Detecting Adversarial Patches with Class Conditional Reconstruction Networks

Perry Deng
Rochester Institute of Technology
perry.deng@mail.rit.edu

Mohammad Saidur Rahman
Rochester Institute of Technology
saidur.rahman@mail.rit.edu

Matthew Wright
Rochester Institute of Technology
matthew.wright@rit.edu

ABSTRACT
Defending against physical adversarial attacks is a rapidly growing topic in deep learning and computer vision. Prominent forms of physical adversarial attacks, such as laser or adversarial patches and objects, share similarities with digital attacks, but are easy for humans to notice. This leads us to explore the hypothesis that adversarial detection methods, which have been shown to be ineffective against adaptive digital adversarial examples, can be effective against these physical attacks. We use one such detection method based on autoencoder architectures, and perform adversarial patching experiments on MNIST, SVHN, and CIFAR10 against a CNN architecture and two CapsNet architectures. We also propose two modifications to the EM-Routed CapsNet architecture, Affine Voting and Matrix Capsule Dropout, to improve its classification performance. Our investigation shows that the detector retains some of its effectiveness even against adaptive adversarial patch attacks. In addition, detection performance tends to decrease among all the architectures with the increase of dataset complexity.

KEYWORDS
Physical Adversarial Attacks; Adversarial Patch; Detection; Capsule Network; EM Routing; Affine Voting; Dropout;

Resources
Source code: https://github.com/PerryXDeng/detecting-adversarial-patches-capsule-networks

1 INTRODUCTION
Deep-learning (DL) systems have shown tremendous success in complex tasks such as image recognition [19], speech recognition [13], object detection, classification, tracking [10, 23], and so on. Much research, however, has shown that DL systems are vulnerable to digital adversarial examples, carefully crafted inputs to cause unintended behaviors. One method of defending against these attacks is to detect the attacks and enable the system to formulate a new action based on the detection. Even though a number of methods have been proposed for detecting adversarial examples [12, 27] in the digital domain, these methods have shown to be ineffective against an adaptive attacker [4] who targets a DL system to cause misclassification and evade detection.

Growing research shows that similar vulnerabilities exist in the physical domain as well [8, 22]. Physical adversarial attacks are created to be robust to variance in scene and sensor factors such as lighting, rotation, and distance, rather than merely making them effective in a single example image. For example, attackers can craft adversarial patches and place them anywhere within the field of view of an object detector to suppress object detection [22]. This is achieved through Expectation over Transformation (EOT) [1]. EOT simulates a distribution of locations, orientations, and photometric properties of the patch, and then optimizes the attack across the dataset with the distribution applied. Unlike digital adversarial examples, adversarial patches have to occupy a continuous area in input images. This property makes them easy to be spot for humans, for example in the cases of toaster patches [3] or printed t-shirts [38]. Adversarial patches are cheap to craft, and unlike digital adversarial examples, can work in many scenarios. Adversarial patch attacks on time-sensitive safety-critical systems, such as a self-driving car, can jeopardize users as well as the general public. Detecting adversarial patch attacks, however, has received considerably less attention until recently.

We note that the only differences between the generation of adversarial patches and digital adversarial examples are that: i) adversarial patches are optimized over a distribution of input images rather than a single image, ii) adversarial patches are area constrained rather than norm constrained. We thus hypothesize that neural networks under attack by adversarial patches are behaviorally similar to neural networks under attack by digital adversarial examples, and methods to detect adversarial perturbations can be transferred to detection of adversarial patches.

Intuitively, it should also be easier to automate the detection of adversarial patches compared to digital adversarial examples, as humans can easily spot them. To the best of our knowledge, SentiNet [8] is the only prior work done on detecting adversarial patches. SentiNet detects and segments adversarial objects from input by leveraging the high salience and scene-transferability of adversarial patches.

In this paper, we adapt a mechanism originally proposed to detect digital adversarial perturbations [31] and empirically examine its effectiveness against simulated physical adversarial attacks. In particular, we examine the effectiveness of class-conditional reconstructions [31] – which is lackluster in the digital domain – against adversarial patches. We show that unlike digital adversarial examples, adaptive attacks using adversarial patches do not neutralize the detector. We have implemented the detection approach on a
modified AlexNet as well as on a Capsule Network (CapsNet), a recently proposed and arguably powerful neural architecture [14, 32]. CapsNets use layers of vector- or matrix-valued capsules that represent the poses and probabilities of objects and can also represent part-whole relationships. Similar to how layers of neurons can represent the extraction of higher-level features from lower-level features in a standard neural network, layers of capsules can represent the extraction of higher-level objects from lower-level objects, shapes, or lines. In addition, we propose an improved CapsNet architecture incorporating DropOut and Affine Voting to enhance the test set classification performance.

In summary, our contributions are as follows:

- We are the first to explore the effectiveness of a digital adversarial example detection method in the adversarial patch domain, and demonstrate it on three standard datasets: MNIST, SVHN, and CIFAR10.
- We are the first to show the effectiveness of CapNets to classify and detect adversarially patched input.
- We propose an improved Matrix Capsule Network with capsule-dropout and affine voting and empirically show their improved test accuracy on the smallNORB dataset.

2 RELATED WORK

Digital Adversarial Examples. Digital adversarial examples are inputs slightly perturbed by attackers to cause unintended behaviors in deep-learning (DL) systems. The creation of adversarial examples in the digital domain has been extensively studied using convolutional neural networks (CNNs) [6, 28, 34]. These digital domain attacks typically assume that the threat actors can modify pixels of the input images at will, and seek to optimize an adversarial objective that would lead to misclassification subject to some constraints on the amount of modification.

While many methods have been proposed to detect digitally perturbed adversarial input [9, 25, 26, 29, 39], none of these approaches have been shown to be robust against an adaptive digital domain attacker aware of the defense mechanism [4, 5, 31, 40]. Some of the more promising approaches, however, force the attacker to employ digital perturbations that are more noticeable to human observers [9, 31, 39, 40].

Physical Adversarial Attacks. Unlike digital adversarial examples, physical adversarial attacks do not assume that an attacker can manipulate any pixel on a particular input image. In the computer vision setting, physical adversarial attacks produce an object (which we generically refer to as a patch, as objects are but patches of pixels in a 2d image) that tricks the image classifier into seeing a particular type of object. These patches are designed to be robust to variance in scene and sensor factors such as lighting, rotation, and distance, rather than merely effective in a single example image. This is achieved through Expectation over Transformation (EOT), a method first proposed by Athalye et al. [1].

EOT simulates a distribution of locations, orientations, and photometric properties of the patch and then optimizes the attack across all the data with the simulated distribution applied. Patches can be designed to modify a specific target object to be classified or to appear with any other objects. Attack that modifies the target objects can directly manipulate the surfaces of the objects using technologies such as 3D printing. One example of this attack produces an object that looks like a turtle to human but is classified as a rifle by the neural network [1]. The second type of physical attacks pose more threat to an undefended system, as they do not assume the ability to modify the target object. Rather they place adversarial objects or patches in the scene intended to interfere with the neural networks. Examples of these attacks include i) hiding people from detection through adversarial patches on t-shirts [35], ii) causing neural algorithms to lose track of vehicles through adversarial patches [16].

The investigation of this paper is adversarial patch centric as it requires relatively low cost to perform such an attack. Adversarial patches can fool a trained DL system to misclassify [3] when applied to a sticker or other objects. Unlike constraining the magnitude of pixel change of digital adversarial examples, this class of attacks constrains the payloads to be patches that cover a continuous area, which makes it easy for humans to recognize.

Capsule Networks. Capsule networks are a recently proposed alternative class of neural network architecture [14, 15, 18, 32] that use layers of vector- or matrix-valued capsules, which represent the poses and probability of objects. Similar to how layers of neurons can represent the extraction of higher-level features from lower-level features in a standard neural network, layers of capsules can represent the extraction of higher-level objects from lower-level objects, shapes, or lines. A typical DNN consists of layers of scalar-valued neurons that are connected together using linear combinations and non-linear activation functions.

By contrast, in a capsule network, layers of capsules vote on the poses and probability of higher level capsules, and the probabilities of their existence are determined by a procedure known as Routing by Agreement [14, 32]. Routing by Agreement eliminates the need for pooling, a dimensionality reduction technique commonly used in a CNN. Two approaches to this routing have been proposed, both of which have the theoretical property of geometric equivariance. This makes them robust to geometric transformations of objects without the need to memorizing different viewpoints and placement of objects, and reduces the need for training data [14]. The first approach, Dynamic Routing [32], measures agreement between capsules using the dot product between votes and capsules that are voted on, and use an ad hoc squashing function to normalize the final vector capsule output with the magnitude being its probability. The Dynamic Routing implementation has shown state-of-the-art performance in tasks such as medical image segmentation [21]. The second approach, EM Routing [14], measures agreement between votes using Gaussian clusters estimated with a modified Expectation-Maximization (EM) algorithm.

The EM Routing implementation allows the use of matrix capsules, and is shown to generalize better to novel viewpoints while requiring less data to train [14] on the smallNORB dataset, compared to CNN. Our architecture is based on matrix capsules with EM Routing, the second implementation. Our proposed improvements to the architecture are discussed in Section 4.3.

Digital Adversarial Example Detection with Reconstruction Networks. MagNet [26] shows that autoencoders have the potential to counter adversarially perturbed input, which are outside of
While attacks have been examined for all of those vulnerabilities, the adversarial detection function $f_{\text{detect}}$ (CVS), namely an autonomous vehicle, which should: i) capture an image of the stop sign from the physical world, ii) feed the image into the trained deep neural network, and iii) provide the classification output to a decision engine that stops the car.

From an attacker’s point of view, the example of self-driving car poisoning assumes the least amount of privileged access to the targeted system (as was done in a non-autonomous vehicle [11]), poisoning the neural network model in the CVS, even though they do not possess privileged access to the instance of the targeted system. This white-box scenario is a reasonable assumption, as it has been shown to be possible for an attacker to transfer payloads trained against a substitute model to the targeted black-box model that attackers do not have access to [30]. With this assumption, we then evaluate the effectiveness of our detection method by analyzing its robustness to two attacks:

1. A naive white-box attack that focuses on creating a strong attack against the classifier with no regard for the detection in place.
2. An adaptive white-box attack that attempts to fool both the classifier and the detector.

For detection to be considered effective, it needs to not only effectively detect strong naive attacks, but also maintain a reasonable performance against adaptive attacks. This evaluation methodology is identical to the ones used by Biggio et al. [2] and Carlini and Wagner [4]. The two attacks used for evaluation are described in detail in Section 5.1.

The manifold boundary of the normal input, by reconstructing the input as if they are inside the normal manifold. This allows the detection of digital adversarial examples when the class-conditioned reconstruction is far from the provided input, a method first proposed by Qin et al. [31]. The method feeds the class-conditioned latent variables of the image classifier into a decoder network turning the architecture into an autoencoder. The decoder output then outputs a tensor that attempts to reconstruct the input images given latent variables that represent features of the predicted class instance. In a capsule network, the latent variables are the class capsule vectors or matrices in the final layer of the classifier. The class conditional decoder can also be attached to a convolutional network, albeit with some modifications to the penultimate layer in the CNN [31]. The $L_2$ distance between the reconstructed input $x_{\text{recon}}$ and the actual input $x$, is minimized when training the network. The adversarial detection function $f_{\text{detect}}$, which outputs 1 when the input is considered to be adversarial, and 0 otherwise, is defined as:

$$f_{\text{detect}}(x) = H(\Delta_{\text{recon}} - \alpha_{\text{crit}}),$$

where $H$ is the Heaviside step function, $\Delta_{\text{recon}} = \frac{||x_{\text{recon}} - x||_2}{\text{height} \times \text{width}_{x}}$ is the reconstruction loss, and $\alpha_{\text{crit}}$ is the critical threshold. In our experiments, we pick $\alpha_{\text{crit}}$ to be at the 95th percentile of reconstruction losses on test set images, so that the false positive rate is controlled at 5%.

### 3 THREAT MODEL

#### 3.1 Physical Adversarial Attacks

From an attacker’s point of view, the example of self-driving car in Figure 1 can be exploited to behave in an unintended and potentially dangerous manner by taking over the steering or power system (as was done in a non-autonomous vehicle [11]), poisoning the neural network training data to plant backdoors [24], manipulating the capturing or preprocessing procedure to modify images to have human-imperceptible perturbations that target neural networks [6, 28, 34], or crafting and placing physical adversarial objects in the environment that target the neural networks [1, 3, 20].

While attacks have been examined for all of those vulnerabilities, placement of crafted adversarial payloads in the physical environment assumes the least amount of privileged access to the targeted systems. Sophisticated adversarial attacks have been proposed to create physical adversarial patches and objects that are robust to environmental variances, and can exploit different parts of the computer vision pipeline such as object classification [1, 3], object detection [7, 22], and object tracking [16].

#### 3.2 Attack Detection

While physical payloads created by the aforementioned techniques are in general more noticeable to humans than digital adversarial perturbations, a system armed with an automatic detector of these attacks can mitigate risks, as they do not require trained eyes to identify such a situation. For example, a self-driving car can relegate its steering to a licensed driver once it detects adversarial objects in the environment, thereby reducing the chances of unexpected maneuvers and traffic collisions, without requiring professionals trained at spotting such attacks.

In this paper, we concentrate on detecting the presence of adversarial patches or objects in the object classification pipeline, as it is relatively easy to demonstrate and measure the results. It is also possible to adapt our model to a more complex computer vision system (CVS). To evaluate the effectiveness of our detection method, we assume the worst-case scenario for adversarial patch or object attacks, in which the threat actors possess access to parameters of the neural network model in the CVS, even though they do not possess privileged access to the instance of the targeted system. This white-box scenario is a reasonable assumption, as it has been shown to be possible for an attacker to transfer payloads trained against a substitute model to the targeted black-box model that attackers do not have access to [30]. With this assumption, we then evaluate the effectiveness of the detection method by analyzing its robustness to two attacks:

1. A naive white-box attack that focuses on creating a strong attack against the classifier with no regard for the detection in place.
2. An adaptive white-box attack that attempts to fool both the classifier and the detector.

For detection to be considered effective, it needs to not only effectively detect strong naive attacks, but also maintain a reasonable performance against adaptive attacks. This evaluation methodology is identical to the ones used by Biggio et al. [2] and Carlini and Wagner [4]. The two attacks used for evaluation are described in detail in Section 5.1.
### 4 NETWORK ARCHITECTURES

#### 4.1 CNN

The CNN architecture mostly follows AlexNet [19], with modifications to allow for class conditional reconstruction as in Qin et al. [31]. In particular, we substitute the penultimate fully connected layer in AlexNet with logistic regressions on $k$ embedding class vectors, one for each class in the dataset, each of size $m$. We set $m = 16$, which is the same size as the matrix capsules used for reconstruction in the capsule architectures. We also substitute the last layer in AlexNet to linearly transform each of the $k$ embedding vectors into a logit, stacked together before softmax activation, which yields the probabilities for each of the $k$ classes. From the penultimate layer consisted of $k$ logistic regressions, we select a single embedding vector from the $k$ class vectors, which is either the vector for the ground truth class during training time, or the predicted class at test time.

The selected class embedding vector is then fed into the decoder portion of the network. The decoder consists of two hidden fully connected layers (Hidden FC1, Hidden FC2) with hypertangent activation and an output layer (Reconstruction Output Layer) that has the same number of neurons as the number of pixels in the input image. The reconstructed output is compared to the input image, which yields the reconstruction loss. The reconstruction loss is compared to a threshold for detection and minimized during model training and adaptive attack optimization. The input images are scaled to side lengths of 224 in order to be consistent with [19].

#### 4.2 CapsNet Architecture

We utilize two CapsNet-based architectures for detection. The baseline model is the one proposed by Hinton et al. [14], while the other model includes the two modifications discussed in Section 4.3. They both consist of a classifier and a decoder.

The classifier is identical to Hinton et al.’s [14] in both models, and as shown in Figure 2, it consists of a convolutional layer (Conv), followed by a primary capsule layer (PrimCaps), followed by two convolutional capsule layers (ConvCaps1, ConvCaps2), and finally followed by a class capsule layer (ClassCaps). As in Hinton et al. [14], we have kernel size of 5 in Conv, 1 in PrimCaps, and 3 in ConvCaps1 and ConvCaps2. The numbers of convolutional kernels, capsule convolutional kernels, capsules, and fully connected neurons differ by the dataset (Table 1), but are the same for our two models. The class capsule layer outputs a classification loss for model training or an adversarial cost for attack optimization. The classification loss is a modified version of hinge loss [14].

In both of our CapsNet models, a capsule is selected from the class capsule layer, which is either the capsule for the ground truth class during training time or, at test time, the most likely capsule. The matrix pose of the selected capsule is flattened into a vector and then fed into the decoder. The decoder operates just as the decoder for the CNN.

| Dataset    | $\lambda_{capsule}$ | $\lambda_{conv, fc}$ | model batch size | patch batch size |
|------------|---------------------|----------------------|-----------------|-----------------|
| smallINORB | 0                   | 2e-07                | 64              | N/A             |
| MNIST      | 0                   | 2e-07                | 128             | 128             |
| SVHN       | 1e-06               | 1e-06                | 24              | 16              |
| CIFAR10    | 0                   | 2e-07                | 16              | 16              |

Table 2: Regularization parameters and batch sizes for training the CapsNet architecture.

For training both our models and our patches, we use a learning rate of 0.003 in ADAM optimizer and two routing iterations for expectation maximization. The final temperature hyperparameter for the modified hinge loss is configured to be 0.01 for all the CapsNet training sessions, consistent with [14]. Batch sizes and regularization hyperparameters used to train the CapsNet architectures on different datasets are contained in Table 2. Since CapsNets are very slow to train, we terminate their training sessions after 5 days even if they have not yet converged. For other implementation details, refer to our source code to be released after publication.

#### 4.3 Improved CapsNet

We were not able to achieve satisfactory accuracies on SVHN and CIFAR10 using the architecture from Hinton et al. [14]. Since inputs that are misclassified are adversarial from the start, with no effort from the attacker, we propose two improvements on the CapsNet architecture to get it on a more equal footing for the comparison with AlexNet. Our Improved CapsNet differs in that the capsule layers are configured with Matrix Capsule Dropout (MCD) and Affine Voting.

![Figure 3: An abstraction of our modified matrix capsule layer, with MCD (Section 4.3) and affine voting (Section 4.3).](image-url)
We provide an intuitive explanation of these improvements along with empirical evidence on the smallNORB dataset. To the best of our knowledge, we are the first to attempt dropout and affine voting on matrix capsule layers (Figure 3).

Matrix-Capsule-Dropout. Dropout was proposed by Srivastava et al. [33] in 2014, and since then it has been used in almost all of the state-of-the-art CNN architectures due to its regularizing effects. It approximates ensemble learning in a single dense network by randomly setting neuron activation outputs to zero with probability \( p \). We leverage the same intuition in designing matrix-capsule-dropout (MCD). If a matrix capsule layer has MCD enabled, then the output capsule activation probabilities of its EM routing procedure during forward propagation will be randomly zero-masked with probability \( p \) after the last expectation maximization step. This is a computationally inexpensive procedure, as we are only masking the activation probabilities of the capsules without tampering with their matrix representation. In our model, we set \( p = 0.5 \) and apply MCD to randomize the capsule activation probabilities of the second to last and third to last layers in the classification network, so the hidden capsules voting for class capsules in the last layers are deactivated randomly. We are able to reduce over-fitting and improve test accuracy with this implementation, as shown in Table 3. The choices regarding \( p \) and affected layers were made without hyperparameter optimization, so better results may be possible. To the best of our knowledge, this is the first work to implement dropout on matrix capsule layers.

We also attempted DropRoute, an adaptation of DropConnect [36] on the Routing by Agreement mechanism, by randomly assigning routing weights to zero after each expectation maximization step. As we can see from Table 3, DropRoute was not effective.

Affine Voting. Hinton et al. mention that a lower-level capsule calculates its votes for capsules in the higher levels by “multiplying its own pose matrix by trainable viewpoint-invariant transformation matrices that could learn to represent part-whole relationships” [14]. In other words, votes on higher-layer capsules’ pose matrices are linear transformations of lower-layer capsules. Geometrically, the relationships between higher- and lower-layer capsule poses estimated by those linear votes are compositions of rotations, reflections, and scaling.

On the other hand, our architecture calculates the votes through affine transformations, which multiplies a pose matrix with a trainable transformation matrix, followed by addition of a trainable bias matrix. While this is partly inspired by the use of bias terms in CNNs, geometrically affine transformations can model compositions of rotations, reflections, and scaling, followed by translations [37]. Thus, they are strictly more expressive than that of the default linear transformations proposed in the EM Routing paper, and can model more general relationships between higher and lower level...
Table 3: Statistics from twelve 120-hour training sessions on smallNORB dataset (lasting 220-240 epochs each). The train partition accuracies and test partition accuracies from the epoch checkpoints with the best test partition accuracies are reported, for each training session. We can see that matrix-capsule-dropout (MCD) noticeably reduces over-fitting.

|             | Unmodified | MCD | DropRoute |
|-------------|------------|-----|-----------|
| Train Acc. | 100.00%    | 99.99% | 99.98% |
| Test Acc.  | 94.52%     | 95.52% | 95.52% |
| Train Acc. | 99.99%     | 99.82% | 99.85% |
| Test Acc.  | 94.48%     | 95.72% | 94.70% |
| Train Acc. | 99.99%     | 100.00%| 99.85% |
| Test Acc.  | 94.04%     | 95.52% | 93.69% |

Wilk p=0.840), and we find a statistically significant improvement.

5 DETECTION EXPERIMENT SETUP

We now present the setup for our adversarial patch detection experiments. Section 5.1 elaborates on the optimization objective for training the adversarial patches, while Section 5.2 defines the evaluation metrics. Note that since physical adversarial attacks are generally first created digitally then transferred over to the physical domain [1, 3], it is plausible that the effectiveness of the detector on the digital attacks also transfers to the physical domain.

5.1 Adversarial Optimization Objective

We use two types of adversarial patching attacks, a naive whitebox attack and an adaptive whitebox attack, to assess detection performance.

Both attacks use an overarching optimization based on Expectation over Transformations (EOT) [1]. With EOT, optimization objective in both attacks can be expressed as:

\[ \theta_{patch} = \arg \min_{\theta_{patch}} \mathbb{E}_{x \sim X, t \sim T, l \sim L} [J_{adv}(x, t, l, P, \theta_{patch}, A)], \]

where \( \theta_{patch} \in \mathbb{R}^{w' \times h' \times 3} \) are parameters for the patch of width \( w' \) and height \( h' \), \( X \) is the training set of images, \( T \) is the distribution of transformations, \( L \) is the distribution of locations over images, \( J_{adv} \) is the scalar valued adversarial cost function, \( P : \mathbb{R}^{w' \times h' \times 3} \rightarrow [0, 1]^{w' \times h' \times 3} \) is a differentiable function transforming the patch parameters into an RGB patch, and \( A : [0, 1]^{w' \times h' \times 3}, [0, 1]^{w' \times h' \times 3}, T, L \rightarrow [0, 1]^{w' \times h' \times 3} \) is a differentiable function applying the patch onto the training RGB image with width \( w' \) and height \( h' \). Note that the RGB values in this paper are floats between 0 and 1 due to dataset normalization. For both attacks,

\[ P(\theta_{patch}) = (\tanh(\theta_{patch}) + 1)_{w' \times h'}/2 \]

which transforms the patch parameters in \( R^{w' \times h' \times 3} \) to \( [0, 1]^{w' \times h' \times 3} \), and thus prevents the need for clipping functions to restrict the image in normalized RGB space. This improves the effectiveness of gradient descent on the smoother objective function and is inspired by the Change of Variables approach from [6].

Our attacks use the patch applier function \( A \) from the source code used by Brown et al. [3], which samples from the distribution of random transformations, locations, training images, and returns a new image with the randomly transformed patch masking a random location on the training image. We also use the same parameters for \( A \), except we changed the uniform distribution of patch scaling to be between 10% and 50% of the input image instead of between 30% and 150%. This was to improve the patching effectiveness at lower scales.

Given the attacker’s goal of causing misclassification, the neural network’s confidence on the correct class needs to be lower than its confidence on the incorrect classes. Due to the universal nature of physical adversarial objects and patches, the most straightforward method to cause misclassification is to aim at a single adversarial target class \( c_{adv} \). In our experiments, we arbitrarily set \( c_{adv} = 0 \).

Both attacks use an iterative numerical method, namely ADAM [17], to search for the strongest \( \theta_{patch} \), due to its relative efficiency compared to other iterative methods and its ability to find stronger adversarial solutions than analytical methods [6]. It is also relatively insensitive to hyperparameter choices.

The two attacks differ in their adversarial cost function \( J_{adv} \), which we define as follows.

**Naive Whitebox Attack.** For this attack, in complement to the overall optimization framework in Equation 2, we have,

\[ J_{naive}(x, t, l, P, \theta_{patch}, A) = \max_{i \neq c_{adv}} (\log(Z(x'_i)) - Z(x')_{c_{adv}}) + \kappa) \]

where \( Z(x'_i) \) is the logit for the activation of class \( i \) capsule, \( \kappa \) is the ideal confidence gap between \( c_{adv} \) and the next most confident class, and \( x' \) is the adversarially patched image. We can define \( x' \) as:

\[ x' = A(P(\theta_{patch}), x, t, l) \]
### Table 4: Evaluation statistics of patches on the MNIST dataset, randomly applied to 40% of input image area.

| $\lambda_a$ | CNN Test Acc.: 99.50% | CapsNet Test Acc.: 99.32% | Improved CapsNet Test Acc.: 99.50% |
|-------------|------------------------|---------------------------|-----------------------------------|
|             | ADR | AFR | ASR | ADR | AFR | ASR | ADR | AFR | ASR |
| 0           | 0.180235 | 0.002251 | 0.000887 | 0.985786 | 0.052193 | 0.002554 | 0.987896 | 0.026430 | 0.001222 |
| 0.5         | 0.167258 | 0.002329 | 0.001331 | 0.962798 | 0.042199 | 0.003665 | 0.971793 | 0.026874 | 0.004664 |
| 1           | 0.194099 | 0.001664 | 0.000998 | 0.869961 | 0.025208 | 0.005552 | 0.812660 | 0.017324 | 0.005997 |
| 1.5         | 0.160825 | 0.002329 | 0.001664 | 0.748917 | 0.015547 | 0.004220 | 0.604886 | 0.010661 | 0.003443 |
| 2           | 0.174579 | 0.001996 | 0.001331 | 0.605775 | 0.010105 | 0.002443 | 0.522710 | 0.010883 | 0.003887 |
| 2.5         | 0.190772 | 0.001775 | 0.000998 | 0.514936 | 0.008773 | 0.002323 | 0.472959 | 0.008884 | 0.003554 |
| 3           | 0.154281 | 0.001886 | 0.001442 | 0.459856 | 0.007885 | 0.001444 | 0.443753 | 0.006996 | 0.002443 |
| 3.5         | 0.165373 | 0.002773 | 0.001775 | 0.446086 | 0.008440 | 0.001333 | 0.421877 | 0.007440 | 0.003554 |
| 4           | 0.166260 | 0.001442 | 0.000887 | 0.440422 | 0.007551 | 0.001888 | 0.407773 | 0.006108 | 0.003220 |
| 4.5         | 0.164485 | 0.002884 | 0.001886 | 0.435980 | 0.008329 | 0.002110 | 0.379678 | 0.005441 | 0.002665 |
| 5           | 0.165594 | 0.002329 | 0.001886 | 0.429872 | 0.007885 | 0.001110 | 0.363021 | 0.004109 | 0.001999 |
| 10          | 0.158829 | 0.001109 | 0.000776 | 0.420988 | 0.005886 | 0.001333 | 0.356469 | 0.003665 | 0.001777 |
| 100         | 0.082298 | 0.002107 | 0.001775 | 0.423431 | 0.006996 | 0.002677 | 0.360355 | 0.005108 | 0.002443 |
| 1000        | 0.074091 | 0.001886 | 0.001553 | 0.429539 | 0.007218 | 0.000888 | 0.359134 | 0.003554 | 0.001777 |
| 10000       | 0.077751 | 0.002551 | 0.002218 | 0.423987 | 0.007218 | 0.001110 | 0.359578 | 0.003554 | 0.001333 |

This adversarial cost function is similar to the most powerful objective identified by Carlini and Wagner [6], with the two differences being that we optimize over batches of adversarially patched images rather than perturbation over a single image, and we do not seek to minimize the distance between $x^\prime$ and $x$, since we expect the patch to be visible to human observers. We set $\kappa = 20$, as this confidence gap was found to increase the transferability of adversarial perturbations [6]. It is possible that other values of $\kappa$ could achieve better adversarial patching results.

**Adaptive Whitebox Attack.** As we seek to both fool the classifier and evade the detector (Equation 1) with this attack, we modify the adversarial cost function $J_{\text{naive}}$ to include the constraint that $f_{\text{detector}}(x) = 0$. However, as the Heaviside step function does not have a non-zero gradient outside the decision boundary, the Lagrangian method cannot directly be used for this constraint. Instead, we replace the constraint as $\Delta_{\text{recon}} = 0$, and include a Lagrangian term for this constraint so that the generated patch will seek to not only fool the detector but also minimize the reconstruction loss. We then have,

$$J_{\text{adaptive}}(x, t, l, P, \theta_{\text{patch}}, A, \lambda_a) = J_{\text{naive}}(x, t, l, P, \theta_{\text{patch}}, A) + \lambda_a \cdot \Delta_{\text{recon}}.$$  (6)

where $\lambda_a$ is a new hyperparameter that serves as the Lagrangian multiplier and balances the objective to fool the classifier and the constraint to evade the detector. To determine the strongest $\lambda_a$ that yields the highest proportion of undetected successful attacks, we utilized a simple grid search on $[0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5,10,100,1000,10000]$.

In our experiments, patches developed with the two methods are applied to and optimized over the test set images, in contrast to the CapsNet models, which are trained on the training set images. This is to maximize the effectiveness and the threat of adversarial patches on the test set images. We use the same batch sizes and learning rates to train the CapsNet patches as we used to train the CapsNet models in Section 4. This is also the case for training the CNN patches on MNIST. However, we used a higher learning rate (0.01) to train the CNN patches on SVHN and CIFAR10, in order to reduce underfitting.

### 5.2 Evaluating the Patches

For each dataset, we separately trained the patches against the three architectures (CNN, CapsNet, and improved CapsNet) with different values of $\lambda_a$, using the optimization objective from Equation 6. Note that when $\lambda_a = 0$, the adaptive attack (Equation 6) is identical to the naive attack (Equation 4), and evaluating the adaptive attack with $\lambda_a = 0$ is as same as evaluating the naive attack. We thus highlight statistics for the $\lambda_a$ that results in the highest adversarial success rates (ASR) for each dataset, as well as the statistics for $\lambda_a = 0$ to compare naive and adaptive attacks, in Tables 4, 5, and 6.

**Adversarial Success Rate (ASR).** We define ASR as the proportion of adversarial inputs that both fool the classifier and evade the detector on test set images. ASR is an overall metric that can be interpreted as how effective the adversarial patch is against a protected model that has a detector.

**Adversarial Fooling Rate (AFR).** We also show AFR, the proportion of adversarial inputs that fool the classifier. AFR is only impacted by the effectiveness of the patch on the classifier and not the detector. AFR can be interpreted as how effective the adversarial patch is against the model unprotected by a detector.

**Adversarial Detection Rate (ADR).** We define ADR as the proportion of adversarial inputs that are detected as adversarial. ADR can be interpreted as the recall rate for the detector, the higher it is, the better it is for the detector.
Table 5: Evaluation statistics of patches on the SVHN dataset, randomly applied to 40% of input image area. Statistics for the naive attack ($\lambda_a = 0$) and the adaptive attack ($\lambda_a > 0$) with the best ASR are in bold text.

| $\lambda_a$ | CNN Test Acc.: 92.42% | CapsNet Test Acc.: 90.92% | Improved CapsNet Test Acc.: 94.8% |
|------------|---------------------|---------------------|---------------------|
|            | ADR | ASR | ADR | ASR | ADR | ASR |
| 0          | 0.248548 | 0.026148 | 0.019354 | 0.083528 | 0.147251 | 0.022895 | 0.606678 | 0.161973 | 0.065217 |
| 0.5        | 0.301668 | 0.034713 | 0.025983 | 0.854478 | 0.155528 | 0.025860 | 0.628376 | 0.167078 | 0.058753 |
| 1          | 0.253037 | 0.025942 | 0.019559 | 0.767181 | 0.156805 | 0.044472 | 0.534956 | 0.162055 | 0.076910 |
| 1.5        | 0.253572 | 0.025077 | 0.019106 | 0.467861 | 0.149146 | 0.083467 | 0.531538 | 0.156868 | 0.072587 |
| 2          | 0.257031 | 0.022112 | 0.017089 | 0.142598 | 0.065596 | 0.058596 | 0.498600 | 0.149457 | 0.073164 |
| 2.5        | 0.292938 | 0.032777 | 0.024707 | 0.175046 | 0.121762 | 0.105950 | 0.502470 | 0.172843 | 0.087800 |
| 3          | 0.290097 | 0.033560 | 0.025654 | 0.151081 | 0.110397 | 0.098744 | 0.435359 | 0.162097 | 0.089962 |
| 3.5        | 0.236813 | 0.020136 | 0.014577 | 0.107227 | 0.061561 | 0.057113 | 0.410985 | 0.150856 | 0.090909 |
| 4          | 0.252954 | 0.021289 | 0.016636 | 0.110397 | 0.079020 | 0.073461 | 0.368454 | 0.144928 | 0.093462 |
| 4.5        | 0.243031 | 0.025489 | 0.019106 | 0.110644 | 0.082232 | 0.076343 | 0.378047 | 0.152297 | 0.095850 |
| 5          | 0.242619 | 0.020877 | 0.016306 | 0.106280 | 0.077208 | 0.072102 | 0.330781 | 0.141881 | 0.097908 |
| 10         | 0.247643 | 0.030389 | 0.023760 | 0.083385 | 0.049043 | 0.046613 | 0.081316 | 0.063282 | 0.060647 |
| 100        | 0.128227 | 0.027754 | 0.024501 | 0.066337 | 0.025530 | 0.023965 | 0.056448 | 0.005517 | 0.005147 |
| 1000       | 0.082438 | 0.018571 | 0.016348 | 0.065761 | 0.023060 | 0.021618 | 0.056777 | 0.005147 | 0.004694 |
| 10000      | 0.079185 | 0.014906 | 0.013012 | 0.066543 | 0.022483 | 0.021124 | 0.056283 | 0.004488 | 0.004076 |

Since we are using targeted attacks with target class $t_{adv} = 0$, input images with a ground truth of $y = 0$ are not included in the analysis. The detection false positive rate is controlled at 0.05 per Section 2.

During evaluation, the patches are applied at random locations and rotations as they were during training, but they are scaled to cover 40% of the input image instead of a randomly sized area. The reason we use 40% instead of smaller sizes is that the patches are ineffective at fooling the classifier if scaled any smaller. To demonstrate this, we plot (in the Appendix) the ADR, ASR, and ASR at different scales for the naive ($\lambda_a = 0$) patch and most effective adaptive ($\lambda_a > 0$) patch for each detector on each dataset. We also show example patches in the Appendix as well.

6 EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we discuss the experimental results of adversarial detection on MNIST, SVHN, and CIFAR10 datasets, using the CNN and both the original and improved CapsNet. The effectiveness of adversarial patches against the classifier and the detector are displayed in Table 4, Table 5 and Table 6 for MNIST, SVHN, and CIFAR10 datasets, respectively. We will only analyze the bolded statistics, which correspond to the naive attack and the best adaptive attack for an architecture on a dataset, unless otherwise stated.

6.1 Results

Results on MNIST. We first observe that the overall fooling rates (AFRs) are quite low across the board at between 0.2-5.2%, leading to low attacker success rates (ASR) of between 0.1-0.6%. MNIST’s low resolution makes it difficult to craft successful attacks. In terms of detection (ADR), both CapsNets substantially outperform the CNN at detecting the attack, with over 81% ADR compared with just 7.8% ADR for the best adaptive attack. The improved CapsNet is slightly worse overall than the original CapsNet on both ADR and ASR, showing that the original CapsNet is better in detecting attacks, though it features lower AFR.

Results on SVHN. First, we note that our improved CapsNet outperforms CNN and the original CapsNet on benign test accuracy by more than 2% and 3.5%, respectively, showing the benefits of our modifications. Against both naive and adaptive attacks, however, the CNN offers very low fooling rates and thereby performs the best with just 1.9% ASR for naive attacks and 2.6% ASR for the best adaptive attack. The improved CapsNet has a significantly worse detection rate (ADR) than the original CapsNet, but it shows improvements against adaptive attacks with 33.1% ADR versus 17.5% ADR for original CapsNet.

Results on CIFAR10. On CIFAR10, the CNN offers the best benign test accuracy at 80.1%, 13.8% better than CapsNet and 4.6% better than the improved CapsNet. The fooling rate (AFR) is much higher on CIFAR10 than the previous two datasets at between 30-57% for the settings with the best ASR results. Here, the naive attack is best countered by the CNN at 23.5% ASR, but the adaptive attack is best countered by the improved CapsNet at 29.4% ASR, which is 6.2% and 17.8% better than the CNN and original CapsNet, respectively. The improved CapsNet’s performance on adaptive attacks is marked by both the highest detection rate and the lowest fooling rate of the three models.

6.2 Analysis

Does the Detection Work? First, we see that in every set of bolded statistics, including the adaptive patches, ASR is lower than AFR by a noticeable amount. This means that a smaller proportion of patched images both fool the classifier and evade detector, than fooling the classifier alone. In other words, some otherwise successful attacks are being detected by the architecture, for every naive attack and best adaptive attack on each dataset with each architecture. Had the detector been ineffective, we would see similar numbers for ASR and AFR.

This is most pronounced on the CapsNet architectures on MNIST (Table 4), where for the improved CapsNet, the best adaptive attack has ASR $\approx 0.005$ and AFR $\approx 0.025$, which means only around a
### Table 6: Evaluation statistics of patches on the CIFAR10 dataset, randomly applied to 40% of input image area. Statistics for the naive attack ($\lambda_a = 0$) and the adaptive attack ($\lambda_a > 0$) with the best ASR are in bold text.

| $\lambda_a$ | CNN | | | | CapsNet | | | | Improved CapsNet | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
|   | ADR | AFR | ASR | | | ADR | AFR | ASR | | | ADR | AFR | ASR |
| 0  | 0.308582 | 0.307248 | 0.234549 | | | 0.554024 | 0.564695 | 0.270565 | | | 0.404333 | 0.380444 | 0.247778 |
| 0.5 | 0.308359 | 0.304357 | 0.220542 | | | 0.56923 | 0.513673 | 0.229658 | | | 0.421778 | 0.352000 | 0.224333 |
| 1  | 0.278235 | 0.446532 | 0.342152 | | | 0.542019 | 0.587594 | 0.282125 | | | 0.380444 | 0.364222 | 0.246778 |
| 1.5 | 0.315362 | 0.424522 | 0.310805 | | | 0.46932 | 0.655069 | 0.373055 | | | 0.373444 | 0.353000 | 0.238000 |
| 2  | 0.307248 | 0.350823 | 0.257225 | | | 0.475767 | 0.517119 | 0.282237 | | | 0.345778 | 0.365000 | 0.259333 |
| 2.5 | 0.301356 | 0.467652 | 0.349378 | | | 0.439084 | 0.670187 | 0.397510 | | | 0.332222 | 0.365778 | 0.267000 |
| 3  | 0.269778 | 0.413851 | 0.317919 | | | 0.407514 | 0.601823 | 0.375500 | | | 0.301667 | 0.317333 | 0.242222 |
| 3.5 | 0.310805 | 0.394842 | 0.294020 | | | 0.400622 | 0.566363 | 0.354936 | | | 0.289667 | 0.248222 | 0.193667 |
| 4  | 0.294131 | 0.386060 | 0.291018 | | | 0.371276 | 0.672743 | 0.443753 | | | 0.284889 | 0.262222 | 0.203333 |
| 4.5 | 0.316696 | 0.393953 | 0.284349 | | | 0.361272 | 0.585038 | 0.384838 | | | 0.238000 | 0.238778 | 0.193667 |
| 5  | 0.293241 | 0.391285 | 0.295798 | | | 0.335816 | 0.630058 | 0.433859 | | | 0.252889 | 0.365000 | 0.294111 |
| 10 | 0.317363 | 0.413517 | 0.306025 | | | 0.187528 | 0.568253 | 0.472099 | | | 0.158111 | 0.287889 | 0.235556 |
| 100 | 0.243775 | 0.448755 | 0.355825 | | | 0.040907 | 0.057803 | 0.055024 | | | 0.037889 | 0.024667 | 0.023444 |
| 1000 | 0.078257 | 0.336483 | 0.316807 | | | 0.040907 | 0.049133 | 0.046354 | | | 0.037333 | 0.019000 | 0.017667 |
| 10000 | 0.048911 | 0.021787 | 0.020787 | | | 0.041018 | 0.047577 | 0.044242 | | | 0.039222 | 0.017000 | 0.016000 |

**Comparison among Datasets.** We can observe from Table 4, 5 and 6 that ASR across all the three architectures increases with the increase in dataset complexity. ASR on MNIST dataset is less than 0.01 whereas it is between 0.01 and 0.1 for SVHN, and between 0.23 and 0.47 for CIFAR10. AFR shows a similar trend in terms of higher data complexity and higher flooring rate, with AFR < 0.06 on MNIST, 0.02 < AFR < 0.17 on SVHN, and 0.30 < AFR < 0.57 on CIFAR10. ADR also follows the same pattern, the higher the data complexity, ADR tends to be lower in both unmodified CapsNet and improved CapsNet, and in CNN for SVHN and CIFAR10. These results suggest that adversarial patch attacks might behave similarly with adversarial examples in that they are harder to detect on more complex dataset.

MNST on CNN has a surprisingly low ADR of 0.08 against the best performing adaptive patch, and 0.18 against the naive patch. Two possible factors might have contributed to this anomaly. One is that we upscale the input images for the CNN on all datasets [224, 244, 3] to be consistent with [19], which causes it to perform worse than on the [28, 28, 3] MNIST dataset, compared to the [32, 32, 3] SVHN and CIFAR10 datasets. Another possibility is that since we use the same CNN architecture hyperparameters for all three datasets, the hyperparameters just happen to perform similarly in terms of ADR on the the [32, 32, 3] datasets, and worse on the [28, 28, 3] MNIST dataset.

### 7 DISCUSSION

Our detection approach only requires one forward pass without the time and memory requirements of back propagation and is faster than that of SentiNet [8]. This allows safety-critical systems to rapidly sound the alarm when an attack is taking place. Nevertheless, for our approach to be used in real world computer vision systems, reconstruction networks needs to be able to reliably scale up to more complex datasets such as ImageNet. On the bright side, we showed that a detection method originally proposed against digital adversarial examples can be effective against adversarial...
patches with some modifications, and other perturbation detection methods are worth a try to mitigate risks brought on by physical adversarial attacks.

We also find that the distribution of reconstruction errors tends to have higher means with larger variance on CIFAR10 and SVHN than on MNIST. This could be due to the fact that CIFAR10 and SVHN are non-segmented images containing background noise or even multiple classes of objects. If this is indeed the case, then the architecture needs to learn to segment the background noise from the class objects to more effectively detect adversarial attacks on more complex datasets. This might be achievable through new architectures that can learn segmentation in an unsupervised manner, or supervised training on image segmentation. We leave this investigation as a part of future work.

8 CONCLUSION

We have shown that capsule networks can be improved using matrix-capsule-dropout and affine voting, techniques inspired by convolutional neural network, and that adversarial digital perturbation detection techniques can be used against adversarial physical payloads with some success. This is promising for the small but growing literature on capsule networks and defenses against adversarial physical attacks, as advances in adjacent but more substantial fields (convolutional neural networks and adversarial digital perturbations, respectively) may be transferred over.

REFERENCES

[1] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. 2017. Synthesizing robust adversarial examples. arXiv:1707.07397 (2017).
[2] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases.
[3] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. 2017. Adversarial Patch. arXiv:1712.09665 (2017).
[4] Nicholas Carlini and David Wagner. 2017. Adversarial examples are not easily detected: Bypassing ten detection methods. In ACM Workshop on Artificial Intelligence and Security.
[5] Nicholas Carlini and David Wagner. 2017. MagNet and 'Efficient Defenses Against Adversarial Attacks' are Not Robust to Adversarial Examples.
[6] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In IEEE symposium on security and privacy (S&P).
[7] Shuang-Tse Chen, Cory Cornelius, Jason Martin, and Duen Horng Chau. 2018. Robust Physical Adversarial Attack on Faster R-CNN Object Detector. ArXiv:1804.05810 (2018).
[8] Edward Chou, Florian Tramér, Giancarlo Pellegrino, and Dan Boneh. 2018. Sentinel: Detecting physical attacks against deep learning systems. arXiv:1812.06292 (2018).
[9] Reuben Feinman, Ryan R. Curtin, Saurabh Shintre, and Andrew B. Gardner. 2017. Detecting Adversarial Samples from Artifacts.
[10] Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are we ready for autonomous driving? the kitti vision benchmark suite. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
[11] Andy Greenberg. 2018. Hackers Remotely Kill a Jeep on the Highway.—With Me in It. https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.
[12] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. 2017. On the (statistical) detection of adversarial examples. arXiv:1702.06280.
[13] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition. The shared views of four research groups. In IEEE Signal Processing Magazine.
[14] Geoffrey Hinton, Sara Saurab, and Nicholas and Fross. 2018. Matrix capsules with EM routing. In International Conference on Learning Representations (ICLR).
[15] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D Wang. 2011. Transforming auto-encoders. In International Conference on Artificial Neural Networks.
A PATCHES AND STATISTICS
VISUALIZATION

This section contains visualization of patches and their adversarial statistics, by the architecture and dataset they were optimized on.
Figure 6: Naive Adversarial Attack.

Figure 7: Best Adaptive Adversarial Attack.