Innocent Until Proven Guilty (IUPG): Building Deep Learning Models with Embedded Robustness to Out-Of-Distribution Content

Brody Kutt, William Hewlett, Oleksii Starov, Yuchen Zhou
AI Research
Palo Alto Networks
Santa Clara, California

***Abstract*—Deep Neural Network classifiers trained with the conventional Categorical Cross-Entropy loss face problems in real-world environments such as a tendency to produce overly confident posterior distributions on out-of-distribution inputs, sensitivity to adversarial noise, and lost performance due to distributional shift. We hypothesize that a central shortcoming – an inability to effectively process out-of-distribution content within inputs – exacerbates each of these setbacks. In response, we propose a novel learning framework called Innocent Until Proven Guilty which prototypes training data clusters or classes within the input space while uniquely leveraging noise and inherently random classes to discover noise-resistant, uniquely identifiable features of the modeled classes. In evaluation, we leverage both academic computer vision datasets and real-world JavaScript and URL datasets for malware classification. Across these interdisciplinary settings, we observe favorable classification performance on test data, decreased loss of performance due to recency bias, decreased false-positive responses on noise samples, and decreased vulnerability in several noise-based attack simulations when compared to a baseline network of equal topology trained with Categorical Cross-Entropy. To the best of our knowledge, ours is the first work that demonstrates significantly decreased vulnerability to blackbox append attacks on malware. By applying the well-known Fast-Gradient Sign Method, we show the potential to combine our framework with existing adversarial learning techniques and discover favorable performance by a significant margin. Our framework is general enough for use with any network topology that could otherwise be trained with Categorical Cross-Entropy.**

***Keywords*-Deep Learning, Out-Of-Distribution Robustness, Prototype Learning, Benign Append Attack, Malicious Injection, Distributional Shift, Recency Bias, Malware Classification, Web Security**

I. Introduction

Categorical Cross-Entropy (CCE) loss is a standard supervised loss function used to train a variety of Deep Neural Network (DNN) classifiers. CCE produces a purely discriminative model with no embedded means to infer out-of-distribution (OOD) content or effectively utilize classes that do not possess uniquely identifiable structure. The Innocent Until Proven Guilty (IUPG) framework provides alternative architectural components and a hybrid discriminative and generative loss function for training DNNs to classify mutually exclusive classes. IUPG includes learning a library of inputs within the original input space that – together with the network – prototype uniquely identifiable subsets of the input space. The network learns to map the input space to an output vector space in which prototypes and members of the relevant input subset map exclusively to a common point. The distances between noise (or any class of data lacking a prototypical description) and *all* prototypes in the output vector space are maximized in training. We call any such classes "off-target" while *target* classes have one or more assigned prototypes. Off-target data helps to chisel down the extracted features of target classes to that which is truly class-exclusive as opposed to coincidental. In the context of malware classification, this equates to learning inseparable features of malware clusters that define their maliciousness while ignoring benign content. During inference, each sample is scanned for these inseparable qualities while ignoring all structure outside the class, hence the technique assumes each sample is "innocent until proven guilty". Increasing the specificity of learned features intuitively increases the network's resistance to any OOD content. As the central hypothesis of this work: *we propose that this is chiefly responsible for the desirable effects we explore.*

In evaluation, we use an equivalent network trained with CCE loss to measure baseline performance – referred to as an IUPG network's CCE counterpart. We (1) explore the test set classification performance of IUPG and its CCE counterpart across various cybersecurity and computer vision settings including different usages of noise; (2) compare the tendency to produce false-positive (FP) responses on OOD inputs; (3) compare the impact of recency bias (performance loss due to distributional shift) on classification accuracy; (4) compare both frameworks' resistance to blackbox append attacks; (5) demonstrate the applicability of existing adversarial training techniques to IUPG. The rest of this work is organized as follows: Section II discusses related work on relevant themes, Section III defines the IUPG framework, Section IV details the experiments we used to evaluate IUPG, and Section V provides a summary and some final remarks.

II. Background

Prototype-Based Learning: Among the earliest works on prototype-based learning is learning vector quantization (LVQ) [12] which can be thought of as a prototype-based

© 2021, Brody Kutt. Under license to IEEE.
DOI 10.1109/SPW53761.2021.00016

k-nearest neighbors algorithm. In the taxonomy of LVQ variants presented in [17], this work is comparable to GLVQ [21] which falls under margin maximization of the data space with Euclidean distance. IUPG, however, combines prototype learning with DNNs and makes unique use of off-target samples. Common goals of prototype-based learning in DNNs include low-shot learning [16] and verdict interpretability [15]. The model in [30] shares several similarities with IUPG. However, prototypes in [30] are defined in the output vector space of the model. These prototypes are not human-interpretable and have no intuitive initialization. Critically, when prototypes are defined this way, we observed frequent convergence to solutions in which multiple prototypes merge to a common point. This is supported by results in [30] which report similar or worse performance with multiple prototypes per class. We also could not find similar work that utilizes off-target samples as IUPG does. This ability in particular allowed us to discover considerable benefits on problems such as malware classification where only one class possesses a uniquely identifiable structure.

Append Attacks: Append attacks are concatenations of adversarial content to inputs with the intent to perturb the classification result [28]. This is of particular relevance to malware classification where benign noise can be appended onto malware to fool the classifier despite the malicious activity remaining intact. Inversely, malicious content can be injected into large benign files to evade detection. In what is known as whitebox attacks [28], the appended noise can be crafted while exploiting model details. In general, a blackbox adversarial attack assumes no knowledge of the model and is often the only attack possible against proprietary defenses. This work provides evidence that even the simplest append attack varieties can pose a serious threat to highly accurate models. To the best of our knowledge, previous work in deep learning lacks a generic solution for append attacks on malware.

Out-of-Distribution (OOD) Classification: It is well understood that DNNs (save some specialized highly non-linear options such as RBF networks) trained with CCE are prone to produce highly overconfident posterior distributions for OOD inputs [22]. Reliably handling OOD content is a critical requirement for real-world systems. Orthogonally to our work, open-world frameworks often equip models with external detectors that aim to identify and discard OOD inputs [4]. Other work relies on learning an external rejection function either concurrently or after training of the classification network [6]. This work demonstrates an embedded adeptness to handle OOD content resulting from IUPG alone.

Combining Strengths: In general, any of the techniques developed for CCE-trained DNNs to overcome whitebox attacks in recent years, such as the many varieties of adversarial training [28], can be equivalently applied to IUPG-trained networks. IUPG loss can be used as a drop-in replacement for CCE within these special training procedures. An example of this is provided in Section IV-D wherein we find consistently greater rates of success with IUPG. Similarly, we suggest the combination of IUPG with external OOD detectors is likely to outperform either in isolation.

III. IUPG Framework

IUPG networks are encoders that map inputs and prototypes from a common input space into an output vector space. The novel components of IUPG exist at the input and output layers of a DNN along with a special loss function. All hidden layer details – including the number of layers of different functional types organized into any topology – can vary as needed based on problem relevance. Consider a network, $\mathcal{N}\colon \mathbb{I} \rightarrow \mathbb{R}^z$, which maps the set of vectorized inputs $\mathbb{I}$ to vectors in $\mathbb{R}^z$. Conventional CCE training of $\mathcal{N}$ includes mapping $\mathcal{N}(\mathbb{I})$ to vectors in $\mathbb{R}^c$ (where c is the number of classes). We will explain the novel components of IUPG by augmenting them onto the abstract network architecture $\mathcal{N}$. Refer to Figure 1 for an illustration of the IUPG components.

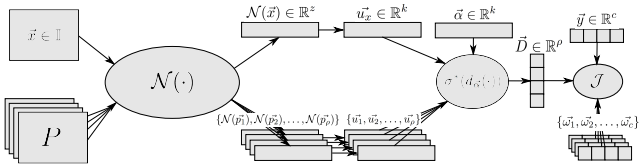

Figure 1: IUPG components augmented on the abstract network $\mathcal{N}$.

Data Guidelines: For CCE training with c classes, it is necessary and sufficient to acquire labeled examples of all c classes. While IUPG can be trained with these datasets, we find it is often useful to include off-target samples. For off-target samples $(\vec{x}, \vec{y})$ we define $\vec{y} = \vec{0}$. How to decide what data or class is "off-target" is problem-dependent but intuitive. If training a "cat" or "not-cat" classifier, there is only one class with a uniquely identifiable structure. "Not-cat" does not possess a prototypical description. Any learned indicators of "not-cat" are likely just facets of the circumstantial training data. "Not-cat" should be defined as the off-target class while "cat" is assigned one or more prototypes. Alternatively, if training a "cat" or "dog" classifier, both classes possess uniquely identifiable structure and an off-target class should be *augmented* to these classes. Statistical noise is often easy to synthesize for off-target data.

Prototypes: IUPG networks process an input and a library of ρ prototypes, $P = \{\vec{p_1}, \ldots, \vec{p_\rho}\}$ prototypes in a Siamese [3] fashion. P is pictured as input to $\mathcal{N}$ adjacent to $\vec{x}$ in Figure 1. Each $\vec{p}$ constitutes learnable weights of the network. Each $\vec{p}$ learns prototypical information about a subset of training data such that all its members will exclusively map close to $\vec{p}$ after processing with $\mathcal{N}$. Subsets

can be learned automatically, specified with class labels, or both. One way to define each $\vec{p}$ is as an element of $\mathbb{I}$ itself. If elements of $\mathbb{I}$ are prohibitively large or a large ρ is desired, a memory-efficient definition of $\vec{p}$ is as the weight vector of a linear combination of training inputs. Concretely, we designate static training samples to form a basis set B. The elements of B are chosen through clustering techniques to span the training distribution. We then define each $\vec{p} \in \mathbb{R}^{|B|}$. Before processing with $\mathcal{N}$, we compute the dot product $\texttt{softmax}(\vec{p}) \cdot B$. Under both prototype definition varieties, the choice for ρ can be guided by domain knowledge or discovered through hyperparameter optimization techniques. Domain knowledge can guide the initialization of each $\vec{p}$. E.g., one may wish to establish cluster center-points as initializations that may correspond to semantically meaningful divisions of a class. We found cluster-based initializations to significantly reduce required training time to convergence.

Distance Function: Vectors in $\mathcal{N}(\mathbb{I})$ are mapped to an output vector space $\mathbb{U} \subset \mathbb{R}^k$ via a fully connected layer. k need not be equal to the number of classes. The intermediate representation of $\vec{x}$ in $\mathbb{U}$ is denoted $\vec{u_x}$ while the representation of P is denoted $U_p = \{\vec{u_1}, \vec{u_2}, \ldots, \vec{u_\rho}\}$, depicted in Figure 1. Measuring the distance between $\vec{u_x}$ and each $\vec{u_j} \in U_p$ is crucial. There are many options available to define distance [18]. We define function $d_{\vec{\alpha}}(\vec{u_x}, \vec{u_j}) = \sum_{i=1}^{k} e^{\alpha_i} |u_{x,i} - u_{j,i}|$. This is $L1$ distance with a learned vector of weights, $\vec{\alpha} \in \mathbb{R}^k$, applied to each dimension after scaling with e^x to ensure non-negativity. This function provided satisfying results such that we did not feel the need to explore more options. We use an adjusted sigmoidal function, $\sigma^*(x) = \frac{2}{1+e^{-2x}} - 1$, to bound all distances ≥ 0 between $[0, 1)$. Note a hyperbolic tangent function can also achieve this. For input $\vec{x}$, the final vector of distances is defined $\vec{D} = [\sigma^*(d_{\vec{\alpha}}(\vec{u_x}, \vec{u_1})), \sigma^*(d_{\vec{\alpha}}(\vec{u_x}, \vec{u_2})), \ldots, \sigma^*(d_{\vec{\alpha}}(\vec{u_x}, \vec{u_\rho}))]$. Verdicts are made by thresholding the values in $\vec{D}$.

Loss Function: IUPG loss seeks to minimize the distance between samples and their designated prototype in $\mathbb{U}$ while simultaneously maximizing the distances between samples and all of their non-designated prototypes. The proposed loss function in Equation 1 is minimizing a summation of cross-entropy calculations between the label distributions of each target class and the prototype-to-class distributions of minimum distances in $\vec{D}$. We will define loss for a single sample $(\vec{x} \in \mathbb{I}, \vec{y} \in \mathbb{R}^c)$ where $\vec{y}$ is one-hot encoded over c **target** classes. Assuming ≥ 1 target classes each with ≥ 1 designated prototypes, the generalized loss function for $(\vec{x}, \vec{y})$ is shown in Equation 1.

$$\mathcal{J} = -\sum_{i=1}^{c} \gamma y_i \log\left(1 - \min\left(\left(\vec{D} + \epsilon\right) \odot \frac{1}{\vec{\omega_i}}\right)\right) + (1 - y_i) \log\left(\min\left(\left(\vec{D} + \epsilon\right) \odot \frac{1}{\vec{\omega_i}}\right)\right) \quad (1)$$

When $y_i = 1$, we use $\gamma \in \mathbb{R}$ to scale the relative influence of the distance to prototypes designated to class i. Recall $\vec{y} = \vec{0}$ for off-target samples, which necessitates the $y_i = 0$ term otherwise their loss would be 0 always. Conceptually, when $y_i = 1$, we penalize the distance between $\vec{x}$ and the closest prototype of class i. When $y_i = 0$, we do the same with inverted distance. We add a constant $\epsilon << 1$ to D to avoid computing $\log(0)$.

Assignment of ρ prototypes to c target classes is specified inside the $\vec{\omega_i} \in \mathbb{R}^\rho$ vectors. Denote the target class $[1, 2, \ldots, c]$ that prototype $\vec{p}$ is designated to as $\mathcal{C}(\vec{p})$. We define each $\vec{\omega_i}$ with $\omega_{i,j} = 1 + \epsilon$ if $\mathcal{C}(\vec{p_j}) = i$ and $\omega_{i,j} = \epsilon$ otherwise. $(\vec{D} + \epsilon) \odot 1/\vec{\omega_i}$ thus linearly shifts the values of $\vec{D}$ for class i such that the distances to designated prototypes is strictly < 1 while the distances to all other prototypes is ≥ 1. Computing $\min((\vec{D} + \epsilon) \odot 1/\vec{\omega_i})$ then gives us the minimum distance among the prototypes designed to class i.

Training and Inference Complexity: If all weights are unchanging, U_p need only be computed once and then can be reused. The time complexity of the mapping from $\mathcal{N}(\vec{x}) \rightarrow \vec{u_x}$ is $O(kz + k)$ which is equal to its CCE counterpart when $k = c$. The proceeding computation of $\vec{D}$ is composed of dot products, application of $\vec{\alpha}$ and application of $\sigma^*(\cdot)$, which scales $O(\rho k^2 + \rho k + \rho)$ accordingly. Assuming U_p is calculated prior, the previous two operations envelop the different operations of IUPG versus its CCE counterpart during inference. Note both are highly parallelizable and typically insignificant compared to the computation of $\mathcal{N}(\vec{x})$. During training, we must additionally compute U_p anew once per training batch. This is equivalent to adding ρ samples to each batch. Note IUPG also increases the number of learnable weights by $\rho|\vec{p}| + |\vec{\alpha}| + z(k - c)$.

IV. Experiments

We consider malicious JavaScript (JS) and URL classification as well as MNIST [14] and Fashion MNIST [29] classification in our experiments. For JS, we consider both a binary generic malware classification problem and a multiclass malware family tagging problem. All models are implemented in Tensorflow [1] and are trained with the Adam optimizer [11]. A training batch size of 32 and learning rate of 5×10^{-5} is used throughout. We use ReLU and sigmoidal activation across all convolutional and fully connected layers, respectively. These hyperparameters allow both IUPG and the CCE trained networks to converge after approximately the same number of batches. *The shared hyperparameters used in this work were tuned while using CCE loss – thus are biased toward CCE counterparts.* For IUPG, we set $k = 32$ throughout. When defining all $\vec{p} \in \mathbb{I}$, we used K-means++ [2] on the training data to calculate prototype initializations. When defining all $\vec{p}$ with a basis set, we used K-means++ to instead determine members of B. Three different networks are used in place of $\mathcal{N}$ for our experiments. The topology of $\mathcal{N}$ for all settings is illustrated in Figure 2.

Image Classification: For brevity, MNIST [14] and Fashion MNIST [29] are treated much the same. Both datasets

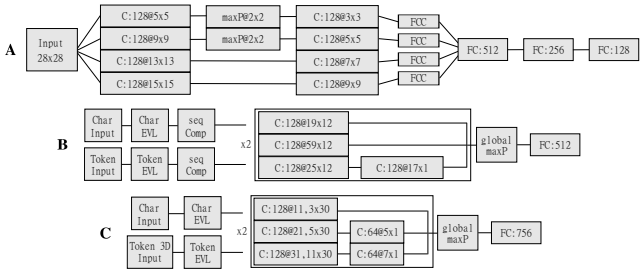

Figure 2: The function $\mathcal{N}$ used for **(A)** MNIST and Fashion MNIST, **(B)** JS, and **(C)** URLs. For **(A)**, the model consists of parallel convolutional layers. `C:128@5x5` is a convolutional layer of 128 5x5 filters. `maxP@2x2` is 2x2 max pooling. `FCC` is a fully connected convolutional layer. `FC:512` is a fully connected layer with 512 units. For **(B)**, char-level and token-level input representations are processed independently. `EVL` refers to an embedded vector lookup operation. `seqComp` refers to a sequence compression operation. `globalmaxP` refers to a global max pooling operation. For **(C)**, `C:128@11,3x30` denotes two different heights used in the filter banks: 11 for char-level input and 3 and token-level input.

are broken into random 50k-10k-10k Train-Test-Val (TTV) splits. When required, we generate Gaussian noise images as well as images of random strokes with a Random Forest Classifier to filter out accidental true positives. Images are preprocessed with max-min scaling and mean subtraction. When using IUPG, each $\vec{p} \in \mathbb{I} \subseteq \mathbb{R}^{28\times28}$ and we designate 1 prototype per target class.

Malicious JS and static URL Classification: Malicious JS and static URL classification are challenging tasks in web security [5, 13, 20]. Append attacks are particularly popular among JS malware, e.g. malicious injections to benign scripts, which motivates its study here. A simple yet critical observation of malware classification is that benignness is definable only insofar as that which is not malicious. *For IUPG, we define benign data as the off-target class, i.e. no prototypes are used to model it.*

Benign JS was collected by crawling the top 1M domains from Tranco list [19]. In addition to Tranco's filtering, we ignored samples flagged by state-of-the-art commercial URL filtering services. We leveraged VirusTotal (VT) [24] as the main source of malicious JS samples. We required a VT score (VTS) of at least 3 which was empirically shown to be reasonably accurate. Our malicious and benign URL data was collected from the use of static and dynamic URL filters and analyzers from an industry cybersecurity company over internet traffic as well external data sources (such as VT). For binary JS malware classification, we used a 450k-600k-600k TTV split with 70:30, 96:4, and 96:4 benign to malicious ratios, respectively. Substantial benign samples are included to accurately measure performance under strict false-positive rate (FPR) requirements. FPRs of $\leq 0.1\%$ are common in the industrial cybersecurity setting because of the exorbitant costs of FPs [23]. For building multiclass malware family tagging classifiers, we isolated 9 distinct malware families with 10k-1k-1k TTV samples per family. Equal parts benign data were added to form our multiclass training dataset. To generate OOD samples, we scrambled the order of tokens in benign scripts uniformly. For URLs, we used a 14M-2M-2M TTV split with 50:50 class ratios. We also collected a separate 2M, 50:50 test set one year after the initial collection to test recency bias.

Our JS and URL classifier architectures, illustrated in **(B)** and **(C)** of Figure 2, is inspired by prior work in NLP [8, 9, 10, 13, 25, 26, 27, 31]. All inputs are represented at two levels of abstraction: streams of chars and tokens. All URLs are padded to a fixed, maximal size while JS files are dynamically padded per batch. For token-level representations, **(B)** uses a single channel vocabulary of learned token embedded vectors chosen based on frequency. For **(C)**, we include a char-by-word channel similar to [13]. We additionally use an independently trained Hidden Markov Model to produce randomness scores for each token which scales a learned embedded vector to produce a third randomness channel. When using IUPG, for JS, each $\vec{p} \in \mathbb{I}$ with a fixed size. We designated 1 prototype per family for multiclass models while binary models have 4 prototypes designated for the malicious JS class – chosen empirically. For URLs, we experiment with all $\vec{p}$ both defined as a member of $\mathbb{I}$ and with a basis set of 100 malicious URLs. Empirically chosen, 4 and 100 prototypes are designated for the malicious URL class, respectively.

A. Classification Performance

		Full			Low-Shot		
FPR $\leq$		**1%**	**0.1%**	**0.01%**	**1%**	**0.1%**	**0.01%**
Multiclass	**IUPG**	$0.60\% \pm 0.02$	$0.77\% \pm 0.02$	$1.06\% \pm 0.11$	$3.54\% \pm 0.35$	$6.08\% \pm 0.31$	$9.36\% \pm 0.22$
	CCE	$0.62\% \pm 0.06$	$0.83\% \pm 0.03$	$2.48\% \pm 0.89$	$5.27\% \pm 0.79$	$7.90\% \pm 0.39$	$16.45\% \pm 3.81$
Binary	**IUPG**	$0.24\% \pm 0.02$	$0.75\% \pm 0.03$	$2.82\% \pm 0.17$	$3.34\% \pm 0.05$	$6.38\% \pm 0.12$	$10.99\% \pm 0.18$
	CCE	$0.46\% \pm 0.05$	$1.13\% \pm 0.07$	$3.50\% \pm 0.23$	$6.20\% \pm 0.68$	$10.49\% \pm 0.99$	$15.67\% \pm 1.15$

Table I: Malicious JS classification test set $\mu\pm$SEM FNR over all non-benign classes. For multiclass models, the low-shot training dataset consisted of 10 randomly selected samples per non-benign class; for binary models, 1000 randomly selected malware samples.

We explore classification performance on our various datasets with different combinations of training and testing

		Trained w/o Noise	Trained w/ Gaussian Noise
MNIST	**IUPG**	0.83 ± 0.03	0.95 ± 0.02
	CCE	1.00 ± 0.07	1.00 ± 0.03
Fashion MNIST	**IUPG**	8.57 ± 0.10	8.40 ± 0.09
	CCE	9.03 ± 0.10	8.94 ± 0.08

Table II: Image classification no-noise test set $\mu \pm$ SEM error percentages.

FPR $\leq$		1%	0.1%	0.01%
MNIST	**IUPG**	10.02 ± 0.54	19.18 ± 0.58	28.61 ± 1.88
	CCE	18.86 ± 2.37	52.73 ± 5.61	76.70 ± 4.33
Fashion MNIST	**IUPG**	31.93 ± 1.75	48.60 ± 2.60	55.72 ± 1.49
	CCE	59.25 ± 1.08	72.10 ± 1.62	81.87 ± 2.60

Table III: Image classification test set $\mu \pm$ SEM error percentages when the test set contains Gaussian noise images and models are trained without noise. Decision thresholds are configured to obey a maximum FPR.

	Original 2019 Test Set				2020 Test Set			
FPR $\leq$	**1%**	**0.1%**	**0.01%**	**0.001%**	**1%**	**0.1%**	**0.01%**	**0.001%**
IUPG†	3.61%	15.21%	33.93%	52.90%	39.92%	51.39%	63.47%	73.96%
IUPG	3.52%	16.01%	36.19%	62.56%	40.39%	53.77%	67.66%	84.55%
CCE	3.51%	16.11%	37.13%	63.56%	41.25%	54.35%	69.22%	100%

Table IV: Malicious URL classification test set FNR. $\dagger$ signifies a basis set was used to define all $\vec{p}$. Decision thresholds are configured to obey a maximum FPR.

with noise. When training a CCE counterpart with synthesized noise, we augment a dedicated noise class. For multiclass models, we define an FP as an off-target sample being classified as *any* target class. We used a single confidence threshold for all target classes. If surpassed, the maximum confidence target class is predicted. When testing without noise in Table II, the maximum confidence target class is predicted always. Note also that in all Tables except Table II, decision thresholds are configured to obey a maximum FPR. Results are presented over 5 trials with varying random seeds where applicable. We find a reliably stable or decreased false-negative rate (FNR), error percentage and variation across Tables I, II, III, and IV.

Note that Table III can also be interpreted as investigating OOD attack susceptibility since models are trained without noise and then tasked with classifying a test set that includes noise. In Table IV, we see IUPG retains more of its performance in the presence of distributional shift over a period of one year compared to CCE. Fitting with IUPG's central hypothesis, the noise-resistant features of the IUPG network are naturally more robust to distributional shifts in the benign class. Prototype definition strategies appear to affect performance. Clusters of malicious URLs are numerous and diverse. Intuitively, we'd see a benefit upon defining a large number of prototypes with a basis set.

	VTS = 0	**VTS > 0**
IUPG	184	312
Ensemble	223	203

Table V: Detections with 0.005% test set FPR configured thresholds organized by VTS.

As an additional investigation of our central hypothesis, we trained a singleton IUPG model (Figure 2, **(B)**) and a larger stacked ensemble of CCE networks for JS classification such that the test set FNR of the ensemble was *lower* than the IUPG model at the same FPR. We amassed a new collection of >5M JS samples taken from top-ranked popular websites. With thresholded test set FPR $\leq 0.005\%$, we cross-referenced all detections from both models on this dataset with VT [24]. A high VTS indicates a strong consensus of maliciousness among a large array of industry cybersecurity service providers. The VTS of the detections of both models is displayed in Table V. Importantly, we see a significant shift toward a higher VT consensus on IUPG detections *despite* an opposite performance gap on the test split. We feel this is important to highlight due to the prevalence of constructing TTV splits from a similar distribution but deploying models in more complex environments.

B. Out-Of-Distribution (OOD) Attack Simulations

In addition to the exploration in Table III, we explored the tendency of IUPG and its CCE counterpart to produce FP responses on OOD inputs at decision thresholds that are representative of confidence levels of in-distribution data. We are thus peering into the differing tendency of the models to output similar confidence levels on OOD samples as in-distribution samples. The results of our analysis are displayed in Figure 3.

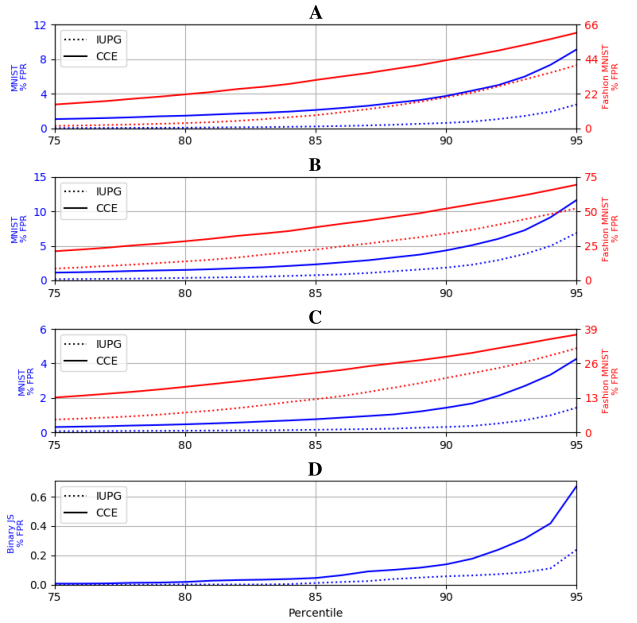

Figure 3: OOD attack simulation results. The FPR was measured over OOD test sets with decision thresholds configured based on the $75^{\text{th}} - 95^{\text{th}}$ percentiles of all confidence scores produced on target class test data. **(A)** Image classification models trained without noise over a Gaussian OOD test set. **(B)** Image classification models trained without noise over a random stroke OOD test set. **(C)** Image classification models trained with Gaussian noise over a random stroke OOD test set. **(D)** Binary JS classifiers over an OOD test set of randomized benign JS.

Fragment Size		10	100	1K	5K	10K	25K	50K	75K	100K
Multiclass Models	IUPG	0.00	8.04	17.06	19.80	20.85	31.22	37.33	36.27	36.22
	CCE	0.00	7.48	21.71	46.85	57.20	74.83	85.20	83.99	83.86
	IUPG‡	0.00	0.01	0.02	0.04	0.04	0.07	0.10	0.13	0.14
	CCE‡	0.01	0.01	0.04	0.09	0.11	0.22	0.54	0.76	0.83
Binary Models	IUPG	0.09	0.21	3.70	9.73	19.15	39.70	56.15	57.44	57.53
	CCE	0.38	3.16	23.72	42.23	52.42	74.38	86.35	84.09	86.69
	IUPG‡	0.05	0.04	0.04	0.25	0.50	1.28	2.57	2.09	5.61
	CCE‡	0.36	0.48	0.83	3.39	6.46	17.12	39.85	45.43	54.70

Table VI: Append attack simulation results. In each cell is the percentage of malware in which the model produces a malicious verdict on the original but a benign verdict upon appending a fragment of benign data of a given size in chars. 20 random fragments are tested per malware. Decision thresholds are configured to obey a maximum of 0.1% FPR on the test set. Adversarial training is denoted by ‡.

We find smaller FPRs with IUPG by a large margin when imposing decision thresholds representative of typical confidence levels on target class test data. A lower tendency to produce FPs on OOD content allows using looser decision thresholds in real-world systems leading to a higher recall. This result helps to corroborate the widening classification performance gap at stricter FPR requirements as seen throughout Section IV-A.

C. Append Attack Simulations

We explored the vulnerability to append attacks of our JS malware classifiers. The results of our simulation are displayed in Table VI. For each epoch, we also tried dynamically modifying all non-benign classes such that 33% of all its members are appended with a random benign fragment in the same TTV split. Fragments are given random sizes between 1000 and 5000 chars.

We found significant margins between the vulnerability of IUPG and its CCE counterpart with and without adversarial training. Critically, note the failure of the binary CCE‡ model to protect against append attacks beyond the fragment sizes used during training. Note that the binary dataset contains hundreds of malware families with one generic label thus represents a significantly harder problem compared to multiclass classification. Our multiclass classes are far less variable thus extracted features are free to be more specific – leading to less susceptibility to activation on noisy benign input. Additionally, note that the blackbox append attack can take the form of malicious injections into large benign files. Over a dataset of real-world malicious JS injections, we discovered the IUPG network to boost the number of detections from 76 to 2,259 over the aforementioned larger ensemble in Section IV-A. This pragmatic result corroborates the results in Table VI.

D. Fast-Gradient Sign Method (FGSM) Attacks

To demonstrate the potential to combine IUPG with existing adversarial training techniques, we combined the image classifiers with the Fast-Gradient Sign Method (FGSM) [7] training procedure. We discovered IUPG yields significantly more resistance to FGSM attacks compared to its CCE counterpart both with and without FGSM adversarial training. This is visualized in Figure 4.

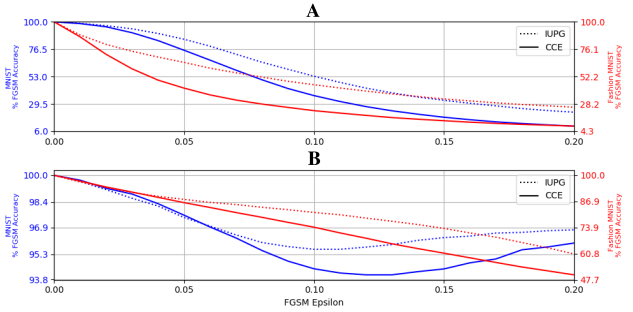

Figure 4: Accuracy over correctly classified test images versus the scaling factor of FGSM perturbations. **(A)** Standard training. **(B)** Models trained with the FGSM training procedure.

In part **(B)** of Figure 4, we use common FGSM training parameters of $\alpha = 0.9$, $\epsilon = 0.25$ on MNIST and $\alpha = 0.5$, $\epsilon = 0.05$ on Fashion MNIST. Fitting in with our core hypothesis, IUPG networks should be less sensitive to low-level perturbations by design. This is especially due to IUPG's prototyping mechanism which encourages exclusive sensitivity to high-level information shared among a subset of data. Both Figure 4 and Table VI demonstrate the superiority of using IUPG *combined* with special adversarial training compared to using either in isolation. We recommend combining strengths for the greatest success in future work.

V. Conclusion

We have presented the IUPG learning framework and demonstrated its impact on classification networks compared to CCE. Our core hypothesis is a boosted capacity to properly handle OOD content as provided by IUPG's inherent noise resistance and increased feature specificity. This feature logically connects all of the supportive results presented in this work: increased or stable classification performance, decreased performance loss due to recency bias, decreased FPs on OOD noise, and decreased vulnerability to some noise-based attacks. Properly handling OOD content is critical for models in real-world environments such as malware classification where benignness cannot be reasonably captured with a finite sample. Future applications of IUPG into existing adversarial learning and OOD detector techniques pose a promising direction for future work. We showed the unique benefits of IUPG are particularly useful to malware classification efforts. In that context, append attacks can lead to risky false-negatives while OOD failures can lead to costly false-positives.

References

[1] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 9780898716245.

[3] J. Bromley et al. Signature verification using a "siamese" time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7:25, 08 1993. doi: 10.1142/S0218001493000339.

[4] J. Chen, Y. Li, X. Wu, Y. Liang, and S. Jha. Robust out-of-distribution detection for neural networks, 2020.

[5] A. Fass, M. Backes, and B. Stock. Jstap: A static pre-filter for malicious javascript detection. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC '19, page 257–269, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450376280. doi: 10.1145/3359789.3359813.

[6] Y. Geifman and R. El-Yaniv. Selective classification for deep neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4878–4887. Curran Associates, Inc., 2017.

[7] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv e-prints*, art. arXiv:1412.6572, Dec. 2014.

[8] R. Johnson and T. Zhang. Semi-supervised convolutional neural networks for text categorization via region embedding. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 919–927. Curran Associates, Inc., 2015.

[9] R. Johnson and T. Zhang. Effective use of word order for text categorization with convolutional neural networks. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 103–112, Denver, Colorado, May–June 2015. Association for Computational Linguistics. doi: 10.3115/v1/N15-1011.

[10] Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751, 2014.

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[12] T. Kohonen. The self-organizing map. *Neurocomputing*, 21(1):1 – 6, 1998. ISSN 0925-2312. doi: https://doi.org/10.1016/S0925-2312(98)00030-7.

[13] H. Le, Q. Pham, D. Sahoo, and S. C. H. Hoi. Urlnet: Learning a url representation with deep learning for malicious url detection, 2018.

[14] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[15] O. Li, H. Liu, C. Chen, and C. Rudin. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. 2018.

[16] X. Liu and et al. Meta-learning based prototype-relation network for few-shot classification. *Neurocomputing*, 383:224 – 234, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2019.12.034.

[17] D. Nova and P. A. Estévez. A review of learning vector quantization classifiers. *Neural Comput. Appl.*, 25(3–4):511–524, Sept. 2014. ISSN 0941-0643. doi: 10.1007/s00521-013-1535-3.

[18] S. Ontañón. An overview of distance and similarity functions for structured data. *Artificial Intelligence Review*, Feb 2020. ISSN 1573-7462. doi: 10.1007/s10462-020-09821-w.

[19] V. L. Pochat, T. van Goethem, and W. Joosen. Rigging research results by manipulating top websites rankings. *CoRR*, abs/1806.01156, 2018.

[20] D. Sahoo, C. Liu, and S. C. H. Hoi. Malicious url detection using machine learning: A survey, 2017.

[21] A. Sato and K. Yamada. Generalized learning vector quantization. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, page 423–429, Cambridge, MA, USA, 1995. MIT Press.

[22] V. Sehwag et al. Analyzing the robustness of open-world machine learning. pages 105–116, 11 2019. ISBN 978-1-4503-6833-9. doi: 10.1145/3338501.3357372.

[23] T. site produced and w.-p. maintained by Byte Productions. The cost of malware containment, Jan 2015.

[24] G. Sood. *virustotal: R Client for the virustotal API*, 2017. R package version 0.2.1.

[25] J. W. Stokes, R. Agrawal, G. McDonald, and M. J. Hausknecht. Scriptnet: Neural static analysis for malicious javascript detection. In *2019 IEEE Military Communications Conference, MILCOM 2019, Norfolk, VA, USA, November 12-14, 2019*, pages 1–8. IEEE, 2019. doi: 10.1109/MILCOM47813.2019.9020870.

[26] M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, and D. Poshyvanyk. Deep learning similarities from different representations of source code. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, page 542–553, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450357166. doi: 10.1145/3196398.3196431.

[27] Y. Wang, W.-d. Cai, and P.-c. Wei. A deep learning approach for detecting malicious javascript code. *Sec. and Commun. Netw.*, 9(11):1520–1534, July 2016. ISSN 1939-0114. doi: 10.1002/sec.1441.

[28] R. R. Wiyatno, A. Xu, O. Dia, and A. de Berker. Adversarial examples in modern machine learning: A review, 2019.

[29] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[30] H.-M. Yang, X.-Y. Zhang, F. Yin, and C.-L. Liu. Robust classification with convolutional prototype learning. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun 2018. doi: 10.1109/cvpr.2018.00366.

[31] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 649–657, Cambridge, MA, USA, 2015. MIT Press.