 + ReLu	1	21632	22	53
Conv-2 + Maxpool + Dropout	2	9216	18	77
FC-1 + ReLu + Dropout	3	128	9	90
FC-2 (Logit)	4	10	6	10

Table 4: Layer representations of the DNN trained on SVHN. The output of the block listed in the first column is used as the layer representation.

Layer block	Layer index	Original dimension	Intrinsic dimension	Projected dimension
Input	0	3072	18	43
Conv-1 + ReLu	1	57600	38	380
Conv-2 + ReLu + Maxpool + Dropout	2	12544	42	400
FC-1 + ReLu + Dropout	3	512	12	120
FC-2 + ReLu + Dropout	4	128	7	10
FC-3 (Logit)	5	10	4	10

The procedure we used for determining the projected dimension is summarized as follows. We used the training partition of each dataset (that was used to train the DNN) for this task in order to avoid introducing any bias on the test partition (which is used for the detection and corrected classification experiments). At each layer, we first estimate the intrinsic dimension (ID) as the median of the LID estimates of the samples, found using the method of (Amsaleg et al., 2015). The ID estimate d_{int} is used as a lower bound

for the projected dimension. Using a 5-fold stratified cross-validation setup, we search over 20 linearly spaced values in the interval $[d_{\text{int}}, 10\, d_{\text{int}}]$ for the projected dimension (found using NPP) that results in the smallest average test-fold error rate for a standard k-nearest neighbors classifier. The resulting projected dimensions for each dataset (DNN architecture) are given in Tables 3, 4, and 5.

Table 5: Layer representations of the ResNet-34 trained on CIFAR-10. The output of the block listed in the first column is used as the layer representation. Legend: RB - Residual block, BN - BatchNorm

Layer block	**Layer index**	**Original dimension**	**Intrinsic dimension**	**Projected dimension**
Input	0	3072	25	48
Conv-1 + BN + ReLU	1	65536	33	330
RB-1	2	65536	58	580
RB-2	3	32768	59	590
RB-3	4	16384	28	28
RB-4	5	8192	15	15
2D Avg. Pooling	6	512	9	9
FC (Logit)	7	10	8	10

Note on Performance Calculation

The following procedure is used to calculate performance metrics as a function of the perturbation norm of adversarial samples. Suppose there are N_a adversarial samples and N_n natural samples in a test set, with $N = N_a + N_n$. Define the maximum proportion of adversarial samples $p_a = N_a \,/\, N$. The adversarial samples are first sorted in increasing order of their perturbation norm. The proportion of adversarial samples is varied over 12 equally-spaced values between 0.005 and $\min(0.3, p_a)$. For a given proportion p_i, the top $N_i = \lceil p_i\, N \rceil$ adversarial samples (sorted by norm) are selected. The perturbation norm of all these adversarial samples will be below a certain value; this value is shown on the x-axis of the performance plots. The performance metrics (average precision, pAUC-α etc.) are then calculated from the N_i adversarial samples and the N_n natural samples.

In order to calculate the performance metrics as a function of the proportion of adversarial or OOD (anomalous) samples, the only difference is that we do not sort the anomalous samples in a deterministic way. For a given proportion p_i, we select $N_i = \lceil p_i\, N \rceil$ anomalous samples at random uniformly, and calculate the performance metrics based on the N_i anomalous and N_n natural samples. To account for variability, this is repeated over 100 random subsets of N_i anomalous samples each, and the median value of the performance metrics is reported. Note that the detection methods need to score the samples only once, and the above calculations can be done based on the saved scores.

D.4. Details on the Adversarial Attacks

We list below the parameters of the adversarial attack methods we implemented using Foolbox (Rauber et al., 2017). We utilize the same variable names used by Foolbox in order to enable easy replication.

- PGD attack (Madry et al., 2018): ℓ_∞ norm with the norm-ball radius ϵ linearly spaced in the interval $[\frac{1}{255}, \frac{21}{255}]$. Using the notation of the Foolbox library: `epsilon = [`$\frac{1}{255}, \frac{3}{255}, \cdots, \frac{21}{255}$`], stepsize = 0.05, binary_search = False, iterations = 40, p_norm = inf.`
- Carlini-Wagner (CW) attack (Carlini & Wagner, 2017b): ℓ_2 norm with the confidence parameter of the attack varied over the set $\{0, 6, 14, 22\}$. Using the notation of the Foolbox library: `confidence = [0,6,14,22], max_iterations = 750, p_norm = 2.`
- FGSM attack (Goodfellow et al., 2015): ℓ_∞ norm with maximum norm-ball radius $\epsilon_{\max} = 1$. Using the notation of the Foolbox library: `max_epsilon = 1, p_norm = inf.`

For the adversarial detection experiments in § 6.1, we reported results for the attack parameter setting that would produce adversarial samples with the lowest perturbation norm (least perceptible), in order to make the detection task challenging. This corresponds to $\epsilon = \frac{1}{255}$ for the PGD attack, confidence $= 0$ for the CW attack, and $\epsilon_{\max} = 1$ for the FGSM attack. These same parameters are also used in the experiments in Appendix F.2, F.3, F.4, F.6, and F.7. In Appendix F.1, the strength of the attack is varied and the attack parameters used are described there. The adversarial samples are generated *once* from the clean samples corresponding to each train fold and test fold (from cross-validation), and saved to files for reuse in all the experiments. This way, we ensure that there is no randomness in the experiments arising due to the adversarial sample generation.

Adaptive Attack:

Here we provide additional details on the adaptive (defense-aware) attack method proposed in § 5. The constant λ in the objective function controls the perturbation norm of the adversarial sample, and smaller values of λ lead to solutions with a smaller perturbation norm. We follow the approach of (Carlini & Wagner, 2017b) to set λ to the smallest possible value that leads to a successful adversarial perturbation. This is found using a bisection line search over ten steps. An adversarial input is considered successful if it modifies the initially-correct class prediction of the defense method. We also follow the approach of (Carlini & Wagner, 2017b) to implicitly constraint the adversarial inputs to lie

in the same range as the original inputs using the hyperbolic tangent and its inverse function. Suppose the inputs lie in the range $[a, b]^d$, the following sequence of transformations is applied to each component of the vectors

$$\begin{aligned} z_i &= \tanh^{-1}(2\,\frac{x_i - a}{b - a} - 1) \\ z'_i &= z_i + w_i \\ x'_i &= (b - a)\,\frac{1}{2}\,(1 + \tanh(z_i + w_i)) + a \end{aligned}$$

effectively allowing the perturbation $\mathbf{w}$ to be optimized over $\mathbb{R}^d$. The resulting unconstrained optimization is solved using the RMSProp variant of stochastic gradient descent with adaptive learning rate (Ruder, 2016). The maximum number of gradient descent steps is set to 1000. For all experiments based on the adaptive attack method, the attack targets the variant of `JTLA` based on p-value normalization at the individual layers and layer pairs, Fisher's method for p-value aggregation, and the multinomial test statistic.

E. Additional Experiments

In this section, we supplement the results in § 6 with more extensive experiments.

E.1. Attack Transfer and Attacks of Varying Strength

We evaluate the performance of the detection methods on a task with different adversarial attacks used in the train and test sets. The strength of the attacks are also varied in both the train and test sets. Recall that we use a 5-fold cross-validation framework for evaluation. Hence, for a train/test split corresponding to fold $i \in \{1, 2, 3, 4, 5\}$, adversarial samples from Attack A are generated in the train split, while adversarial samples from Attack B are generated in the test split. Supervised methods, `Mahalanobis` and `LID`, learn using both the adversarial and clean samples from the train split, while the unsupervised methods, `JTLA`, `Odds`, `DKNN`, and `Trust`, learn using only the clean samples from the train split.

ℓ_2-CW attack to ℓ_∞-PGD attack

The ℓ_∞-PGD (Madry et al., 2018) attack is applied on the test split with perturbation strength parameter ϵ varied over the values $\{1/255, 3/255, 5/255, 7/255, 9/255\}$. For each clean sample, one of these ϵ values is randomly selected and used to create an attack sample. The ℓ_2-CW attack (Carlini & Wagner, 2017b) is applied to the train split with the confidence parameter value randomly selected from the set $\{0, 6, 14, 22\}$. The result of this experiment is given in Figure 6, with the average precision of the compared methods plotted as a function of the perturbation norm on the x-axis. The ℓ_∞ norm is used on the x-axis to match the norm type used by the PGD attack (that is applied to the test split). We observe that `Mahalanobis` and both variants of `JTLA` have the best performance on CIFAR-10, while both variants of `JTLA` outperform the other methods on SVHN. On MNIST, `JTLA` based on the aK-LPE method (§ 4.2.B) and `Trust` have the best performance. The methods `Mahalanobis`, `Trust`, and `Odds` do not consistently have good performance, while the `LID` method has poor performance on all datasets.

ℓ_∞-PGD attack to ℓ_2-CW attack

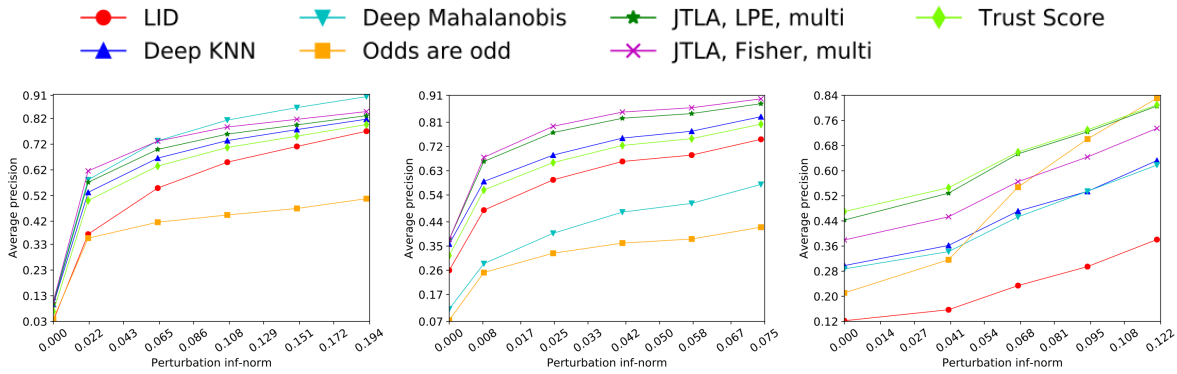
In this experiment, the ℓ_2-CW attack is applied to the test split and the ℓ_∞-PGD attack is applied to the train split. The attack parameters (strength) are varied as described earlier. The result of this experiment is given in Figure 7, with the average precision of the methods plotted as a function of the ℓ_2 norm of the attack (since the ℓ_2-CW attack is applied to the test split). From the figure, we make the following observations. On CIFAR-10, `Odds` outperforms the other methods, followed by the two variants of `JTLA`. On SVHN, the two variants of `JTLA` have the best performance, followed by `Odds`. On MNIST, the methods `JTLA`, `Odds`, and `Trust` have very similar average precision that is higher than the other methods. The methods `Mahalanobis`, `DKNN`, and `LID` do not have good performance in this experiment. We hypothesize that the `Odds` method (Roth et al., 2019) works well in detecting CW attacks because it uses a test statistic based on the noise-perturbed log-odds ratio of all pairs of classes, which is well-matched to the CW attack that is based on skewing the log-odds ratio of the class pair involved in the attack.

E.2. Ablation Experiments

We discuss the ablation experiments that we performed to get a better understanding of the components of the proposed method. Specifically, we are interested in understanding

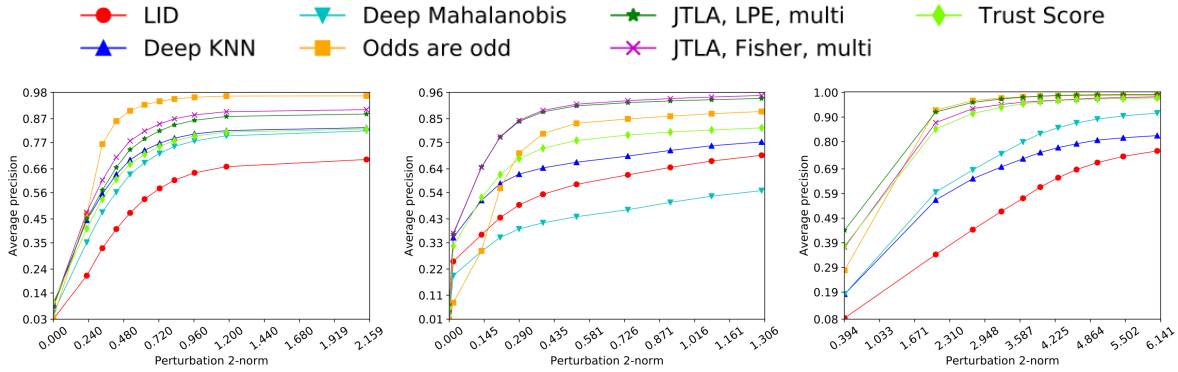
1. The relative performance of the proposed p-value based normalization methods (§ 4.2) and the p-value aggregation methods (§ 4.3).
2. The value of including p-values from test statistics at all layer pairs (in addition to the individual layers).
3. The relative performance of using only the last few layer representations for detection.
4. The relative performance of the two scoring methods in § 4.4 on the task of adversarial detection.

Tables 6, 7, and 8 summarize the results of these experiments on the task of detecting adversarial samples from the ℓ_∞-PGD attack with $\epsilon = 1/255$, and the ℓ_2-CW attack with confidence set to 0. DNNs trained on the SVHN and CIFAR-10 datasets (described earlier) are used, and average precision and pAUC-0.2 (partial AUROC below FPR $= 0.2$) are reported as detection metrics. Unlike the results in Fig. 2 and Fig. 3, we do not vary the proportion of adversarial samples, and report performance with all the adversarial



(a) CIFAR-10 (b) SVHN (c) MNIST

Figure 6: Average precision on the attack transfer experiment: CW to PGD attack.



(a) CIFAR-10 (b) SVHN (c) MNIST

Figure 7: Average precision on the attack transfer experiment: PGD to CW attack.

samples included.

Table 6 focuses on points 1 and 2, and compares the proposed p-value normalization and aggregation methods. P-values from the layer pairs are included in the first case and not included in the second case. The best performing configuration (across the rows) is highlighted in bold. We observe that p-value normalization at the layers with aggregation using Fisher's method has the best performance in most cases. Including the layer pairs did not result in a significant improvement in these experiments. It is surprising that Fisher's method has better (in some cases comparable) performance compared to the HMP and the aK-LPE methods despite its simplistic assumption of independent tests (p-values). We believe this can be attributed to the fact that the multivariate p-values estimated by the aK-LPE method (Eq. (9)) require a large sample size to converge to their true values. Since we apply this estimator class conditionally, the moderate number of samples per class (ranging from 500 to 5000 in our experiments) may result in estimation errors. Also, we conjecture that Fisher's method achieves a higher detection rate (TPR) at the expense of a higher FPR, while the HMP method has a more conservative TPR with a lower FPR.

Table 7 focuses on point 3 and compares the performance of using all the layer representations (listed in tables 4 and 5) with the performance from using only the final one, two, or three layer representations [18]. We observe that including more layers generally increases the detection performance, confirming the intuition behind using multiple layers. For the CW attack on CIFAR-10, using only the final (logit) layer has comparable performance to using all the layers. This is consistent with the design of the CW attack based on only the logit layer representation.

Table 8 focuses on point 4 to understand if the score function (12) is better suited for adversarial samples since it considers the aggregate p-values from the candidate true classes. From the table, we observe that the adversarial score function clearly outperforms the OOD score function for each combination of normalization and aggregation method.

[18] We focus on the deeper layers since their representations are most useful for classification.

Table 6: Ablation experiment - performance of different p-value normalization and aggregation methods, and the effect of including/excluding layer pairs. The multinomial test statistic is used in all cases.

Normalization method	Aggregation method	SVHN, PGD ($\epsilon = 1/255$)		SVHN, CW (confidence = 0)		CIFAR-10, PGD ($\epsilon = 1/255$)		CIFAR-10, CW (confidence = 0)	
		average precision	pAUC-0.2	average precision	pAUC-0.2	average precision	pAUC-0.2	average precision	pAUC-0.2
p-values from layers & layer pairs (§4.2.A)	Fisher	0.7382	**0.8025**	0.9631	0.9213	0.7710	**0.7790**	0.9664	0.9377
	HMP	0.7296	0.7966	0.9611	0.9171	0.7614	0.7705	0.9653	0.9344
p-values from layers	Fisher	**0.7393**	0.8005	**0.9634**	**0.9214**	**0.7734**	0.7781	**0.9667**	**0.9380**
	HMP	0.7247	0.7925	0.9591	0.9140	0.7538	0.7617	0.9616	0.9315
Multivariate p-value (aK-LPE, §4.2.B)	None	0.7161	0.7840	0.9559	0.9042	0.7437	0.7518	0.9650	0.9296

Table 7: Ablation experiment: effect of including only the last few layers for detection.

Normalization method	Aggregation method	Layers included	SVHN, PGD ($\epsilon = 1/255$)		SVHN, CW (confidence = 0)		CIFAR-10, PGD ($\epsilon = 1/255$)		CIFAR-10, CW (confidence = 0)	
			average precision	pAUC-0.2	average precision	pAUC-0.2	average precision	pAUC-0.2	average precision	pAUC-0.2
p-values from layers & layer pairs (§4.2.A)	Fisher	All	**0.7382**	**0.8025**	**0.9631**	**0.9213**	**0.7710**	**0.7790**	0.9664	0.9377
		Final (logits)	0.6347	0.7396	0.9141	0.8538	0.7399	0.7636	0.9664	0.9371
		Last 2	0.6440	0.7431	0.9213	0.8599	0.7410	0.7650	**0.9688**	**0.9401**
		Last 3	0.6847	0.7674	0.9388	0.8857	0.7473	0.7657	0.9660	0.9358

E.3. Results on the FGSM Attack

Figure 8 presents the average precision of the different detection methods as a function of the ℓ_2 norm of perturbation for the FGSM attack method. It is clear that both variants of JTLA outperform the other methods, consistent with the trend observed on other attacks in § 6.

E.4. Evaluation of Partial Area Under the ROC Curve

Here we compare the performance of methods using pAUC-0.2, a metric calculating the partial area under the ROC curve for FPR below 0.2. Comparing the area under the entire ROC curve can lead to misleading interpretations because it includes FPR values that one would rarely choose to operate in. Therefore, to reflect realistic operating conditions, we chose a maximum FPR of 0.2. Recall that for the PGD attack we vary the proportion of adversarial samples along the x-axis because most of the samples from this attack have the same perturbation norm.

On the CIFAR-10 dataset (Figure 9), we observe that JTLA has better performance than the other methods in most cases. On the adaptive attack, Mahalanobis has slightly better performance than JTLA with the multivariate p-value estimation method (aK-LPE). We make similar observations on the SVHN dataset (Figure 10), with a minor exception on the adaptive attack where the Odds method performs better than JTLA as the perturbation norm increases.

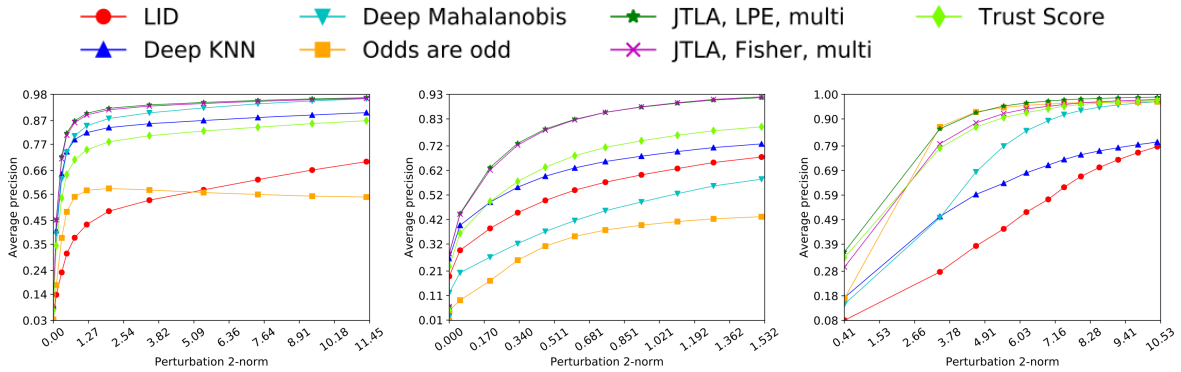
On the MNIST dataset (Figure 11), we observe some different trends in the performance compared to the other datasets. On the CW and FGSM attacks, the methods Odds and Trust perform comparably or slightly better than JTLA (particularly the variant based on Fisher's method). On the adaptive attack, the performance of JTLA based on Fisher's method decreases significantly as the perturbation norm increases. On the other hand, the variant of JTLA based on the aK-LPE method outperforms the other methods on this attack. We think that this contrast in performance is due to the fact that the adaptive attack samples were optimized to fool the variant of JTLA based on Fisher's method. Also, attack samples with higher perturbation norm are more likely to be successful. On the PGD attack, Odds outperforms the other methods, but the pAUC-0.2 of all methods, except LID and DKNN, are higher than 0.95 in this case. We conjecture that the good performance of most methods on MNIST could be due to the simplicity of the input space and the classification problem.

E.5. Results on the MNIST Dataset

In Figure 12, we compare the average precision of different methods on the MNIST dataset for the CW, PGD, and adaptive attacks (results for the FGSM attack were presented in Appendix E.3). We observe that Odds has good performance on this dataset, outperforming JTLA in some cases. The method Trust (which uses the pre-logit, fully connected layer) also performs well on this dataset. This could be due to the fact that on the MNIST dataset, the attack samples exhibit very distinctive patterns at the logit and pre-logit DNN layers, which are the focus of the methods Odds and Trust. We note that Odds and Trust do not carry over this good performance to all datasets and attacks. Also, both variants of JTLA perform well in the low perturbation norm regime.

Table 8: Ablation experiment: comparison of the adversarial and OOD score functions (§4.4) on different adversarial attacks. The adversarial score function outperforms the OOD score function in all cases.

Normalization method	Aggregation method	Scoring method	SVHN, PGD ($\epsilon = 1/255$)		SVHN, CW (confidence = 0)		CIFAR-10, PGD ($\epsilon = 1/255$)		CIFAR-10, CW (confidence = 0)	
			average precision	pAUC-0.2	average precision	pAUC-0.2	average precision	pAUC-0.2	average precision	pAUC-0.2
p-values from layers & layer pairs (§4.2.A)	Fisher	Adversarial	**0.7382**	**0.8025**	**0.9631**	**0.9213**	**0.7710**	**0.7790**	**0.9664**	**0.9377**
		OOD	0.7078	0.7736	0.9524	0.8973	0.7492	0.7612	0.9620	0.9253
	HMP	Adversarial	**0.7296**	**0.7966**	**0.9611**	**0.9171**	**0.7614**	**0.7705**	**0.9653**	**0.9344**
		OOD	0.6946	0.7617	0.9482	0.8882	0.7396	0.7501	0.9599	0.9205
Multivariate p-value (aK-LPE, §4.2.B)	None	Adversarial	**0.7161**	**0.7840**	**0.9559**	**0.9042**	**0.7437**	**0.7518**	**0.9650**	**0.9296**
		OOD	0.6986	0.7664	0.9511	0.8941	0.7065	0.7247	0.9575	0.9149

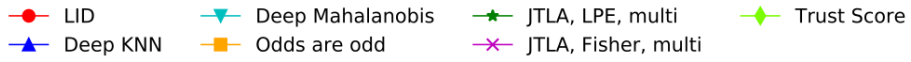


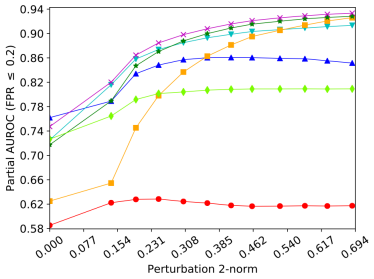
(a) CIFAR-10 (b) SVHN (c) MNIST

Figure 8: Average precision on the FGSM attack ($\epsilon_{\max} = 1$) for all datasets.

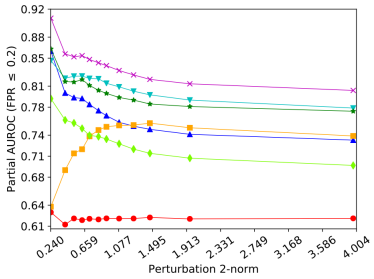
E.6. Running Time Comparison

Table 9 reports the wall-clock running time (in minutes) of the different detection methods per-fold, averaged across all attack methods. `Trust` consistently has the least running time, while both variants of `JTLA` have low running time as well. The running time of `Mahalanobis` is comparable to `JTLA` on MNIST and SVHN, but is higher on CIFAR-10. This is because `Mahalanobis` performs a search for the best noise parameter at each layer using 5-fold cross-validation, which takes a longer time on the Resnet-34 DNN for CIFAR-10. `Odds` and `LID` have much higher running time compared to the other methods. For each test sample, `Odds` computes an expectation over noisy inputs (from a few different noise parameters), which increases its running time as the size of the DNN increases. The computation involved in estimating the LID features at the DNN layers increases with the sample size used for estimation and the number of layers.

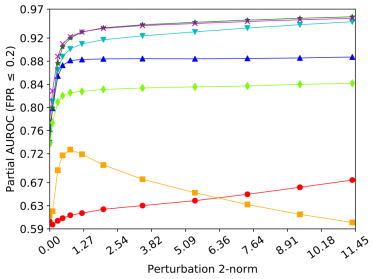




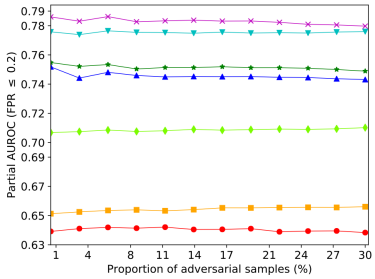
(a) CW, confidence $= 0$



(b) Adaptive attack

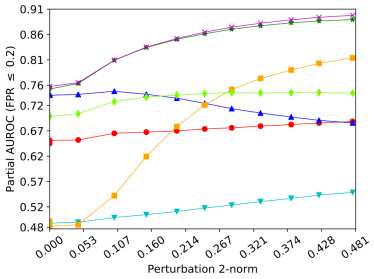


(c) FGSM, $\epsilon_{\max} = 1$

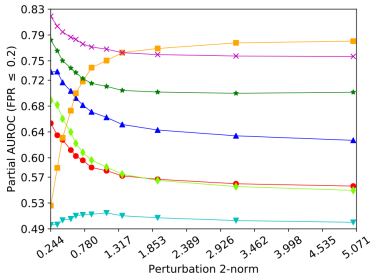


(d) PGD, $\epsilon = 1/255$

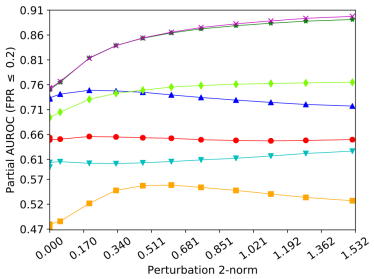
Figure 9: CIFAR-10 experiments: pAUC-0.2 for different attacks.



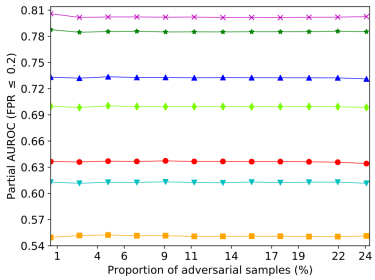
(a) CW, confidence $= 0$



(b) Adaptive attack

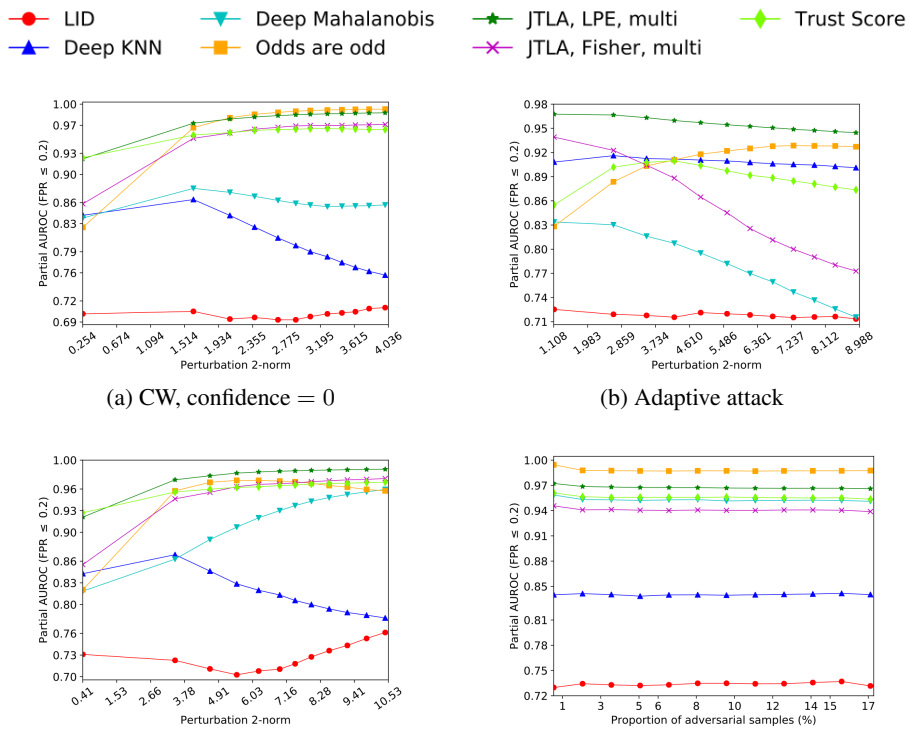


(c) FGSM, $\epsilon_{\max} = 1$



(d) PGD, $\epsilon = 1/255$

Figure 10: SVHN experiments: pAUC-0.2 for different attacks.



(a) CW, confidence $= 0$

(b) Adaptive attack

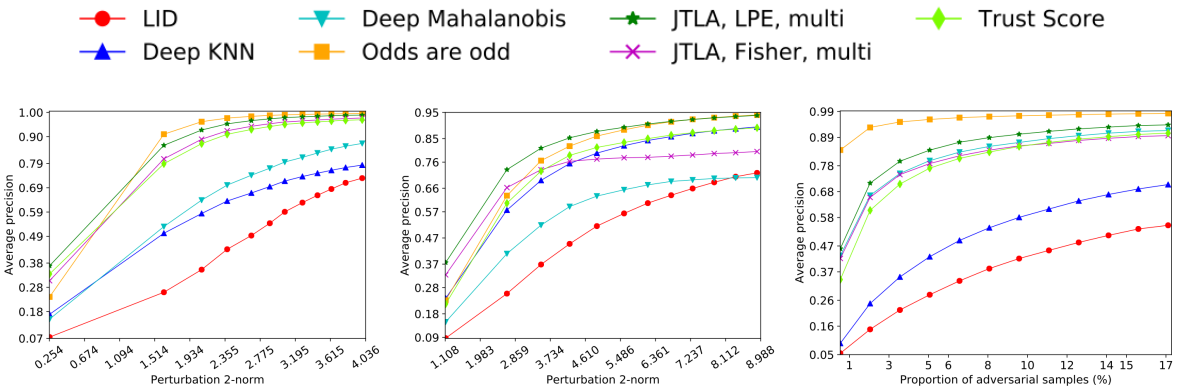
(c) FGSM, $\epsilon_{\max} = 1$

(d) PGD, $\epsilon = 1/255$

Figure 11: MNIST experiments: pAUC-0.2 for different attacks.

Table 9: Average wall-clock running time per-fold (in minutes) for the different detection methods.

Dataset	JTLA, Fisher	JTLA, LPE	Mahalanobis	Odds	LID	DKNN	Trust
CIFAR-10	2.73	2.18	15.08	142.94	49.74	11.99	0.53
SVHN	6.37	5.01	4.19	33.60	100.80	23.54	0.60
MNIST	0.92	0.85	1.18	6.79	6.96	1.73	0.24



(a) CW, confidence $= 0$

(b) Adaptive attack

(c) PGD, $\epsilon = 1/255$

Figure 12: MNIST experiments: average precision for different attacks.