DeepWatch: White-Box Watchpoint for Deep Neural Networks

ABSTRACT

Program state inspection through breakpoint debugging is a vital part of any debugging process which ensures the correctness of software under development. In contrast, deep neural network based models lack the transparency to inspect hidden states making it difficult to interpret the internal mechanics. In traditional Software Engineering, an erroneous behavior of a program could be isolated using interactive debugging. But currently, state of the art deep networks do not have a well defined behavioural inspection technique for perturbed inputs.

To enable ML developers to detect the quality of unseen inputs and have a mechanism to detect adversarial inputs, we design a completely automated debugging and inspection technique, DeepWatch. It takes a pretrained model as an input and inserts additional inspection layers or side channels to provide transparency by measuring the KL divergences between the distribution of hidden layer activations against corresponding templates that explicitly encode each class label. We then use them as features for the downstream task of anomalous input detection.

We propose a novel technique for adversarial example detection without changing the parameters of the original network and demonstrate the effectiveness of our approach on the MNIST and CIFAR-10 dataset, achieving detection AUC scores of 0.99 on the standard CW attack algorithm for the MNIST dataset.

MK: while there are some analogies between this work and interactive debug primitive, the main purpose and evaluation method feel more like an adversarial input detection technique. I am not 100% sure how best to sell to the SE audience. Would it be a better fit for ML venues?

ACM Reference Format:
. 2020. DeepWatch: White-Box Watchpoint for Deep Neural Networks.

1 INTRODUCTION

Deep learning models have their roots spread out to various domains like computer vision, speech recognition, robotics, natural language processing and biomedical domains. Since they have been deployed in such mission critical and sensitive areas, it is imperative that the outputs of such models must be reliable. Many attempts have been made on the verification and training of provably robust networks [5][20][21] and detection of malignant or adversarial input to neural networks.

Grosse et al [8] proposed an extra outlier class in the network only for adversarial examples / anomalies. Gong et al. [6] train an additional binary classifier to separate real and adversarial examples. Metzen et al. [13] detect adversarial examples via training a detection neural network, which takes input from intermediate layers of the classification network to output a probability of the example being adversarial.

In this paper we present a novel method of self diagnosis of adversarial input by borrowing the idea of probing that has been a long standing idea in Software Engineering. This method is analogous to interactive debugging in Software Engineering, more specifically to a watchpoint that gives us a view of the state of the neural network at a particular layer. The "state" here is basically the activation distribution at that layer.

DeepWatch investigates the idea of a probe that is able to self-diagnose adversarial inputs, by inserting a probing layer into an existing model. The key insight is that by adding a layer that encodes the predefined template associated with each class label, we are able to assess the adversarial nature of completely unseen inputs by calculating the deviation from the template.

We empirically demonstrate that adding such white-box probes is feasible, does not add too much performance overhead nor accuracy trade-offs and can generalize to a variety of existing models on well known benchmarks such as MNIST and CIFAR.Our method is also robust to inputs generated by popularly attacks such as PGD, FGSM and CW.

The key research questions we address in the following sections are:

- How does varying the probe properties such as the depth and number of the probe locations affect the ability to detect an adversarial attack?
  We find that although probes deeper into the network have a higher detection performance than those earlier in the network,the performance is best (in terms of AUC score) across all attacks when a combination of all the probes is used than when using a single probe.
- How does degree of perturbation of an image affect our ability to detect adversarial examples?
  We assess that the higher the degree of perturbation of the adversarial input image, the better is the performance of our classifier.
- Can this detection methodology be transferable to aid cross-attack detections?
  Our analysis shows that the probes are indeed model and attack agnostic. They can be used on any pre-trained model and can detect adversarial examples generated by a variety attack.

2 RELATED WORK

Adversarial attacks: A large body of work on iterative white-box attacks [3, 4, 11, 12, 14, 17] have been built upon the technique proposed by Goodfellow et al.[7] using gradient descent based approaches to discover specific perturbations, which when added to an input image can flip the output of a network. Empirical analysis shows that for norm-constrained attacks, PGD (iterative projected gradient descent) [12] is the most effective approach and it reasonably approximates the optimal attack.

Conference to be determined, March 2020,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

To understand the attack methodology, let f_y be the output of the final layer of a neural network classifier. And let $F(x) = \text{argmax}_y f_y(x)$ be the final prediction for an input x. The goal of the adversarial attack algorithm is to find Δx such that $F(x+\Delta x) \neq F(x)$ under the constraint:$\mathcal{S} = \{\Delta x : \|\Delta x\|_p \leq \epsilon\}$. Here we describe three popular attack methods:

Carlini-Wagner attack [3]: minimize $\|\Delta x\|_p$ + c $\mathcal{F}(x + \Delta x)$ such that $x + \Delta x \in \text{dom}_x$. where $\mathcal{F}$ is a cost function such that $\mathcal{F}(x + \Delta x) \leq 0$ if and only if F$(x + \Delta x) = t$ where t is the target class. The hyper-parameter c is chosen using binary search. Possible choices and comparisons between objective functions can be found in [3]. dom_x denotes the data domain constraint eg. $dom_x = [0, 1]^D$.

PGD attack [12]: $x^{t+1} = \prod_{x+\mathcal{S}}(x^t + \alpha \text{sgn}(\nabla_x \mathcal{L}(\theta, x, t)))$ $\prod_{\mathcal{S}}$ is the projection operator onto the set $\mathcal{S}$ defined at the beginning of the section. t is the target label, α is the step size and $\mathcal{L}(\theta, x, t)$ is a suitable loss function.

FGSM attack [7] devised a fast method to compute adversarial examples by applying a one-step perturbation of width ϵ on the image in the direction of highest increase of the linearized cost function under l_∞ norm. $x_{\text{adv}} = x + \epsilon \text{sgn}(\nabla_x \mathbf{J}(x, y_{\text{true}}))$ where x_{adv} denotes the new adversarial example.

Detection methods: [8] argue that adversarial examples lie outside the natural image manifold on which the network is not trained on - under this view, defending against adversarial attacks is an anomaly detection problem. Based on the detection output, we may choose to reject the input or subject it to additional post processing steps in downstream applications. Adversarial perturbation detection techniques typically focus on understanding characteristic patterns within the input features [22] or network activations they produce [1, 2, 8, 13, 19]. Our work is closely related to the latter set of techniques which operate on features extracted from intermediate layers. Aigrain et al. [1] detect misclassifications by training a binary network on the logits of a simple 3 layer pretrained network. Despite good results, it is unclear whether their approach is viable for deeper networks over many layers. Additionally, our approach differs from theirs as we add probes throughout the network at multiple positions to consolidate individual results. Metzen et al. [13] use a similar approach by adding a detector sub-network to classify intermediate features. A drawback of these techniques is that the detector networks are black boxes with no transparency about the decision making procedure. Our method, in contrast, calculates the KL divergence of activation patterns with expected patterns at each layer which helps us understand how each successive layer is fooled as we go deeper into the network.

A recent approach by Yin et al. [23] involves multiple class detectors to detect and correctly classify adversarial examples using asymmetrical adversarial training (AAAT) which is formulated like a mini-max problem. While their method also achieves good results, their detectors are trained adversarially which increases training time, reduces generalization [18], and reduces flexibility for more general tasks like anomaly detection. Pang et al. [16] introduced yet another detection based defence that uses K-density estimation (a Cross Entropy training coupled with a K-density detector). Their methodology includes a novel training procedure that introduced reverse cross entropy (RCE) and a thresholding test strategy implemented by the kernel density (Kdensity) detector. Our method differs from them in the sense that they introduce a new objective to train the entire network i.e their method cannot be used on a given pretrained network without changing its parameters which might induce problems in the original model.

Our final model architecture is perhaps closest to Mustafa et al.[15] where they add multiple probes emanating from the intermediate layers of the network and use their activation values as part of the final objective function. They focus on class-wise disentanglement of intermediate representations which is similar to the template coded comparison introduced in this work. Apart from the KL divergence based class-wise separation used for downstream tasks, our approach differs in that we focus on adversarial example detection rather than a defense based on increasing robustness and that we preserve the parameters of the original network.

3 APPROACH

3.1 Probe and Template

To create a new classifier trained to classify adversarial examples, we have inserted linear layers as probe layers or probes into the pre-trained networks that have been trained on the MNIST and CIFAR-10 datasets.

Probe Input: The set of activations of the layer at the probe insertion location.

Probe output: The set of activations (denoted by $\mathcal{H}$) which are generated by the probe. $|\mathcal{H}|$ is the size of the probe output which will compared with the *template*.

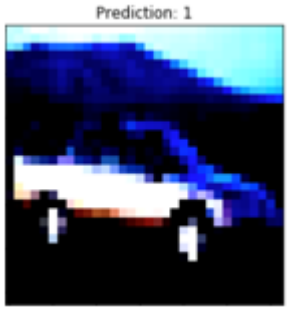
We define the *template* as the ideal or the expected activation pattern for the given input example which is used to encode class information for that particular example. A ***template*** $\mathcal{T}_k$ is a function on an arbitrary set of neurons $\mathcal{A}_k$ for a particular class K:

$$\mathcal{T}_k[i] = \begin{cases} 1, & \text{if } i \in \mathcal{A}_k \\ 0, & \text{otherwise} \end{cases}$$

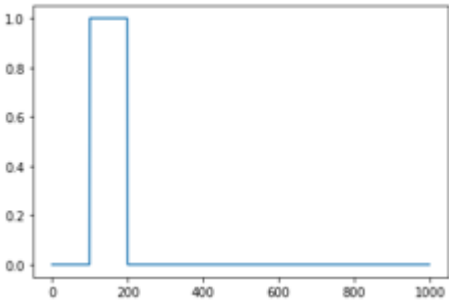
Where $\mathcal{T}_k[i]$ denotes the i^{th} element in the *template*. As an example, consider the figure «»(add number) below. The top row includes a non-adversarial picture of a car which has the class number 1. The network correctly predicts the class and we arbitrarily choose the neurons from 100 to 200 to be activated i.e they have the value 1 while the others have the value 0. The second row of images shows the same image with an adversarial perturbation which the network incorrectly predicts as the class 0 (Airplane). The template then encodes the class information based on the network's prediction. Note here that while the image doesn't change too much, the activation patterns are very different and we can leverage this information by comparing them with the created templates.

3.2 Novel neuron coverage metric

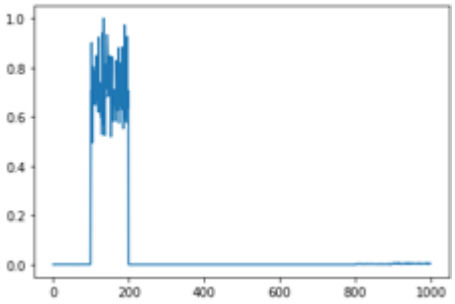
To identify any possible patterns in activations and systematically measure the parts of a deep learning system exercised by different test inputs, the related works that we have referenced have defined a neuron coverage metric as the ratio of the number of unique activated neurons for all test inputs and the total number of neurons in the DNN. This was done by measuring every single neuron's



(a) Benign input image of a car belonging to class 1

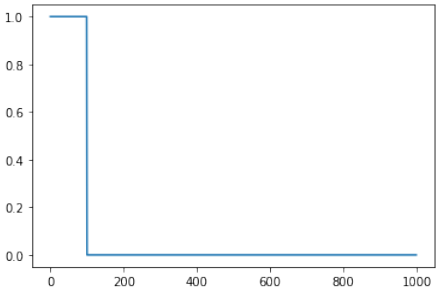


(b) Ideal activation distribution for class 1

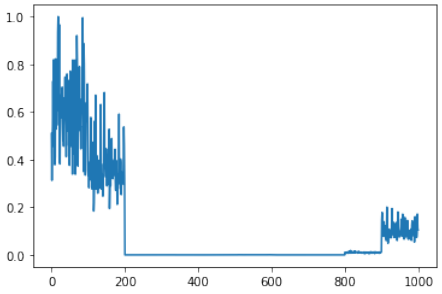


(c) Activations obtained from the hidden layer of the trained model

(d) Adversarial input image of a car



(e) Ideal activation distribution for class 0



(f) Activations obtained from the hidden layer of the trained model

Figure 1: As an example, for an input image belonging to class 1, the ideal distribution should have activated neurons in the 100-200 range of the hidden layer. When we compare the benign activation distribution we obtain from the model and the ideal distribution, the KL divergence should be low and for an input image adversarially assigned class 0, the ideal distribution should have activated neurons in the 0-100 range of the hidden layer. But when we compare the adversarial class activation distribution with the ideal distribution, the KL divergence should be high.

activation and if its output is higher than a certain threshold, it is considered as activated. In our endeavour to get more insight into activation patterns, we consider the entire distribution of activations rather than targeting each neuron individually. Additionally, we apply constraints on neuron coverage during the training process, rather than inferring patterns after the completion of training. We essentially define a new neuron coverage metric based on KL Divergence. It helps create an activation pattern template for each class.

3.3 Methodology

Figure 2 outlines our methodology. Each probe comprises of a hidden linear layer which has a fixed number of neurons. If the dataset has images in c classes, a template activation distribution pattern is decided for each class. A simple way to do this is to choose the hidden layer size to be c x 100. Then, each consecutive 100 neurons could be dedicated to one class. So for the first class, the template would consist of neurons 0-100 with an activation of 1 and the rest of the neurons would not be activated.

First, the probes are inserted at n different locations after each block in the pre-trained network. Then, we create a *probing network* by freezing all but the trainable probe locations. We train the probes using the same dataset that the pre-trained model was trained on using a modified loss function. The output of each of the probes is an activation distribution.

Modified loss function. The idea of looking for anomalies in activation distribution pattern demands a robust distance metric to find the difference between the actual activation pattern and what is expected. Such a metric that satisfied our needs was KL Divergence, borrowed from information theory (add ref). Our new loss function to train the probe is:

$$CrossEntropy(f(x), y) + \lambda * KL(a(x), t_y) \quad (1)$$

a is the activation pattern which we obtained by passing through the network and f is our neural network and t is the target class that is correctly classified. The hyperparameter λ decides the influence of the KL divergence term on the loss. For our experiments, we set λ to 0.1. In other words,

$$\mathcal{KL}(a(x), t_y) = \sum_{i=1}^{|h|} a(x)_i \ln \frac{a(x)_i}{(t_y)_i} \quad (2)$$

where $a(x)$ denotes the activation pattern in the final layer for the input x and t_y is the corresponding template for the true label y. It is to be noted by the reader that the Cross Entropy function

Layer #	FCNet - 10	RESNET-110
1	Linear(784, 698) + ReLU + Dropout(0.2) + Linear(698, 612) + ReLU Probe 1 = Linear(612, 100)	Conv(3, 16, 3 ×3) + BN Probe 1 = Linear(32 ×32 × 16, 5000) + $ReLU$ + $Linear$(5000, 1000)
2	Dropout(0.2) + Linear(612, 526) + ReLU + Dropout(0.2) + Linear(526,440) + ReLU Probe 2 = Linear(440, 100)	$\begin{bmatrix} Conv(16, 16, 3 \times 3) + BN \\ Conv(32, 3 \times 3) + BN \end{bmatrix} \times 18$ Probe 2 = Linear(32 ×32 × 16, 5000) + ReLU + Linear(5000, 1000)
3	Dropout(0.2) + Linear(440, 354) + ReLU + Dropout(0.2) + Linear(354, 268) + ReLU Probe 3 = Linear(268, 100)	$\begin{bmatrix} Conv(16, 32, 3 \times 3) + BN \\ Conv(64, 3 \times 3) + BN \end{bmatrix} \times 18$ Probe 3 = Linear(32 ×32 × 16, 5000) + ReLU + Linear(5000, 1000)
4	Dropout(0.2) + Linear(268, 182) + ReLU + Dropout(0.2) + Linear(182, 96) + ReLU + Dropout(0.2) + Linear(96, 20) + ReLU Probe 4 = Linear(20,100)	$\begin{bmatrix} Conv(32, 64, 3 \times 3) + BN \\ Conv(64, 64, 3 \times 3) + BN \end{bmatrix} \times 18$ Probe 4 = Linear(32 ×32 × 16, 5000) + ReLU + Linear(5000, 1000)
5	Dropout(0.2) + Linear(20,10)	FC(64,10)

Table 1: FCNet-10 and ResNet 110 architecture

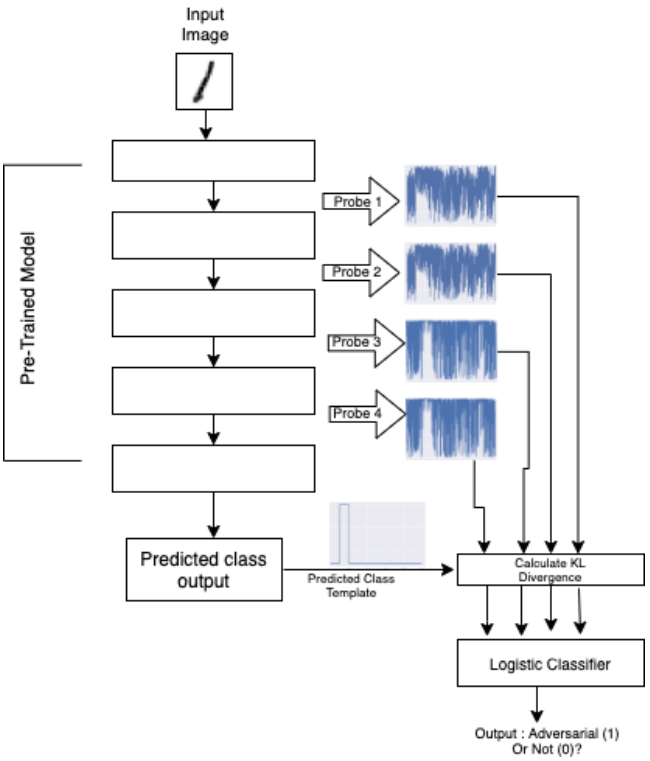


Figure 2: Outline of the methodology

in Eq. (1) can be replaced with any cost function that the original network was trained on.

Third, a logistic classifier is trained on the minimised KL divergence between the template of the ground truth class and the activation distribution of each probe to classify each image along a score distribution as adversarial (1) or non-adversarial (0). Therefore, for n probes there would be n KL divergence values that act as features for the logistic classifier. A score closer to 0 would mean non-adversarial and closer to 1 would be adversarial.

Therefore our probing system works as follows : When a test image is passed to the network, each probe generates an activation distribution. This is fed into the logistic classifier which outputs a confidence score for each image.

Intuitively, this objective incorporates the following attractive properties: (i) It increases the inter-class distance by mapping data points of a particular class to a subspace of lower dimensionality; (ii) It encourages features of a particular class type to be close to each other decreasing intra-class distance.

4 EVALUATION AND RESULTS

In this section, we experiment with the probe properties to test our method's abilities to identify adversarial examples generated using various attacks such as PGD, FGSM, CW on the MNIST and CIFAR-10 datasets.

In order to generate adversarial input examples that are visually similar to the original examples but still fool the network, several attacks have been developed. Here are some popular attacking methods that we used in our evaluations:

Fast Gradient Sign Method (FGSM):. Goodfellow et al. [12] presented an one-step attacking method to create an adversarial example with the perturbation ϵ and the training loss L(x, y). It essentially generates adversarial examples by maximising the loss with respect to the input image.

Projected Gradient Descent (PGD). This is a white box attack and also goes by I-FGSM since it is an iterative version of FGSM.

Carlini and Wagner (CW attack): Carlini and Wagner [2] introduce an optimization-based method, which is one of the most powerful attacks.

4.1 Experimental Set Up

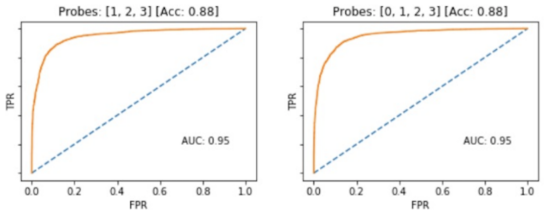
We adopted the MNIST dataset, a database of handwritten digits that has a training set of 60,000 examples, and a test set of 10,000 examples.[10] We trained a FCNET-10 model on one complete iteration of the dataset. We also adopted the CIFAR-10 dataset [9] consisting of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 40000 training images and 40000 test images. We trained a ResNet-110 neural network on this dataset over 10 epochs with 10 output neurons, one for each class again. The probes are also trained on 40,000 CIFAR-10 and 60,000 MNIST images using a modified loss function. The hidden linear layer in each probe has 1000 neurons, with each of the 10 classes being represented by 100 neurons. The probing network outputs a 1000 neuron activation for adversarial example detection. This means that for each class, the ideal activation distribution would peak at the input's chosen group of 100 neurons, and the other neuron activations would be 0. Once the network has been trained in this way, any benign input image should produce an activation distribution with a low KL divergence to the template of the ground truth class.The modified loss function reduces the KL divergence between the activation pattern of the probes and that of the template.

Pre-trained Models Architecture for ResNet-110 and FCNET-10. Table 1 shows the breakdown of the layers in the models we have used. As indicated, we applied the ReLU activation and have inserted probes (Probe 1,2,3,4) with two linear layers after every block in the RESNET model. And for FCNET-10 being a smaller network, we inserted probes with a single linear layer between each layer. The output of each of the now probed FCNET-10 and RESNET-110 networks is that of the ten identifiable classes.

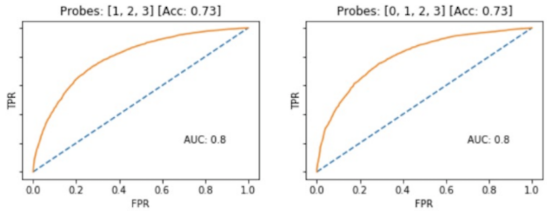
4.2 RQ1: How does varying the depth of the probe location, the number of epochs and freezing training on previous layers affect the ability to detect an adversarial attack?

Our main focus for this research question is to experiment how inserting a 'probe' layer of neurons at different blocks of the pre-trained network affects adversarial example detection performance. A probe inserted after block-1 is referred to as probe location 1, a probe inserted after block-2 of the pre-trained network is probe location 2 and so on.

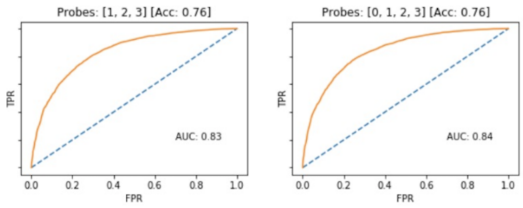
4.2.1 Detection performance. Table 2 shows the results of our approach when tested our approach as described on MNIST and CIFAR-10 datasets with Resnet-110 and FCNET-10. We used different metrics to calculate the performance of our detection technique. Accuracy is the percentage of correctly classified adversarial examples. AUROC (Area under ROC) measures separability of classes or in other words, how well our detector is able to detect adversarial examples with a high confidence. This is measured by calculating



(a) PGD



(b) FGSM



(c) CW

Figure 3: Best detection performance when the probe is inserted at different locations for (a)PGD, (b)FGSM and (c) CW attacks on RESNET-110 and CIFAR-10- a combination of probes provides a better detection accuracy as indicated by high AUC scores

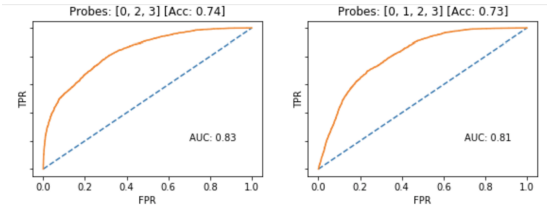
the area under an ROC curve where the x-axis is the False Positive Rate (FPR) and y-axis is True Positive Rate(TPR). Another metric is the KL divergence measured between the probe activations and their respective templates.

Our experiments with the probe properties yielded the results for each combination of best probe locations under various attacks as shown in Figure 3 for RESNET-110 trained on CIFAR-10, and Figure 4 for a fully connected 10 layer network (FCNET-10) trained on MNIST. It shows the resulting ROC curves for adversarial example detection performance when the probe is inserted at different locations. For instance, we obtained a high AUC of 0.95 on the PGD attack examples performed on RESNET-110 with the hyperparameters epsilon and alpha set to 0.1 and 0.001.

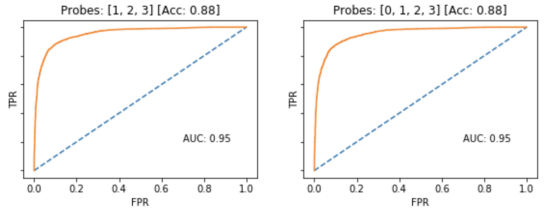
We also obtained similar results on MNIST (AUC=0.95 with epsilon=0.1) with FGSM attack examples on FCNET-10. Figures 4show further results. From the AUCs obtained, we can conclude that using multiple probes produces a better detection accuracy than using them individually.

	FCNET (Trained with MNIST)						RESNET-110 (Trained with CIFAR-10					
Attacks	PGD		FGSM		CW		PGD		FGSM		CW	
No. of probes	Probe Set	AUC	Probe Set	AUC	Probe Set	AUC	Probe Set	AUC	Probe Set	AUC	Probe Set	AUC
1 probe	[3]	0.83	[3]	0.95	[1]	0.94	[3]	0.93	[2]	0.69	[2]	0.74
2 probes	[0,3]	0.83	[1,3]	0.94	[1,3]	0.99	[1,3]	0.94	[1,3]	0.77	[1,3]	0.8
3 probes	[0,2,3]	0.83	[1,2,3]	0.95	[1,2,3]	0.999	[1,2,3]	0.95	[1,2,3]	0.8	[1,2,3]	0.83
4 probes	[0,1,2,3]	0.81	[0,12,3]	0.95	[0,1,2,3]	0.999	[0,1,2,3]	0.95	[0,1,2,3]	0.8	[0,1,2,3]	0.84

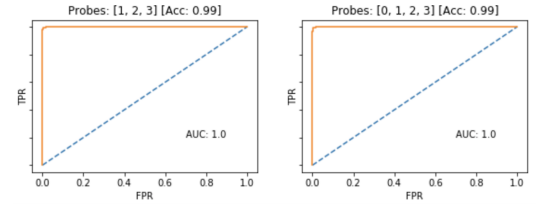
Table 2: Best AUCs for 1, 2 and 3 probe combinations over multiple attacks



(a) PGD



(b) FGSM



(c) CW

Figure 4: Best detection performance when the probe is inserted at different locations for (a)PGD, (b)FGSM and (c) CW attacks on FCNET-10 and MNIST - a combination of probes provides a better detection accuracy as indicated by high AUC scores.

H1.Best number of probes and their probe locations Although probes deeper into the network give a better detection performance than those earlier in the network, the AUCs show better accuracy across all attacks when a combination of all the probes is used than when using individual probes.

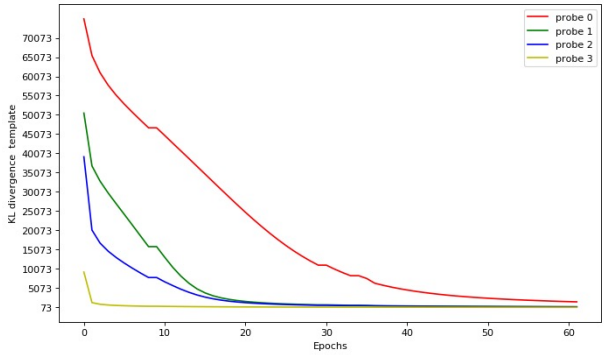
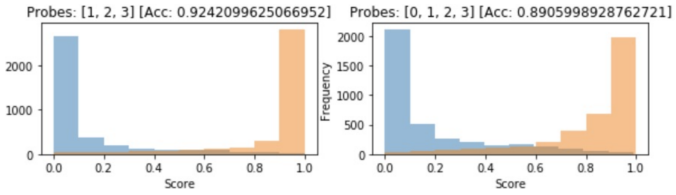


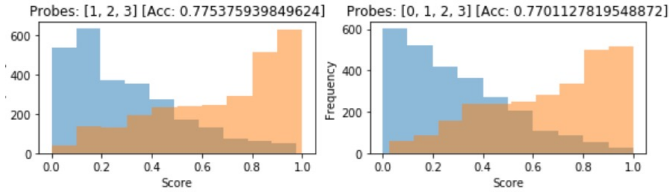
Figure 5: KL Divergence between probes and the templates at each probe location

4.2.2 Detection performance and KL Divergence. The results of detection accuracy performance can also be visualised by computing the KL Divergence between the output activation patterns of each probe and their original template activations. Figure 5 is a graph plotting the KL divergence between the activations of each probe location and the template versus the number of epochs the probe is trained for. Each probe uses an SGD optimizer with a learning rate of 1e-3 trained over 60 epochs. It can be inferred that the higher the probe location is, i.e, the earlier in the network the probe is inserted into, the less refined the features learned are and so the KL at initial probe locations is higher with a greater change after each iteration of probe training. On the other hand, the further down the network the probe is inserted, the more refined the features are and the lower is the KL divergence. The elbow in the plot towards epoch 25 indicates the threshold for number of epochs to sufficiently train the probed network. Any fewer epochs and the network suffers from greater KL Divergence differences and thereby poorer detection by the logistic classifier.

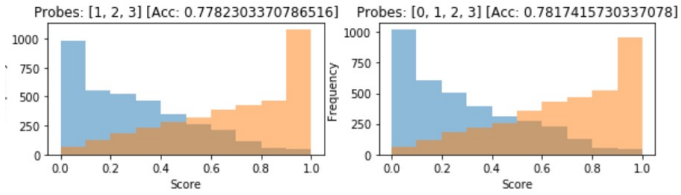
H2.Relationship between Probe location Depth and KL Divergence The earlier in the pre-trained network the probe is inserted, the less refined are the features to learn resulting in higher KL divergence. The later/ deeper down the pre-trained network the probe is inserted, the more refined are the features to learn resulting in lower KL divergence.



(a) PGD



(b) FGSM



(c) CW

Figure 6: Logistic classifier when the probe is inserted at different locations for (a)PGD, (b)FGSM and (c) CW attacks on RESNET-110 and CIFAR-10 - a combination of probes provides higher separabiility as indicated by the confidence scores

H3. Amount of training versus probe location An epoch threshold of 25 has been experimentally observed. Any fewer epochs and the network suffers from greater KL Divergence differences and results in poorer detection by the logistic classifier.

Figure 6 indicates the performance of the logistic classifier in detecting adversarial and non adversarial examples under varying probe location conditions and attacks. The histograms in the figure show confidence scores on the adversarial and non-adversarial examples dataset when the probe is inserted after corresponding blocks in the architecture we described. The histogram bins the number of adversarial and non-adversarial examples against a confidence score. It shows the confidence with which we are able to detect the nature of the examples- adversarial or not. A high score would mean that they've been identified as adversarial examples while a low score would indicate that they are benign. A high separability and less overlap in the figure indicates a better overall classifier with a reliable score.

Attack	Epsilon (ϵ)	ResNet 110 Accuracy	AUC score
PGD	0.02	14.9	0.82
	0.03	10.2	0.85
	0.1	0.26	0.95
	0.3	0.0	0.97
FGSM	0.015	26.8	0.81
	0.06	17.19	0.89
	0.125	12.41	0.92
	0.25	10.28	0.95
	0.5	10	0.96

Table 3: Effect of varying perturbation (ϵ) of image using PGD and FGSM attacks- increasing trend of detection accuracy with larger ϵ

While it is clear that the examples are being correctly detected as benign or adversarial as shown by the 0 to 1 score scale, there is no single probe location that stands out with a better detection than the rest. A probe that is placed higher or lower in the network has little to do with it being inherently better at detecting adversarial examples. So, placing probes at different locations in the network and using all of their outputs as features to the logistic classifier will give the best result.

4.3 RQ2: How does degree of perturbation of an image affect our ability to detect adversarial examples?

To evaluate the robustness of our classifier, we tinkered with the amount of distortion that an input adversarial example is subjected to. Degree of perturbation or distortion, (ϵ) as the name suggests, is the degree to which an image has been changed from its original state. The more we change an image, the more the effects of the change on the network will be. Therefore, we expected that a higher degree of perturbation will result in a more robust and accurate detection of adversarial examples by our probing network, i.e. a higher AUC score. But this will also result in poorer accuracy of classification by the pre-trained model.

We conducted experiments to find out how the classifier accuracy varies with an increase in the degree of perturbation(ϵ) of the original image. Table 3 shows the results of these experiments for RESNET-110 using PGD and FGSM attacks. In general, adversarial examples that have been formed from minimal perturbations(smaller ϵ) have a smaller KL divergence from the template and are harder to detect by the classifier as indicated by the low AUCs . On the other hand, the more perturbed the adversarial image is(larger ϵ), the higher is the KL divergence between the input and the template and the easier it becomes for the classifier to detect adversarial examples (as indicated by the higher AUCs)

Trained on / Performance on	FGSM	CW	PGD
PGD	0.78	0.82	0.95
CW	0.82	0.85	0.72
FGSM	0.81	0.8	0.65

Table 4: Good transferability of our probing network on RESNET-110 across diverse attacks

H4: Effect of degree of image perturbation Larger peturbations make it easier for the classifier to detect adversarial input while smaller ones are harder to detect.

4.4 RQ3: Can this detection methodology be transferable to aid cross- attack detections?

There are several new kinds of attacks that are being created every day. We aimed to develop a detection mechanism that would be robust enough against all of these cross-attacks. We evaluate how our probing network trained on one kind of popular attack fares when subjected to examples from a different kind of attack.

In order to do this, we have trained the logistic classifier on one attack and evaluated the detection performance by computing the AUC for another attack. The high AUC results as shown in 4for these cross experiments indicates that our probing network mechanism is promising. A note to be made is that these AUC scores are the best values achieved for all combinations of probe locations.

H5: Transferability of probing The probes are model and attack agnostic. They can be applied to any pre-trained network to obtain high detection accuracy across an array of attacks.

4.5 RQ4: Is the detection methodology dependent on the size of the hidden layer?

In order to reduce the brunt of hyper-parameter tuning, we performed this experiment to determine if the number of neurons in the hidden layer can cause any change in the detection accuracy of the probing network. We then try and obtain the best hidden layer size that aids in better detection.

The current hidden layer size we have chosen for the probing network is a 1000 neurons. We experimented with different hidden sizes as tabulated in 5 on PGD attack with 4000 test examples. It is surprising to note that there is not too much difference with change in the hidden size. Smaller hidden layers achieve around 0.95 of detection AUC while very large hidden layers suffer slightly with around 0.93 AUC. So, we decided to adhere to smaller sizes in order to reduce parameters introduced.

Hidden Layer size	AUC Score
100	0.948710
500	0.959034
900	0.954129
1300	0.950594
1700	0.948836
2100	0.9458128
2500	0.9404924
2900	0.938910
3300	0.936057
3700	0.932921

Table 5: Effect of Hidden Layer size of Probing network on detection AUC- smaller sizes yield better accuracy

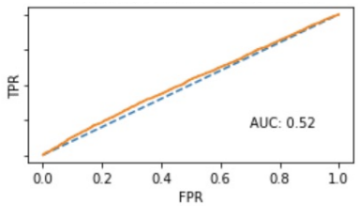


Figure 7: Poor detection performance indicated by poor AUCs when the logistic classifier is trained directly on activations in a middle layer

H6: Dependence on Hidden size Although the hidden layer size has negligible effect, it is experimentally preferable to have smaller hidden layer sizes to avoid suffering with slightly poorer AUCs.

5 DISCUSSION AND FUTURE WORK

Detection performance without probes. As an experiment, we also trained the logistic classifier directly on the activations of a middle layer. We used the activations of 4000 probes directly without training the classifier on the KL divergence between the probe and template activations. Figure 7 shows a low AUC of 0.5 indicative of poor performance as compared to the probed network approach. And figure 8 further illustrates the poor performance of the classifier for adversarial and non-adversarial examples. Both kinds of examples have similar score distributions without probes being used for detection whereas the distribution of confidence scores is more defined and closer to 0 and 1 for non-adversarial and adversarial examples respectively with probes inserted. This further validates our research findings that the logistic classifier yields better detection accuracy by having a base template for the original example to compute KL divergence against the probe activations than being directly trained on the middle layers' activations.

As future work, we would like to explore how we can further improve the robustness of the probing network by re-feeding detected adversarial examples. It would confirm our hypothesis that by retraining the model with invalid inputs, we improve model's

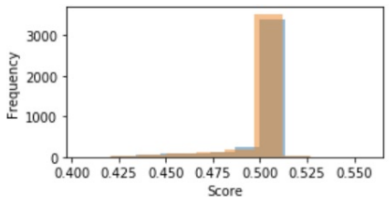


Figure 8: Poor classification of adversarial examples indicated by weak confidence scores when the logistic classifier is trained directly on activations

attack accuracy. It also would be interesting to note the model's performance on out of distribution examples, not drawn from the same adversarial data.

6 CONCLUSION

In this paper, we develop a novel self-diagnostic method to detect adversarial examples by drawing the analogy of probing and inserting a probe layers into an existing model. The key insight is that we are able to assess the adversarial nature of completely unseen inputs by calculating the deviation or KL divergence between the probe layers and the template. We demonstrated that adding such white-box probes is feasible, does not add too much performance overhead nor accuracy trade-offs. Moreover, we can generalize it to a variety of existing models on well known attacks and datasets. We experimentally addressed some factors that might affect the detection performance of our methodology such as the location and combination of probes and the degree of perturbation. We concluded that although probes deeper into the network have a higher detection performance than those earlier in the network,the performance is best (with AUCs as high as 0.95) across all attacks when a combination of all the probes is used than when using a single probe. We also assessed that we obtain better classifier performance with AUCs as high as 0.97 for higher degrees of adversarial input image perturbation(0.3). Finally, we demonstrated that our probing network is indeed model and attack agnostic with high accuracies across all of them.

REFERENCES

[1] Jonathan Aigrain and Marcin Detyniecki. 2019. Detecting Adversarial Examples and Other Misclassifications in Neural Networks by Introspection. *arXiv preprint arXiv:1905.09186* (2019).
[2] Nicholas Carlini and David Wagner. 2017. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. 3–14.
[3] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*. IEEE, 39–57.
[4] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. 2018. Ead: elastic-net attacks to deep neural networks via adversarial examples. In *Thirty-second AAAI conference on artificial intelligence*.
[5] Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. 2018. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265* (2018).
[6] Zhitao Gong, Wenlu Wang, and Wei-Shinn Ku. 2017. Adversarial and clean data are not twins. *arXiv preprint arXiv:1704.04960* (2017).
[7] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).
[8] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. 2017. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280* (2017).
[9] https://www.cs.toronto.edu/ kriz/cifar.html. 2006.
[10] https://yann.lecun.com/exdb/mnist/. 2006.
[11] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236* (2016).
[12] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards Deep Learning Models Resistant to Adversarial Attacks. *ArXiv* abs/1706.06083 (2017).
[13] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. 2017. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267* (2017).
[14] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2574–2582.
[15] Aamir Mustafa, Salman Khan, Munawar Hayat, Roland Goecke, Jianbing Shen, and Ling Shao. 2019. Adversarial defense by restricting the hidden space of deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 3385–3394.
[16] Tianyu Pang, Chao Du, Yinpeng Dong, and Jun Zhu. 2018. Towards robust detection of adversarial examples. In *Advances in Neural Information Processing Systems*. 4579–4589.
[17] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 372–387.
[18] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. 2018. Adversarially robust generalization requires more data. In *Advances in Neural Information Processing Systems*. 5014–5026.
[19] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. 2017. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766* (2017).
[20] Robert Stanforth, Sven Gowal, Timothy Mann, Pushmeet Kohli, et al. 2018. A Dual Approach to Scalable Verification of Deep Networks. *arXiv preprint arXiv:1803.06567* (2018).
[21] Eric Wong and J Zico Kolter. 2017. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851* (2017).
[22] Weilin Xu, David Evans, and Yanjun Qi. 2017. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155* (2017).
[23] Xuwang Yin, Soheil Kolouri, and Gustavo K Rohde. 2020. Adversarial Example Detection and Classification with Asymmetrical Adversarial Training. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SJeQEp4YDH