Adversarial Robustness via Biologically Inspired Mechanisms

Maya Burhanpurkar

December 6, 2021

CS 91r

Contents

1 Introduction

Before my eyes can focus after waking up to an alarm clock, my phone, which has just finished charging in accordance with what it has learned about my sleeping pattern from my usage data, scans my face to verify my identity before unleashing a slurry of notifications from the Financial Times informing me of Germany's new climate-focused lending policies and Adani's acquisition attempt of SoftBank's solar division, among other news. In the span of mere seconds, at least three machine learning algorithms have made my life easier.

Despite having owned my phone's lithium-ion battery for years and irresponsibly charging it overnight each day, its capacity has remained virtually unchanged since the date it was purchased thanks to Apple's Optimized Battery Charging machine learning algorithm that digests my years of usage data to predict when I will sleep and when I will wake up and schedule charging during those intervals to maximize efficiency and battery life [2]. Apple has likewise developed such robust machine learning algorithms for face detection and identification that they have begun to use it for device security, enabling me to unlock my iPhone with a glance [1]. Finally, my Harvard-sponsored news application has noticed my penchant for climate-related news based on my browsing data and has tailored notifications for me first thing in the morning.

Machine learning based algorithms make countless decisions for us daily, and are being used in ever increasing mission critical applications. These algorithms are used to process data from the cameras of self-driving cars to identify which obstacles are pedestrians and which are cars [6], to determine whether and at what rate mortgages should be awarded [12], and to determine whether children should be separated from their parents [5]. Largely seen as dispassionate arbitrators of vast volumes of data, their shortcomings are known to few and studied by fewer. Recent work has demonstrated the susceptibility of convolutional neural networks to adversarial attacks, extreme sensitivity to small perturbations, among many other undesirable properties. With such techniques rapidly diffusing into more of our

daily lives, addressing these concerns has become critical.

In this work, we focus on the problem of adversarial robustness. Adversarial examples are inputs to neural networks that are almost identical to real inputs with the addition of imperceptible but precisely calibrated noise that can cause machine learning algorithms to fail catastrophically [21]. By using the very algorithm that trains machine learning models against the models themselves, such inputs can be constructed with damaging consequences. Pandas being confidently misclassified as gibbons in the ImageNet dataset can easily become pedestrians being misclassified as the open road for self-driving algorithm image processing [25]. The human visual cortex, however, does not appear to be susceptible to such perturbations. This motivates the notion that making biologically inspired modifications to convolutional neural networks could help increase robustness to such adversarial inputs, ideally without sacrificing accuracy.

We extend the work of Vuyyuru, Banburksi, Pant, and Poggio in [22], which examined the use of several biological mechanisms for enhancing adversarial robustness, including coarse, retinal, cortical sampling of images. We build on their implementation by augmenting their sampling techniques with algorithmic selection of sampling points by the use of parallel spatial transformer networks and multi-gaze models. We further implement a variety of performance enhancements in the hopes of improving adversarial robustness and decreasing computational expense. The current state of the art in enhancing adversarial robustness—adversarial training—is both computationally expensive and restrictive in that it only protects against the specific adversarial attacks it has been trained against, and is therefore only a defensive mechanism, not one that can proactively protect machine learning algorithms from bad actors. By mimicking the brain, we can offer more robust proactive protection against such adversarial attacks for a fraction of the cost.

The retinal sampling technique offers improved standard performance on ImageNet10 while also improving adversarial robustness. Moreover, both the parallel spatial transformer

network and multi-gaze networks examined in this work offer improvements to adversarial robustness while roughly preserving standard classification performance. While these approaches still do not offer the level of robustness to specific adversarial attack as adversarial training, they instead demonstrate the practicality of *real-time* and *lightweight* robustness techniques.

The subsequent report is organized as follows. We review the fundamentals of machine learning and explain how they enable the generation of adversarial examples as well as summarize the techniques, standard and biologically inspired, for combating them. We then detail the experiments performed as a part of this work—namely, the addition of parallel transformers, attention mechanism, and multi-gaze mechanisms as well as several performance enhancements—and finally conclude with a discussion of the results and their significance.

2 Background

In this section, we review the fundamentals of machine learning and explain how they enable the generation of adversarial examples. Finally, we summarize the techniques, standard and biologically inspired, for combating them.

2.1 Neural Networks

The field of **machine learning** can be partitioned into a variety of sub-classes of paradigms, including supervised, semi-supervised, unsupervised, and reinforcement learning. Several other classes of problems, including dimensionality reduction, anomaly detection, representation learning etc. can also fall under the umbrella of machine learning.

Artificial **neural networks** (ANNs) describe a class of functions typically used in the context of **supervised machine learning**. Supervised machine learning describes the process of learning a function that maps inputs to outputs consistent with user-specified data. Consider the example of image classification: a supervised approach to the problem would be to one in which a learning algorithm is given a set of potential input images as well as their labels. Based on its understanding of these input-output pairs, the algorithm should learn how to map previously unseen inputs to reasonable outputs. For example, given many images of cats and their corresponding label, a successful learning algorithm will be able to take in a new image of a cat and output the correct label ("cat").

ANNs describe a specific family of function that can be used in such a setting. ANNs consist of neurons that are connected to one another by weights and biases. First, an input to the function is flattened. For example, to classify a $(320, 320, 3)$ RGB image, the input should be flattened into a length $320 \times 320 \times 3$ array. This input is then passed to the input neurons of the network, which make up the **input layer**. The number of input neurons should be exactly equal to the length of the input to the function. Following the input layer are a series of **hidden layers**. These layers each have some number of neurons that

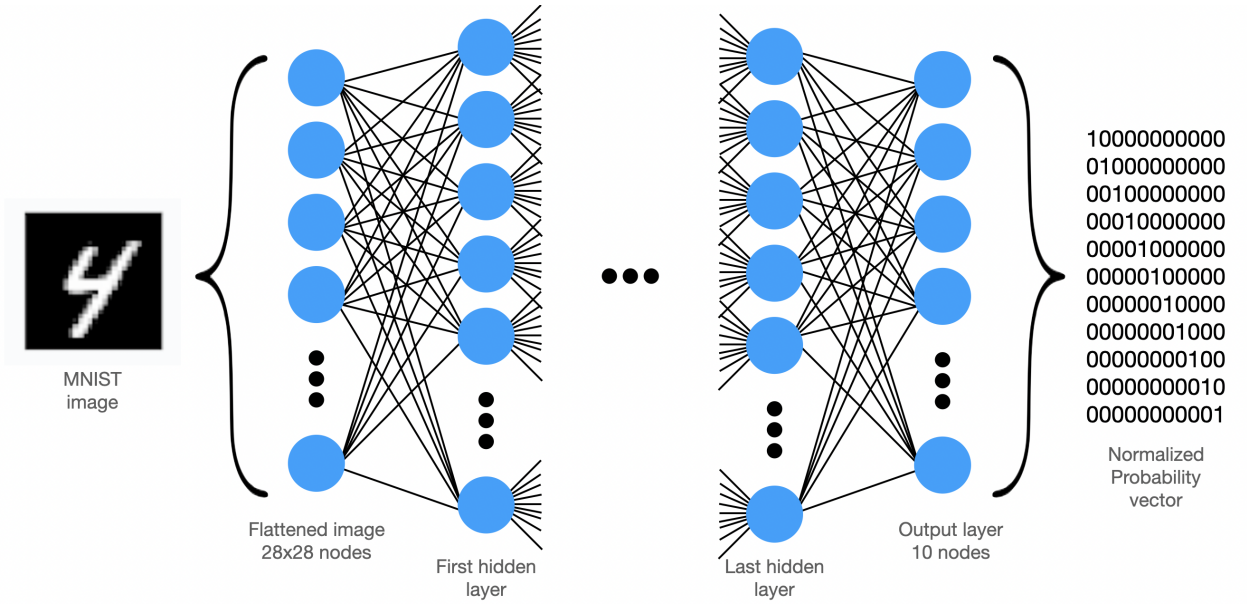

Figure 1: Information flow in a typical ANN. The input image is flattened, and each pixel value is given to its respective input node. Black connections between nodes represent weights while nodes themselves can have biases (additive weights) and activation functions (nonlinearities). The value at a node i, v_i, is the value of the activation function applied to the weighted average of the values of the nodes that flow into node i, plus node i's bias term. The final output layer has, in this case, 10 nodes, one corresponding to each possible input digit. If the output nodes have a softmax as the activation function, these outputs can represent a normalized probability vector. The weights and biases in this network must be trained with an optimization algorithm like Stochastic Gradient Descent (SGD).

are typically connected only to neurons in the layers immediately preceding them. Such networks are called **feed-forward**. The connections can have weights associated with them, such that if neuron a is connected to neurons b, c, and d, the value of neuron a is given by a linear combination of the values at b, c, and d, plus some bias associated with the neuron a, and a final function, called an **activation function**, that is applied to the resulting number. This function is often **non-linear**, allowing for the ANN as a whole to learn complex functions of its inputs. This structure continues until we reach the **output layer**. In an image classification problem, for example, the output layer may consist of n neurons, each representing the n possible input classes of the images. This structure is demonstrated in Figure 1.

Already, the complexity of ANNs should be apparent. Other than the dimensions of the

input and output layers, which are constrained by the task at hand, the dimensions of all layers in the ANN are variable, as are the weights and biases associated with each neuron, the kind of non-linearity present at each neuron, and the number of hidden layers themselves. It is this flexibility that enables ANNs to be so powerful and represent such a large class of functions.

Many more specialized formulations of ANNs exist that have been developed for particular tasks. Of particular relevance to the problem of image classification are **convolutional neural networks** (CNNs). Rather than immediately flattening input images, their structure is preserved by the input nodes, such that each layer is two dimensional rather than one dimensional. To traverse from one layer to the next, learned kernels of fixed size, for example 3×3 pixels, are convolved with the image at the previous layer. This convolution process is illustrated in Figure 2. For images in particular, these convolutional kernels offer the advantage that they preserve the structure in images because they are invariant to small shifts in the inputs. These convolutional filters can be stacked to form the deep convolutional networks common today, as shown in Figure 3.

The structures of ANNs and CNNs alone cannot enable machines to learn. The myriad weights and biases must be tuned to the particular learning task at hand. As in a least-squares regression, this can be accomplished by defining a **loss function** to be minimized on the training data. There are a variety of common loss functions used in the training of neural networks, including the mean squared error itself. For the task of image classification, the crossentropy, defined as

$$\mathcal{L}(X, y) = -\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij} \log[p_\theta(X_i)_j]$$

for n training examples and m classes, where p_θ is the current predictor, X_i is the i input example, and y_{ij} is the jth entry of the label corresponding to the ith example. Note that,

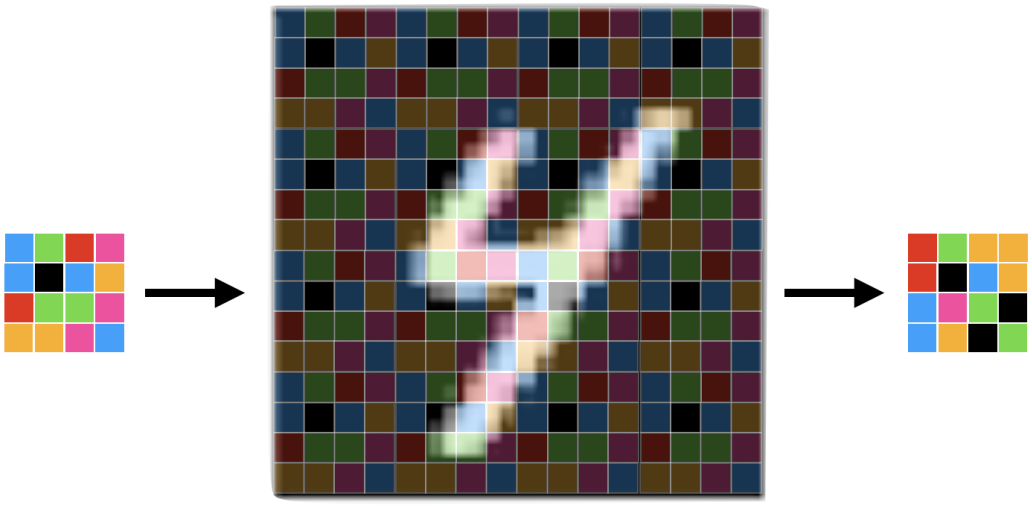

Figure 2: An example of a convolution kernel applied to an MNIST image with a stride equal to the kernel width. The leftmost grid represents a convolutional kernel, which is an array of weights. These weights are then convolved with an input image with a particular stride value. Each convolution becomes a pixel in the output (the rightmost kernel). For example, a stride of 1 means that each the image is only reduced the kernel width after the convolution is applied. In this case, a stride of 4 is used on a $4x \times 4$ kernel. Since the input was 16×16, this resulted in a 4×4 output.

because we require $\sum_j p_\theta(X_i)_j = 1$ for all i[1], if $y_{ij} = 1$ (i.e. sample i has label j) then $-\sum_j \log[p(x_i)_j]$ is minimized if $p(x_i)$ is a one-hot vector with the element at index j equal to 1. This is precisely what we would hope for—given that we have inputted an image X_i of a dog, we would hope that $p_\theta(X_i)$ returns 1 for the "dog" output neuron and 0 for everything else, which is what is encoded by y_i. The mean squared error is another common choice of loss function

$$\mathcal{L}(X, y) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} [p_\theta(X_i)_j - y_{ij}]^2$$

The goal of a learning algorithm is to minimize this loss by tuning the weights and biases that must be tuned in the corresponding ANN. In the absence of analytical techniques for the fitting of neural networks to data, numerical methods are employed. Today, the standard pair of algorithms for training ANNs are stochastic gradient descent (SGD) and its variants

[1]This is typically accomplished by applying a softmax, defined as $\sigma(x) = \frac{e^{x_i}}{\sum_i e^{x_i}}$, as the activation function for the output layer.

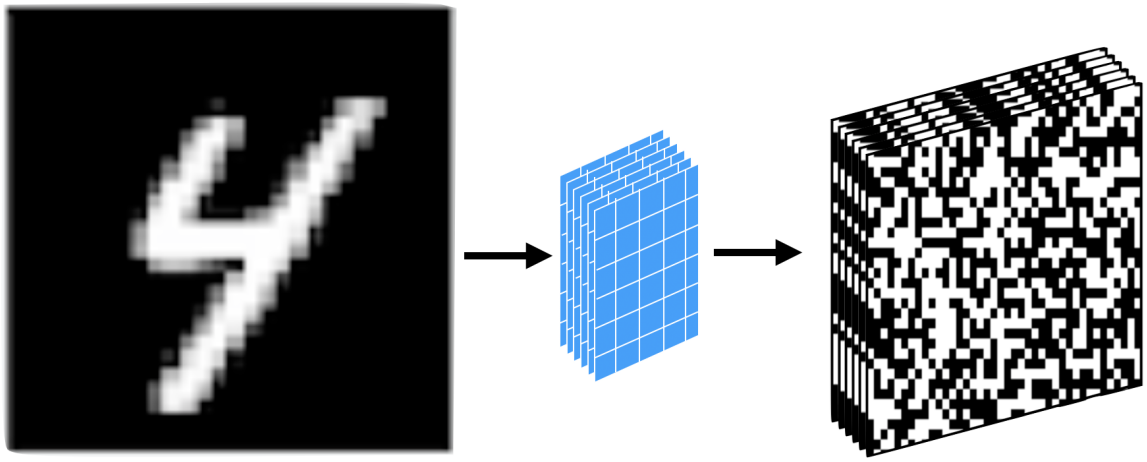

Figure 3: The convolution operations from Figure 2 can be stacked to for a full Convolutional Neural Network. Starting from the input image, suppose that we apply n convolutional filters. This will generate n new secondary images. We can subsequently apply m more filters to these n resulting images, generating nm new images. This process can repeat until we achieve the desired number of filter at the desired scale. Note that with each subsequent layers of filters, we extract higher-level features (i.e. those at larger scales). Since the filter layers result in such a large number of output pixels, several dense layers can be employed before the final output layer of the network.

in conjunction with the backpropagation algorithm.

SGD is a heuristic algorithm for finding the minimum of a function. **Gradient descent** describes the process of starting from some point in the parameter space and iteratively taking steps in the direction opposite the gradient of the function at that point. Through this process, each step should move the weights closer to a local minimum of the function. In the case of an ANN, we may initialize the weights and biases of the network to random values. Starting from this random point in the parameter space of the network, we evaluate the network on the training data and compute the loss. Using this loss, we determine the gradient of the parameters of each layer of the network with respect to these inputs and move in the opposite direction. Formally, for the weights at layer n, each step of gradient descent dictates that we update

$$w_n \leftarrow w_n - \epsilon \frac{\partial}{\partial \mathcal{L} w_n}$$

for some choice of the parameter ϵ. This process should be continued until convergence. SGD simply describes splitting the training data into mini-batches and computing the loss with respect to these mini-batches instead of the whole dataset each time. This is a practical adjustment designed to accommodate limited computer memory and causes increased variability in the gradient descent algorithm as some mini-batch losses may not be representative of the overall loss.

The **backpropagation** algorithm is a dynamic programming solution to the problem of computing the gradients required for SGD. Note that we can use the chain rule to compute the derivative of the loss with respect to any particular neuron. Using the example of the L2 loss, let $y_n = \sigma(a_n)$ where σ is our choice of activation function while $a_{n+1} = w_n y_n$. That is, y_i represents the output vector at layer i of the ANN, w_i is the corresponding weight matrix between layers i and $i+1$ and a_i is the input to layer n. Then, the error gradient of the

input layer at the output layer of the network can be computed as

$$\delta_N = \frac{\partial \mathcal{L}}{\partial a_N} = 2(y_N - y)\sigma'(a_N)$$

by the chain rule. Then, for any inner layer,

$$\delta_n = \frac{\partial \mathcal{L}}{\partial a_n} = \frac{\partial \mathcal{L}}{\partial a_{n+1}} \frac{\partial a_{n+1}}{\partial a_n} = \delta_{n+1} \frac{\partial w_n y_n}{\partial a_n} = \delta_{n+1} \frac{\partial w_n y_n}{\partial y_n} \frac{\partial y_n}{\partial a_n} = \delta_{n+1} w_n \sigma'(a_n)$$

so for a particular weight matrix w_n

$$\frac{\partial \mathcal{L}}{\partial w_n} = \frac{\partial \mathcal{L}}{\partial a_{n+1}} \frac{\partial w_n}{\partial a_{n+1}} = \delta_{n+1} \frac{\partial w_n y_n}{\partial w_n} = \delta_{n+1} y_n$$

This can continue throughout the network until we reach the input neurons. By computing the gradients "backwards" (i.e. starting from the output neurons), we can use the gradients from the previous layers to compute the gradients in the current layer. This is considerably more efficient than computing gradients directly for each neuron in the more intuitive forward direction. Thus, for SGD, we want to shift each weight matrix according to

$$\Delta w_n = -\epsilon \frac{\partial \mathcal{L}}{\partial w_n} = -\epsilon \delta_{n+1} y_n$$

In practice, there is far more to effectively training ANNs for supervised tasks than just SGD and backpropagation alone. Because SGD is prone to getting stuck in local minima, many modern loss function optimizers like Adam and RMSProp have "momentum" parameters that allow the algorithm to hop out of local minima [10]. Tuning such parameters require care, and the choices of ϵ and momentum are often decayed over the course of training with heuristic schedules. Furthermore, given the large capacities of modern day neural networks, overfitting to the input data can pose a challenge. Thus, further parameter tuning with weight regularization parameters and dropout parameters is necessary. Finally, the choice of

architecture itself is a hyperparameter. Practitioners are free to select the number of neurons in each layer, the depth of the network, and activation functions.

2.2 Adversarial Attacks

In a landmark paper in 2013, Szegedy *et al.* showed that the very SGD used to make neural networks so powerful can be turned against them [21]. In their paper, they defined the notion of the robustness of a model as the average size of the minimum perturbation to an input image x that causes a misclassification. That is, $R = \langle \rho(x) \rangle_x$ where $\rho(x) = \min_\delta d(x, x + \delta)$ such that $x + \delta$ is misclassified. The choice of distance metric d and minimizer depend on the application, but the original paper used the L2 norm and box-constrained L-BGFS.

The experimental work in [21] demonstrated the practical severity of the problem—by adding noise to images that is imperceptible to the human eye, a well-trained classifier in terms of test set accuracy could be made to output nonsensical predictions. Several examples of this effect are shown in Figure 4. A classifier was trained on ImageNet10 to 91% accuracy and the unperturbed images used to generate the samples in the figure were correctly classified.

The finding was remarkable in that it contradicted the prevailing assumption that deep neural networks can encode non-local generalizations over the input space, that is, that they can be used on regions of the input space that they have not been trained on. The work of Szegedy *et al.* demonstrated that indeed deep neural networks lack even *local* generalization abilities, let alone non-local generalization abilities. The human brain possesses extraordinary generalization ability: show a child an image of a giraffe and they will be able to recognize new giraffes in a zoo. It is not so for neural networks, throwing into question both our increasingly heavy reliance on them for ever more complicated tasks as well as their suitability as a model for the brain.

Since the publication of [21], there has been a great deal of further research into techniques for evaluating adversarial robustness and generating the adversarial attacks themselves. For

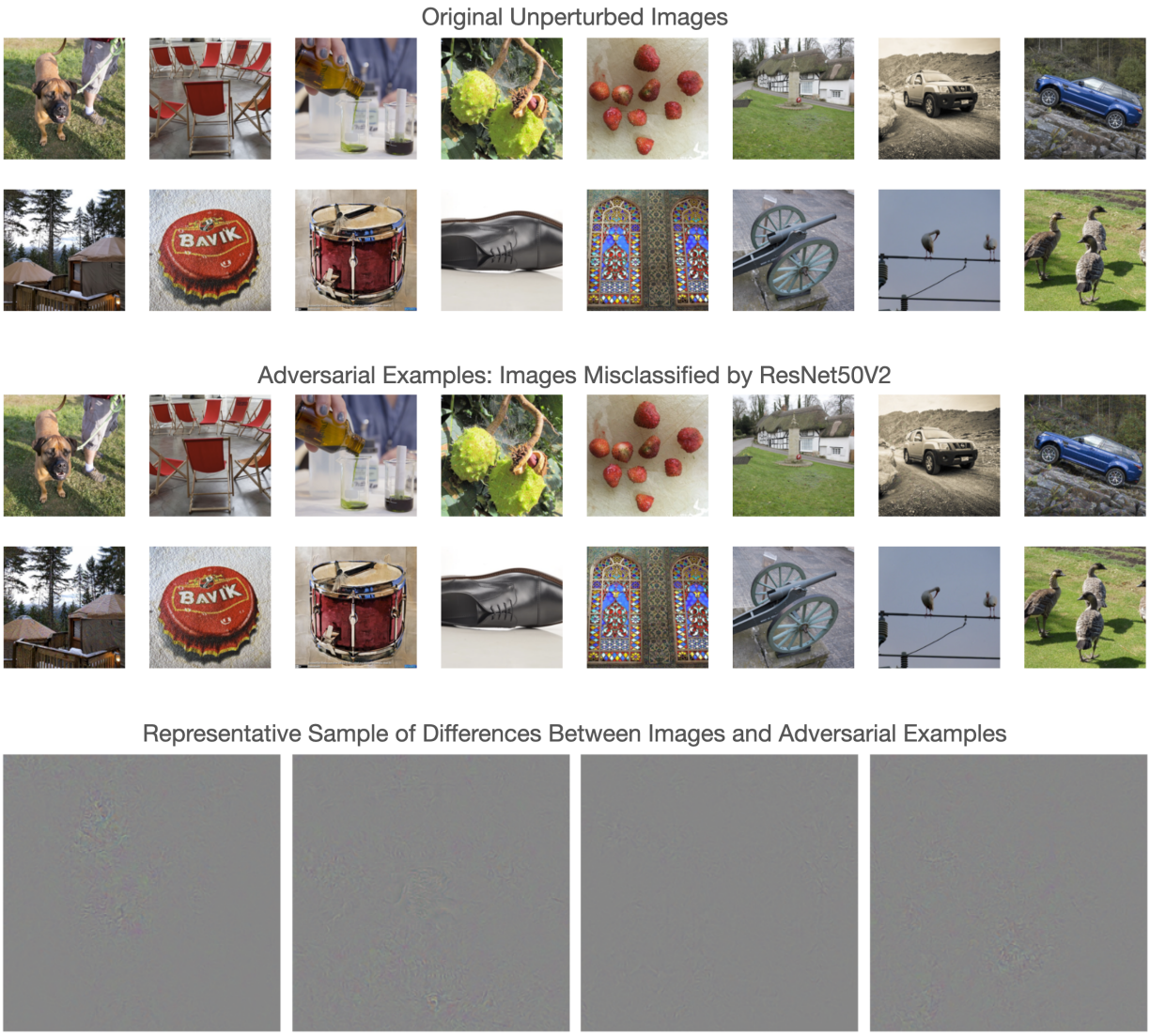

Figure 4: Top: Images from the test set that were classified correctly by the pre-trained ResNet50V2 from Tensorflow. Middle: Images generated by the L2 Carlini-Wagner Attack that result in misclassifications in the pre-trained ResNet50V2. These images are visually indistinguishable from their counterparts in the top row. Bottom: Four examples of the pixel difference between the original and adversarial examples plotted on the same scale as the original images. Adversarial examples were generated using the Foolbox library following the tutorial here.

example, [17] showed that using the definition of robustness given in [21] that relies on the effectiveness of an approximate minimizer can result in misleading robustness scores. A more successful strategy for evaluating robustness is to attack the model in question with many adversarial attack strategies and use the minimum successful perturbations across the strategies [20]. There are a wide variety of attack models in use today. The techniques can be classified into one of several categories: gradient-based attacks, score-based attacks, and decision-based attacks.

Gradient-based attacks use SDG on trained models to generate adversarial attacks. That is, for a fixed model and input x, we approximate

$$\mathcal{L}(x + \rho, y) \approx \mathcal{L}(x, y) + \rho \nabla_x \mathcal{L}(x, y)$$

That is, we can compute the gradient of the model with respect to a reference label y_0 and then move x *in the direction of the gradient* (rather than opposite it, as usual) by the minimum amount that leads to a misclassification [20]. This approach can be generalized in a variety of ways, such as by allowing the algorithm to take several steps in the direction of the gradient iteratively, using the gradients to compute saliency scores and changing the feature corresponding to the maximum saliency, and approximating the classifier as a linear function, computing the minimum distance to a class boundary, and taking a step in the direction of the corresponding class [15].

In cases of non-differentiable models, **score-based attacks** can be employed. Instead, gradients can be computed directly on the probabilities or scores outputted. For example, in a single-pixel attack, a *single* pixel value is multiplied by a perturbation parameter p, causing a misclassification [16]. This method was found to be alarmingly effective on CIFAR10, with 80% of pixels in an image being "critical" to the classification with $p = 100$. This method can be modified into a local-search attack by applying large perturbations to each pixel in an image to determine the sensitivity of each pixel to the classification. It then iteratively

perturbs pixels in the order of significance until the image is adversarial [16].

Finally, **decision-based attacks** utilize only the class labels outputted by models to design adversarial inputs. These attacks are particularly relevant for real-world situations in which attackers may not have oracle access to model gradients and output probabilities. In a boundary attack, we begin with an adversarial example, which is simple to find by initializing to noise or simply an image from another class in the training set and with each iteration of the training, noise from a proposal distribution is added to the image at that iteration provided that the distance to the target image is reduced and that the proposed image is adversarial, otherwise the perturbation is rejected [3]. The proposal distribution is selected such that the perturbed image is from the correct domain (e.g. each pixel is in [0, 255]), the L2 norm of the perturbation is within some pre-defined δ, and the perturbation reduces the distance between the perturbed image and the target image. In practice, this can be approximated by sampling from a Gaussian that is rescaled such that the first and second criteria are satisfied [3]. There are many simpler forms of decision-based attacks, including Gaussian blur attacks, salt and pepper noise attacks, additive uniform noise attacks, and contrast reduction attacks in which the named noise is added to the input image with minimal parameter as determined by a line search [20].

The descriptions above are not intended to be exhaustive, but rather a representation of the diversity of attacks that have been devised since the original work in [21]. The results of variety of attack methods from the Foolbox library employed on the same model as used in Figure 4 on an image taken from ImageNet10 are shown in Figure 4.

2.3 Adversarial Training and Other Robustness Techniques

With such an abundance of available adversarial attacks and efficient libraries like Foolbox that implement them, much attention has been devoted to defending against adversarial attacks. The main categories of defences are defensive distillation, feature squeezing, defensive randomization, and adversarial training.

The most successful defense found so far has been **adversarial training** [11]. The idea of adversarial training is to add adversarial examples to the training data, continually generating new adversarial examples as the training progresses. While highly effective for one-step methods, the major drawbacks to adversarial training are that it requires both generating adversarial examples and retraining the classifier using the examples, effectively doubling the total time to deploy a model, while also being relatively ineffective at iterative attacks [11]. Furthermore, it cannot be done in real-time as a preprocessing based technique can be. Thus, there has been considerable attention given to combining adversarial training with other defense mechanisms in the literature.

A natural defence against gradient-based attacks is to mask the gradients, which can be accomplished via **defensive distillation**. Defensive distillation is conceptually derived from knowledge distillation, a technique for compressing neural networks [8]. In knowledge distillation, an ANN is first trained as a classifier with a softmax at high temperature in the final layer on the binary labels. The output probability vectors are then used to train a second ANN with a smaller architecture at the same temperature. At test time, the temperature is set back to 1. In defensive distillation, the procedure is tweaked slightly in that the second ANN has the same architecture as the first. Accuracy is preserved for the the same reason as in regular distillation and adversarial robustness is increased because the training decreases the gradients of the network, making it more difficult to craft adversarial examples [19]. The authors of the technique concluded, however, that gradient masking will not generally be effective at protecting against adversarial examples because of their transferability [18].

One can also take a classification approach to the adversarial examples problem. In **feature squeezing**, a model is trained to identify adversarial examples [24]. The guiding principle is that because of the large feature spaces of inputs to classifiers relative to the classification problem at hand (e.g. to recognize a CIFAR10 image of a dog, we require at least $32 \times 32 \times 3 = 3072$ input nodes), generating adversarial examples is simple because the domain size permits many. If the output of the model evaluated on both inputs is

substantially different (determined by an empirical threshold), the input is likely adversarial. Examples of such squeezing techniques include reducing the color depth of each pixel in an image and using spatial smoothing to reduce the differences among individual pixels [24]. This technique can easily be composed with other techniques, because it does not involve changing the underlying model. Like defensive distillation, feature squeezing does not completely eliminate adversarial examples, and the authors showed that some examples slipped through the network undetected.

Another input preprocessing based approaching is **defensive randomization** [23]. This technique relies upon random resizing and random padding inputs before entering the classifier at test time. As with feature squeezing, defensive randomization should be used in conjunction with adversarial training to guarantee high robustness to adversarial attacks. The authors demonstrated that defensive randomization offers additional protection to iterative adversarial attacks while adversarial training offers additional protection from single step attacks.

2.4 Biologically Inspired Adversarial Robustness Techniques

While ANNs were originally designed with the brain's neurons in mind, there are several salient differences between the processing of data in a CNN and the human visual cortex. Primate eyes have fovea that sample different spatial regions of inputs at different resolutions. Eyes also have attention mechanisms that enable the brain to focus on the most important spatial regions of an input. The eyes are rarely static, instead exploring spatial inputs in a fixational way. Finally, the visual cortex has many feedback and top-down recurrent connections [26].

Recurrent attention networks: In [26], a sequential attention mechanism was investigated as a tool for improving adversarial robustness. The network consisted of an initial ResNet[2] whose outputs are repeatedly processed by an LSTM that recommends a variety of

[2]ResNets are "residual networks" that consist of stacked CNN layers with "skip connections" that allow

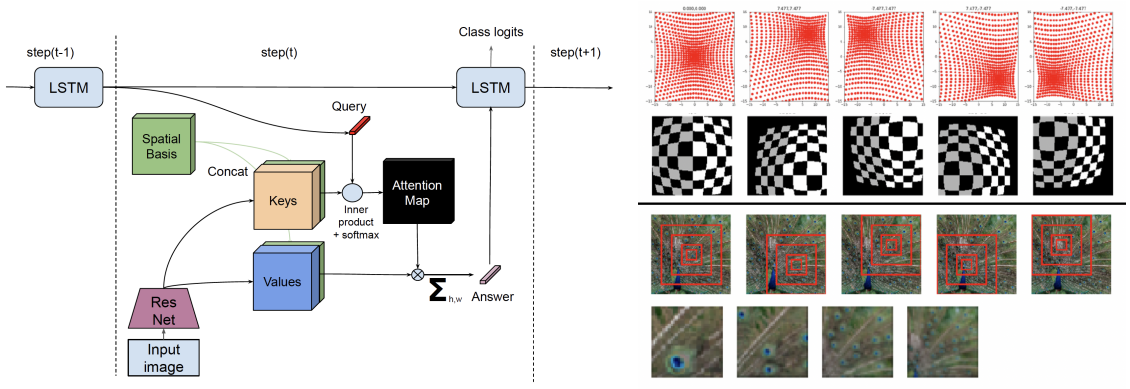

Figure 5: Left: The architecture for the recurrent attention model in [26]. An input image is given once to a ResNet before an LSTM performs repeated attention sampling. Right (top): Retinal fixations showing foveal sampling at five fixation points in red and the corresponding transformation on a checkerboard. Right (bottom): Cortical fixations at five sampling points at four scales and an example of the four scale images that would be simultaneously processed by the ResNet in [22].

attention maps. These new images are then given back to the LSTM to generate subsequent images, as well as the output logits. This process is shown in Figure 5 (left). The authors demonstrated that this model offered a substantial robustness improvement over a standard ResNet, and the strength of the robustness increased with the number of attention steps.

Foveation-based mechanisms: In [13], the impact of foveation on adversarial robustness was evaluated. The authors define foveation as a transformation that performs a crop, leaving only part of the original input to be processed—that is, changes of position and scale. By doing minimal crops and resizes, the accuracy of models can be approximately preserved while adversarial robustness is improved.

Retinal and cortical fixations: Finally, in [22] the relevance of cortical and retinal fixations to classification accuracy and adversarial robustness were evaluated. These are demonstrated in Figure 5 (right).

Retinal fixations describe the non-uniform sampling of inputs performed by retinal photoreceptors. The density of cones exponentially decreases with eccentricity, resulting in the

for data to skip convolutional blocks [7]. In this way, the identity function is easily learnable by the network, resulting in the function spaces of ResNet-xs being strictly larger than the function spaces of ResNet-ys for $x > y$. ResNets offer state of the art classification on standard machine learning image classification datasets.

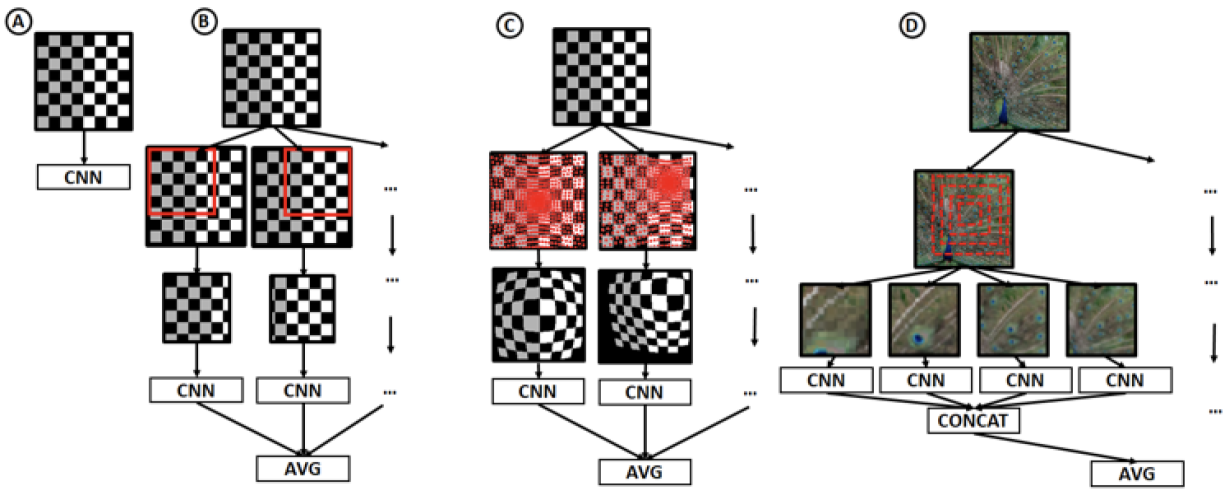

Figure 6: The four models tested in [22]. Model A is a standard baseline CNN, for which they used a ResNet18. This is the "baseline model". Model B has five branches, each of which perform a random crop before five ResNets which each share the same weights. During training, only one branch is used. This is the "coarse fixations" model. Model C has five branches and was trained similarly to Model B, but instead of random crops, random retinal fixations were applied. At test time, the retainal fixations were applied in five fixed locations (the four corners and the centre). This is the "retinal fixations" model. Finally, model D has five branches and applies "cortical fixations", resulting in each scale being passed to its own ResNet. The number of parameters in these ResNets were adjusted such that Model D has the same number of weights as the other three models.

fish eye effect in Figure 5 (top right). Retinal sampling in [22] was implemented by mapping the image to polar coordinates centered at one of five pre-defined fixation points and resampling according to r. This is similar to the approach in [4], with the accommodation made for multiple non-central fixation points.

Not only does the resolution vary with eccentricity—so too does the size of the receptive field, enabling for scale and translation invariant processing. This is implemented as cortical fixations in [22], in which four sampling scales are used at each fixation point. This is shown in Figure 5 (bottom right).

Four models were compared in [22]: a baseline ResNet-18 classifier, a branched coarse fixations model with random cropping centered at five pre-defined fixation points (four corners and the center), a retinal fixations model with retinal sampling centered at the five fixation points applied before being passed to separate ResNet branches, and a cortical fixations

model in which four scales are sampled at each of the five fixation points and each sampled scale is passed to a separate ResNet. In the cases of the coarse, retinal, and cortical fixations models, the five parallel outputs were concatenated before being passed to a single dense layer for classification. At training time, the model randomly trains on a single fixation from all possible fixations and an auxiliary loss is trained at the end of each branch of the network, but at test time, all fixation points are evaluated. These three models are shown in Figure 6.

The authors found that coarse and retinal fixations have almost no impact on accuracy as compared to the baseline ResNet with the same number of parameters, and in the case of ImageNet actually improved classification accuracy, while cortical fixations resulted in a modest (1-3%) decrease in classification accuracy. Moreover, each fixation model offered increased adversarial robustness over the baseline classifier for ImageNet and its variants, with coarse and cortical fixations performing comparably and retinal fixations outperforming both.

The significance of this result is twofold. First, the authors showed that it is possible to offer adversarial robustness at *no computational cost* (all four models shared the same number of parameters) while *preserving model fidelity*. Second, the enhancements to adversarial robustness and baseline classification accuracy offered by the fixation models lend continued support to the hypothesis that ResNets offer a reasonable model for the visual cortex.

3 Experiments

This project uses the work of [22] as a baseline and aims to address several areas of potential improvement. Specifically the goals of the subsequent experiments are threefold:

1. To improve upon the adversarial robustness provided by retinal fixations by augmenting the existing transform with bilinear sampling and a revised soft attention model.

2. To enable intelligent attention mechanisms via spatial transformer networks and multi-gaze mechanisms, improving on the fixed attention model used in the paper.

3. To explore potential performance enhancements to offset the computational and energy cost associated with the extensions to the standard ResNet required for adversarial robustness.

Goal 1 is addressed in §3.2 and §3.3, goal 2 is addressed in §3.3, and goal 3 is addressed in §3.4. We begin with §3.1 by reproducing the key results of [22] and discussing relevant aspects of their experimental setup.

3.1 Foundations and Baselines

The source code used in [22] can be found here. We begin by recapitulating the most significant elements of their experimental setup, and confirming that their results can be reproduced. Although the authors evaluated robustness on CIFAR10, ImageNet10, ImageNet100, and ImageNet, we will focus our attention on ImageNet10, which enables probing the effectiveness at resolutions higher than those available in CIFAR while avoiding the computational constraints associated with training on the full ImageNet set. We use central cropped and (if necessary) bilinearly upsampled $320 \times 320 \times 3$ images.

The ImageNet10 dataset is defined as {n01742172, n02099712, n02123045, n01644373, n01665541, n01855672, n02510455, n01484850, n01981276, n02206856}, which is {Snake (boa constrictor), Dog (Labrador retriever), Cat (tabby), Frog (tree frog), Turtle (leatherback

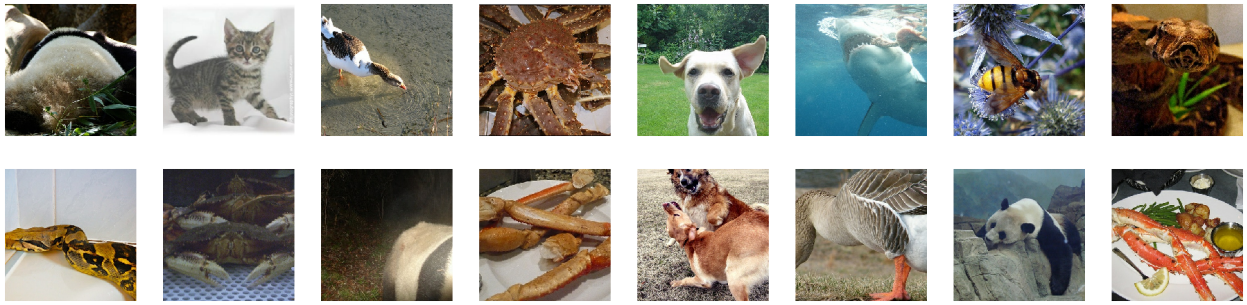

Figure 7: A representative sample of the images selected to form the test set of ImageNet10.

turtle), Bird (goose), Bear (giant panda), Fish (great white shark), Crab (king crab), Insect (bee)}, consistent with [22]. The categories were chosen to be visually distinct. Data is augmented using L-R flips and at most 20 pixel random offset crop. A sample of images from the test dataset are shown in Figure 7.

We define the accuracy to mean the Top 1 accuracy, and we define the adversarial accuracy as

$$1 - \frac{M + A}{N}$$

where M is the number of misclassified unperturbed images, A is the number of adversarial images, and N is the total number of images in the test set.

Adversarial attacks are conducted using the Foolbox library. We only evaluate robustness to Projected Gradient Descent (PGD), which is considered to be a universal first-order adversary and therefore defence against PGD offers a considerable security guarantee [14]. PDG was first described in [11] and is defined by performing an iterative transformation of an input image $x_{0,adv} = x_{original}$:

$$x_{i,adv} = Clip_{x_{i-1,adv},\epsilon}\{x_{i-1,adv} + \alpha \nabla_x Sign(\mathcal{L}(\dots))\}$$

where the clipping operation constrains the $x_{i,adv}$ to be in the appropriate range (e.g. requiring that each pixel is in [0, 255]), α defines the step size, and ϵ enforces similarity between $x_{i,adv}$ and $x_{i-1,adv}$ with some p-norm (i.e. $|x_{i,adv} - x_{i,adv}|_p \leq \epsilon$). In this paper, we use the

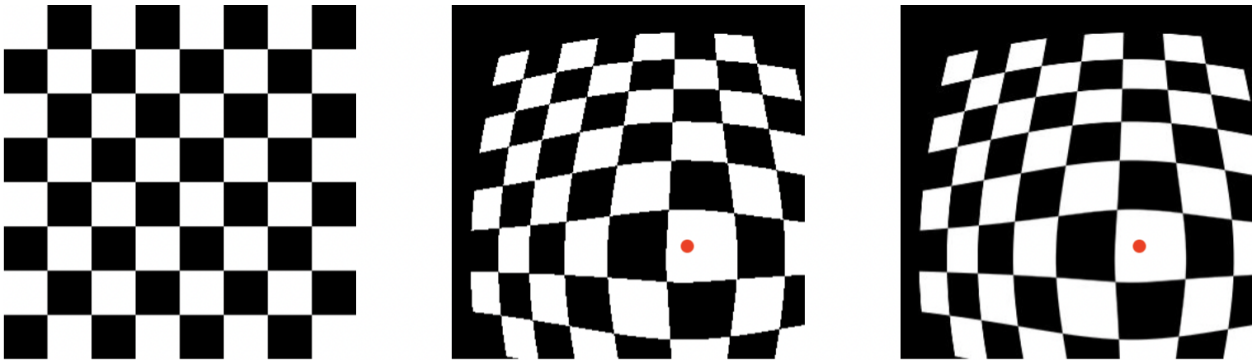

Figure 8: Left: A flat checkerboard. Center: The checkerboard with a retinal fixation at the red fixation point using the original discrete sampling. Right: The checkerboard with a retinal fixation at the red fixation point using bilinear sampling.

$p = 2$ and $p = \infty$ norms, but there is no consensus in the literature as to which choice of norm is most consistent with visual similarity.

All four models discussed in §2.4 are evaluated on ImageNet10, the results of which are shown in Table 1.

Model	Standard Accuracy	$\epsilon = 0.05$	$\epsilon = 0.02$	$\epsilon = 0.01$	$\epsilon = 0.005$	$\epsilon = 0.001$
Standard ResNet	89.2%	0.00%	0.02%	3.0%	24.4%	79.6%
Coarse Fixations	91.2%	0.00%	0.01%	1.20%	16.4%	80.8%
Retinal Fixations	90.0%	0.00%	0.2%	9.0%	41.0%	82.8%
Cortical Fixations	87.8%	0.00%	1.00%	11.8%	42.8%	82.0%

Table 1: Robustness to 20 step PGD.

First note that the coarse fixations model has the strongest standard performance, which is consistent to what was found in [7]. We also note that all three fixations models offer robustness improvements over the standard model. This motivates our work, in which we will test adaptive fixation mechanisms to further bolster robustness.

3.2 Bilinear Sampling

We begin by incrementally improving the retinal fixations technique described in [22]. We specifically change the underlying sampling method for retinal sampling from discrete to **bilinear sampling**.

After transforming the image to polar coordinates as described in §2.4 and resampling according to the retinal mapping, non-integer indices given by the mapping were previously truncated to integers. We improve upon this method by performing bilinear sampling instead of truncating. In bilinear interpolation, the four nearest pixel values at diagonals to the transformed index are averaged to determine the value at the index. This allows transformed images to appear smoother than with discrete sampling. The impact of changing the sampling method on an input checkerboard pattern can be seen visually in Figure 8.

Suppose that after we transform our coordinates, we must determine the pixel value P at a location (x, y) in the input image, where x and y need not be integers, which will go at integer pixel values (a, b) in the transformed image. To determine the value of pixel at (x, y) in the original image, we get the pixel values $P_{11}, P_{12}, P_{21}, P_{22}$ at $(\lfloor x \rfloor, \lfloor y \rfloor)$, $(\lfloor x \rfloor, \lfloor y \rfloor + 1)$, $(\lfloor x \rfloor + 1, \lfloor y \rfloor)$ and $(\lfloor x \rfloor + 1, \lfloor y \rfloor + 1)$ respectively.

Using these points, we can interpolate in the x direction to get P'_1 and P'_2 as

$$\begin{aligned} P'_1 &= (\lfloor x \rfloor + 1 - x)P_{11} + (x - \lfloor x \rfloor)P_{21} \\ P'_2 &= (\lfloor x \rfloor + 1 - x)P_{12} + (x - \lfloor x \rfloor)P_{22} \end{aligned}$$

Finally, we can average these to find P as

$$P = (\lfloor y \rfloor + 1 - y)P'_1 + (y - \lfloor y \rfloor)P'_2$$

This is the process of bilinear interpolation.

We find that making this small change to the code results in notable performance improvements. Adding bilinear sampling to the retinal fixations model marginally decreased standard performance from 90.0% to 89.8% (note that this was implemented in the model with batching—see §3.4 for more details—so the reference point for standard accuracy is 90.0% and not 90.2%); however, it offered substantial adversarial robustness benefits, as

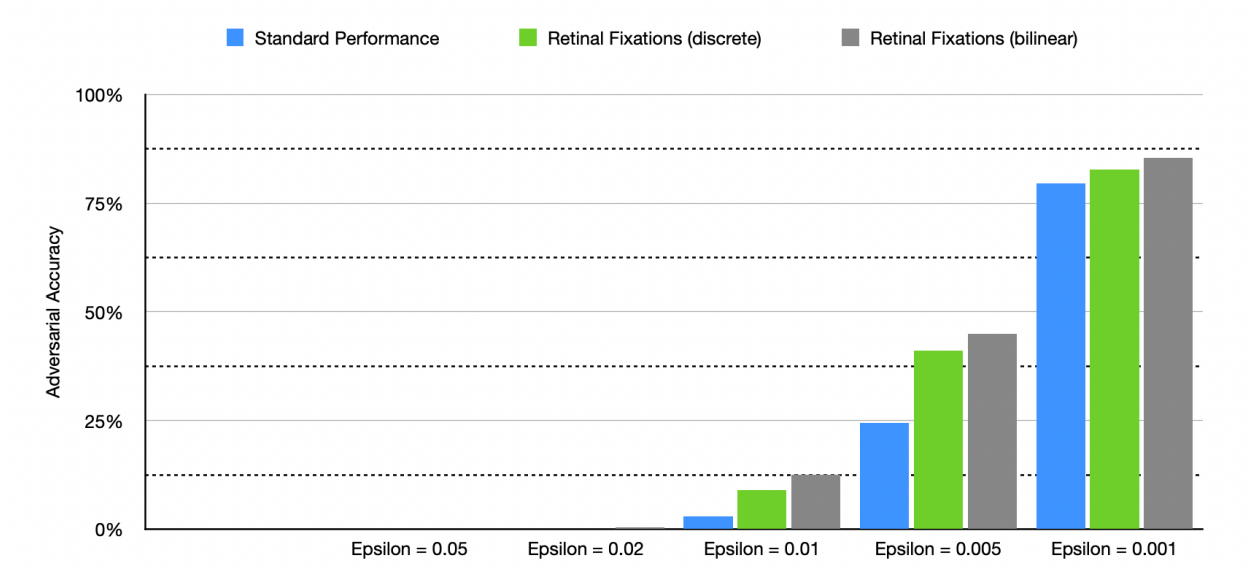

Figure 9: Demonstration of bilinear sampling robustness to 20 step PGD as shown by a comparison of a standard CNN, a CNN with retinal discrete fixations, and a CNN with retinal bilinear fixations.

shown in Table 2.

Model	Standard Accuracy	$\epsilon = 0.05$	$\epsilon = 0.02$	$\epsilon = 0.01$	$\epsilon = 0.005$	$\epsilon = 0.001$
Standard ResNet	88.2%	0.00%	0.2%	3.0%	24.4%	79.6%
Discrete Sampling	90.0%	0.00%	0.2%	9.0%	41.0%	82.8%
Bilinear Sampling	89.8%	0.00%	0.4%	12.6%	45.0%	85.4%

Table 2: Demonstration of bilinear sampling robustness to 20 step PGD as shown by a comparison of a standard CNN, a CNN with retinal discrete fixations, and a CNN with retinal bilinear fixations.

These results are shown graphically in Figure 9. This simple sampling change had a profound impact on the robustness of the model, perhaps due its smoothing nature—it is an averaging kernel after all—which minimizes the effect of small perturbations.

3.3 Revised Soft Attention Models

Concerning the soft attention models, we move on to more substantial improvements. In [22], attention points are fixed. In more detail, the training/testing process is as follows. During the training process, the attention models—coarse, cortical and retinal fixations—

are trained on *random* fixations within a central [(-80, -80), (80, 80)] bounding box within the 320×320 image. That is, the model is trained on a *single* branch. At test time, however, five duplicated branches are utilized as shown in Figure 6 at the five *fixed* fixation points shown in Figure 5 (right) and their output logits are averaged before going into the final softmax classification layer.

In this section, we explore the use of two more biologically-inspired approaches to attention. Rather than training on random fixation points and evaluating on fixed points at test time, we instead use two mechanisms for *learning* the fixation points. In both cases, we allow the model to learn a total of five fixation points.

First, we consider the use of a **spatial transformer network** (STN) following the initial augmentation layer of the network [9]. The spatial transformer network over time learns to apply linear transformations to input images that enhance classification accuracy, without the use of an auxiliary loss. In [9] and in many papers since then, STNs have been shown to learn to "standardize" input images. For example, in the case of MNIST, an intermediate STN may apply shear and rescaling transformations to the number 2 such that each input 2 is at approximately the same size and at the same orientation. In a dataset of birds, the STN can be used to locate and crop the bird in the image [9].

Second, we consider a much simpler and even more biologically-inspired approach to learning fixation points, creating what we call a **multi-gaze** model. Using the retinal transformations from 3.1, which take only (x, y) coordinates as input, we create a simple internal network that learns five of such (x, y) coordinates and subsequently applies the retinal transformations centered at those locations. This is particularly similar to the retinal fixations model in [22], but offers enhanced flexibility over the locations of the fixation points.

Both methods are well-motivated: human fixations are by no means random, but rather focus on the most important regions of the visual input. These models achieve a closer approximation to this approach than those in [22].

3.3.1 Parallel Transformer Model

The fixations of the eye are not limited to five points, but rather dynamically adapt to differing stimuli. To address this, we implement a spatial transformer network, as was first described in [9]. Spatial transformer networks consist of three components: a localization net, a grid generator, and a sampler. These components can be added to any network and are completely differential, allowing their parameters to be trained along with the rest of the network.

The **localization network** takes as input the original image and outputs a number of parameters of a linear transformation $\mathcal{T}_\theta$. Depending on the desired transformations, $\mathcal{T}_\theta$ can have a varying number of parameters.

Recall that we can scale a vector x by a factor of a by multiplying it with a matrix of the form $\begin{pmatrix} a & 0 \\ 0 & a \end{pmatrix}$. We can rotate a vector by an angle α by applying a rotation matrix $\begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$. Finally we can shear a vector by m in the y-direction and n in the x-direction by applying $\begin{pmatrix} 1 & m \\ n & 1 \end{pmatrix}$. By adding an extra dimension to the input vector, say $(x, y, 1)$ instead of (x, y) where the 1 was chosen arbitrarily, we can perform even more interesting transformations. For example, we can achieve a translation of a in x and b in y by applying $\begin{pmatrix} 0 & 0 & a \\ 0 & 0 & b \\ 0 & 0 & 1 \end{pmatrix}$. Note that transformations of the form $\begin{pmatrix} a & 0 & b \\ 0 & a & c \\ 0 & 0 & 1 \end{pmatrix}$ are referred to as "attention" mechanisms, while transformations of the form $\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$ are affine transformations.

Finally, note that we can combine these operations through matrix multiplication! Our

STN can thus generate rotation, translation, rescaling, and translation operators.

After the linear transformation matrix has been created by the localization network, the **grid generator** must generate a new sampling grid. That is, the original image was defined in some $x^s - y^s$ coordinate system, but after applying rotations, shears, rescalings, and translations, we will have a new coordinate system $x^t - y^t$. We need to map each point in the original system to the new one. This can be accomplished by simple matrix multiplication. That is, if we want to know the values we should put at (x_i^t, y_i^t) in the target image, we can use the source coordinates (x_i^s, y_i^s) as

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = \begin{pmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{pmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

where the G represents the regular grid being sampled. The grid generator therefore just does tensor multiplication.

Finally, the **sampler** takes the grid generated by the grid generator based on the parameters θ from the localization network as well as the original input image and transforms it accordingly. There are several ways to do this, but we have already covered the most common one: bilinear sampling! Note that bilinear sampling is a linear operation and is therefore differentiable. Letting P_{ij}^c be the pixel at $[i, j, c]$ in the input, the bilinear sampler from the and V_k^c be the value of the pixel at (x_k^t, y_k^t) in the channel c,

$$V_k^c = \sum_{i=1}^{H} \sum_{j=1}^{W} P_{ij}^c \max(0, 1 - |x_k^s - j|) \max(0, 1 - |y_k^s - i|)$$

The derivatives with respect to x_i, y_i, and P_{ij}^c are all easily computable in this form. For example,

$$\frac{\partial V_k^c}{\partial P_{ij}^c} = \max(0, 1 - |x_k^s - j|) \max(0, 1 - |y_k^s - i|)$$

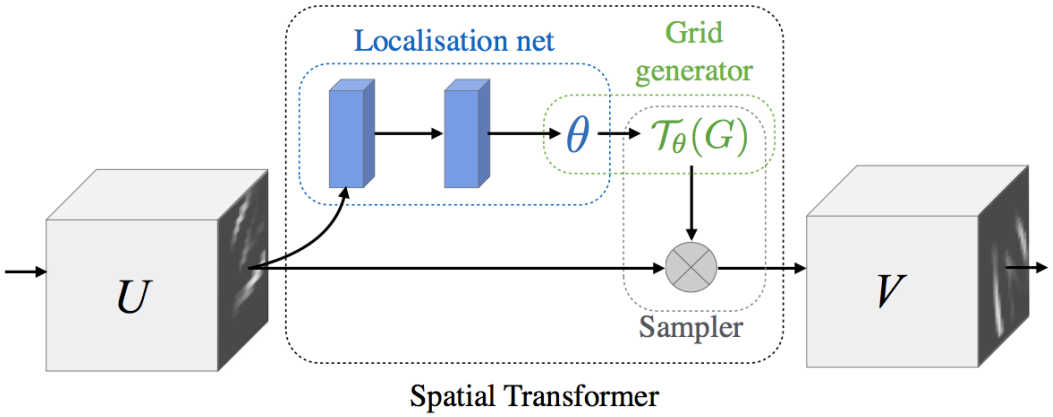

Figure 10: The Spatial Transformer Network architecture, reproduced from [9]. The localization net generates parameters θ. These θ transform the original coordinates to a new frame, which is accomplished by the grid generator. Finally the sampler applies bilinear sampling to the original image in accordance with the new grid. This can be inserted anywhere in an existing network, as indicated by the sub-networks U and V that sandwidh the STN.

and

$$\frac{\partial V_k^c}{\partial x_i^s} = P_{ij}^c \max(0, 1 - |y_k^s - i|) \begin{cases} 0, & \text{if } |j - x_k^s| \geq 1 \\ 1, & \text{if } j \geq x_k^s \\ -1, & j < x_k^s \end{cases}$$

Thus, the entire spatial transformation network, from the localization network to the grid generator and sampler are differentiable and can be trained alongside the rest of the network as normal classification training is underway. The entire network architecture is shown in Figure 10.

We modify the network from [22] as follows. We place a STN following the initial data augmentation layer of the original network. The STN can generate `num_transformers` $\times$ `num_theta_params` parameters, where `num_transformers` is the number of branches that will appear in the subsequent network and `num_theta_params` describes the kind of transformation—either affine (6) or attention (4). These parameters are then reshaped into the corresponding transformation matrices and applied to the input images, generat-

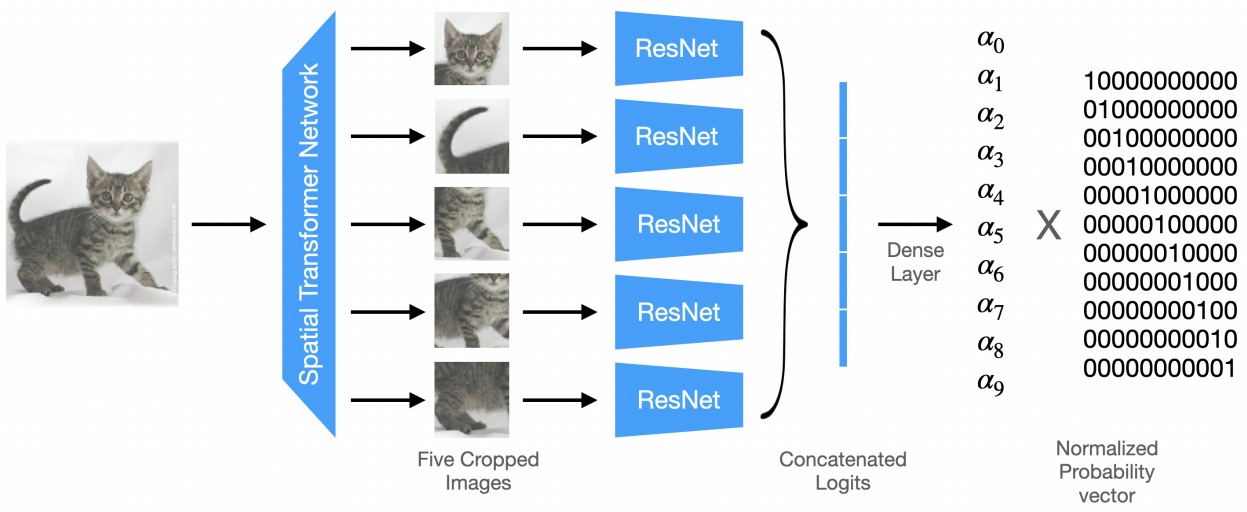

Figure 11: The parallel transformers model tested in this work. An input image is passed to a spatial transformer network, which itself is a ResNet. This ResNet outputs 4×5 or 6×5 parameters depending on the desired transformation. Recall that 4 parameters define an attention model and 6 generate full affine transformations. These transformations are then applied to the input, generating five new images which can be passed to ResNets whose logits are concatenated and passed to a final classification layer. In contrast to the approach in [22], the training and test architectures are the same—the model is trained with all five fixations. The weights can either be distinct or shared between the five ResNets. For more on this, see §3.4.

ing `num_transformers` new images from the original input. Each of these branches are then passed to a standard ResNet. The logits from each of the branches are then concatenated and passed to a single dense layer for classification. This network structure is shown in Figure 11.

The bounding boxes generated by the parallel transformer network on a sample of images in the test set are shown in Figure 12.

3.3.2 Multi-Gaze Model

The parallel spatial transformer networks of the previous section do more than the attention model we began with. They enable more general spatial transformations, including reflections and shears that are not truly biologically inspired. A less overparameterized approach is described below.

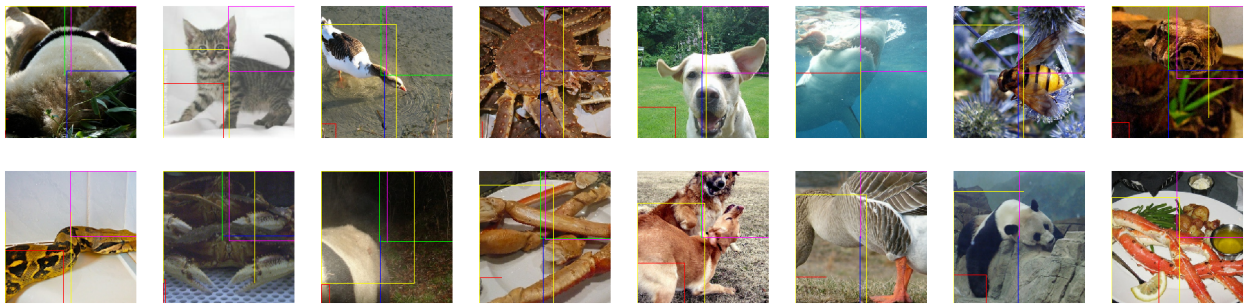

Figure 12: Examples of bounding boxes generated by a trained spatial transformer network on input images in the ImageNet10 test set.

The STN learns fixations in a roundabout way: if we want to focus on the head of a cat, we must first scale the image and then crop it using the bilinear sampler bounds. In the retinal fixations model we began with, we needed only to specify the coordinates of the gaze and the retinal transformation took care of the rest. To allow the gaze to be dynamic, we define a **multi-gaze** model, which uses a ResNet to predict `num_branches` $\times$ 2 coordinates which define the locations for the retinal sampling to be applied. Once again, those five transformed images generated from the output are passed to `num_branches` parallel ResNets which do the classification as in the parallel-transformers model.

The effect of gaze selection is shown in Figure 14 and the model architecture is shown in Figure 13. Some of the gazes generated by the multi-gaze network on a sample of images in the test set are shown in Figure 15.

3.3.3 Results

The original multi-gaze model deployed tended to select gazes in the extreme corners of images. An additional L2 regularized multi-gaze model was therefore trained (with regularization parameter 0.01) and tested.

The results are summarized in Figure 16. Notably, both the parallel transformer model and the multi-gaze model offer benefits over the standard ResNet model with varying adversarial perturbation sizes. For larger perturbations (0.01 and 0.005), the parallel transformer model offers the greatest adversarial accuracy while for smaller perturbations (0.001),

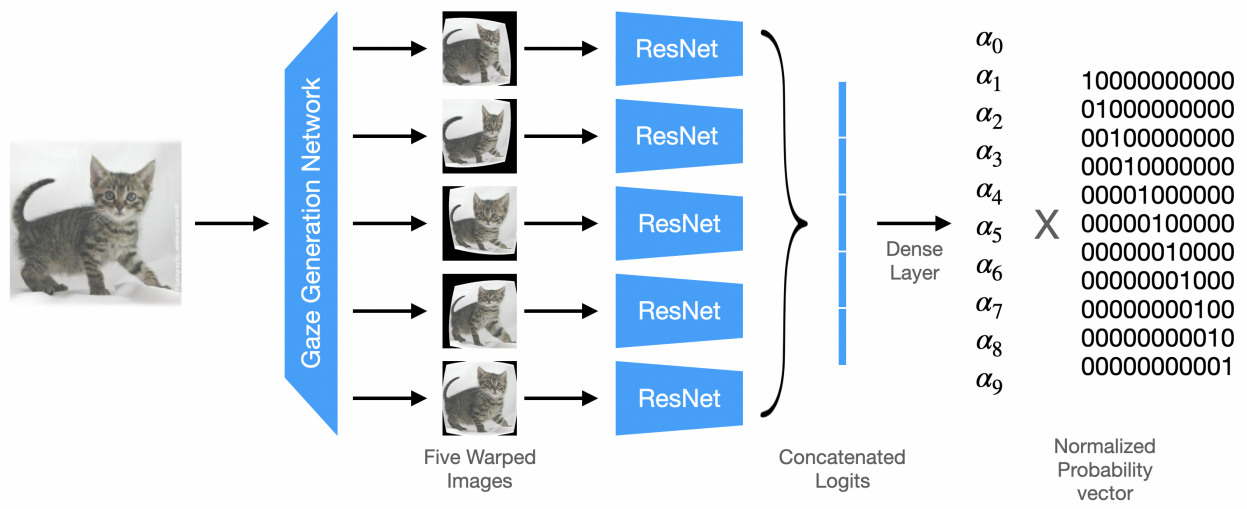

Figure 13: The multi-gaze model that was tested in this work. An input image is passed to the gaze generation network, which is itself a ResNet. The ResNet outputs 2×5 parameters which define only the (x, y) coordinates of the gaze (i.e. the fixation point). The actual warping is pre-specified in, whereas in the spatial transformer network, the model was free to learn its own deformations. The gazes are then applied to the image to generate five new images which are passed to subsequent ResNets. The logits from these five ResNets are concatenated and passed to a final dense layer for classification. In contrast to the approach in [22], the training and test architectures are the same—the model is trained with all five fixations. The weights can either be distinct or shared between the five ResNets. For more on this, see §3.4.

Figure 14: Left: The original image of a cat with a red dot at the location of the gaze to be applied on the right. Right: The warped image of the cat, with the fixation of the gaze applied at the location of the red dot.

Figure 15: Examples of gazes generated by the multi-gaze network on input images in the ImageNet10 test set. Note that unlike in Figure 12, each image in this figure only shows one gaze–for each example, four are omitted.

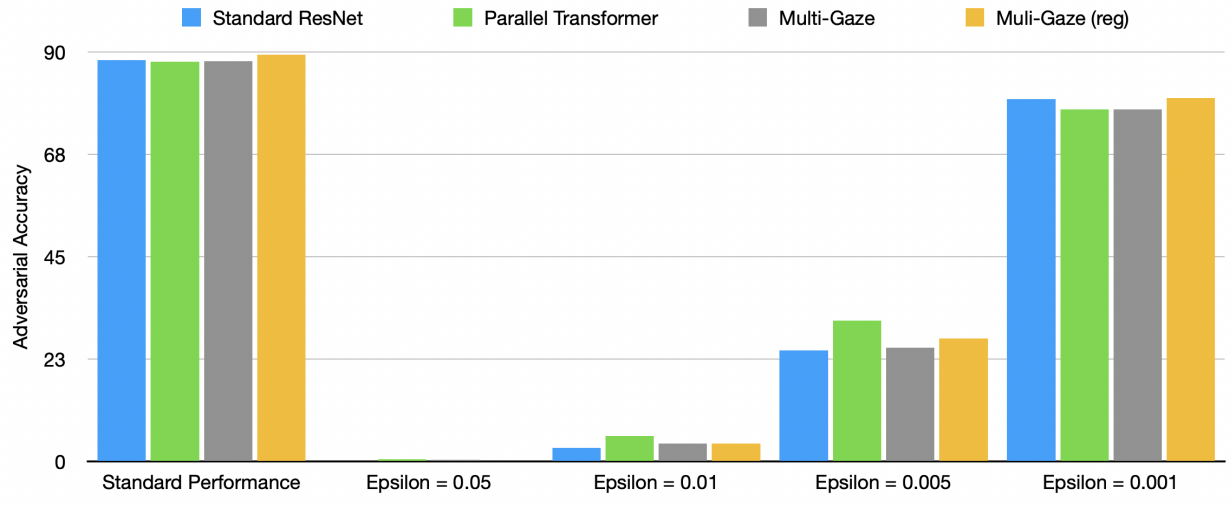

Figure 16: Demonstration of bilinear sampling robustness to 20 step PGD as shown by a comparison of a standard CNN, a CNN with retinal discrete fixations, and a CNN with retinal bilinear fixations.

the multi-gaze model offers the greatest benefit. Although difficult to see in the plot, the multi-gaze model does outperform the standard ResNet in this setting. Interestingly, the multi-gaze model with regularization outperformed the standard ResNet on unperturbed images, consistent with the trend that the multi-gaze model is particularly effective for smaller perturbation sizes.

3.4 Performance Enhancements

We implement two performance enhancements over the models in [22].

We first implement batching for the ImageNet10 dataset, resulting in substantially reduced memory for jobs. To do this, we convert the ImageNet10 dataset into `TFRecords` objects that can be efficiently fetched. We load images into memory, apply the relevant preprocessing, and create a generator object. These generators are cached by converting the images and labels to `BytesList` features which can finally be saved as `TFRecords`. Implementing batching had a dramatic impact on memory usage on the cluster—the memory required to executing the training dropped from approximately 60GB to 12GB. The training time also decreased modestly, as `TFRecords` objects can be more efficiently read than the raw image files.

The second performance enhancement was to implement weight sharing between branches of the network. Because we expect that classification should not depend on the position of the subject in the image, we can share weights between the branches of the parallel transformers and gaze mechanisms and achieve comparable classification accuracy for a fraction of the training and evaluation time as well as model size. This hypothesis was not exactly borne out, as both standard performance and adversarial accuracy decreased modestly as a result of using shared weights; however, the training time and model size did decrease significantly, presenting an option for enhancing adversarial robustness in memory or computationally constrained scenarios.

4 Discussion

In this work, we found several techniques that can enhance adversarial robustness without meaningfully sacrificing standard performance. Both the parallel transformer network and regularized multi-gaze model proved effective, as did changing the sampling mechanism from discrete to bilinear.

In future, it may be useful to investigate combinations of these mechanisms, as well as optimizing the number of branches and gazes in the transformer network and multi-gaze

model. Additionally, in this work, we concatenate logits from the branches/gazes, but other combination mechanisms can be investigated, like averaging. Finally, it is worth investigating each of these mechanisms under additional adversarial attack schemes.

References

[1] An on-device deep neural network for face detection. *Apple Machine Learning Research*, Nov 2017.

[2] About optimized battery charging on your iphone. *Apple Support*, Jan 2020.

[3] Wieland Brendel *, Jonas Rauber *, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*, 2018.

[4] Pouya Bashivan, Kohitij Kar, and James J. DiCarlo. Neural population control via deep image synthesis. *Science*, 364(6439), 2019.

[5] Alexandra Chouldechova, Diana Benavides-Prado, Oleksandr Fialko, and Rhema Vaithianathan. A case study of algorithm-assisted decision making in child maltreatment hotline screening decisions. In Sorelle A. Friedler and Christo Wilson, editors, *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*, volume 81 of *Proceedings of Machine Learning Research*, pages 134–148, New York, NY, USA, 23–24 Feb 2018. PMLR.

[6] Hironobu Fujiyoshi, Tsubasa Hirakawa, and Takayoshi Yamashita. Deep learning-based image recognition for autonomous driving. *IATSS Research*, 43(4):244–252, 2019.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[8] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[9] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu. Spatial transformer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Gar-

nett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[11] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale, 2017.

[12] Michelle Lee and Luciano Floridi. Algorithmic fairness in mortgage lending: From absolute conditions to relational trade-offs. *SSRN Electronic Journal*, 01 2020.

[13] Yan Luo, Xavier Boix, Gemma Roig, Tomaso Poggio, and Qi Zhao. Foveation-based mechanisms alleviate adversarial examples, 2016.

[14] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[15] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[16] Nina Narodytska and Shiva Kasiviswanathan. Simple black-box adversarial attacks on deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1310–1318, 2017.

[17] Aran Nayebi and Surya Ganguli. Biologically inspired protection of deep networks from adversarial attacks, 2017.

[18] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P. Wellman. Sok: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 399–414, 2018.

[19] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597, 2016.

[20] Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. In *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning*, 2017.

[21] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[22] Manish Reddy Vuyyuru, Andrzej Banburski, Nishka Pant, and Tomaso Poggio. Biologically inspired mechanisms for adversarial robustness. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 2135–2146. Curran Associates, Inc., 2020.

[23] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. In *International Conference on Learning Representations*, 2018.

[24] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[25] Jiliang Zhang and Chen Li. Adversarial examples: Opportunities and challenges. *IEEE Transactions on Neural Networks and Learning Systems*, 31(7):2578–2593, 2020.

[26] Daniel Zoran, Mike Chrzanowski, Po-Sen Huang, Sven Gowal, Alex Mott, and Pushmeet Kohli. Towards robust image classification using sequential attention models. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9480–9489, 2020.