Visualization and Selective Dropout for Adversarial Image Classification

William Hinthorn
hinthorn@princeton.edu
Adviser: Prof. David Dobkin

Abstract

This paper first seeks to reproduce results published by Christian Szegedy et al in [12] regarding the generation of images which appear to the human observer as identical to parent images correctly classified by a neural network but which are are labelled by the network (with a corresponding high certainty) as an arbitrary incorrect label. I then build on [13] in constructing a framework useful for the generation and analysis of adversarial images by examining the signal propagation through deep hidden layers. This is enabled by adapting visualization techniques from [14] to analyze differences in activations of subsets of nodes (both fully-connected neurons and convolutional feature maps) when stimulated by images and their adversarial counterparts. Based on observations of concentrated assignment of blame over a layer, a novel method of hard negative training is proposed called selective dropout, which balances the need to identify and accurately classify unseen adversarial images of known perturbation groups with the desire to maintain the high classification accuracy of the original network on the much larger naturally occurring data set. Neural networks trained using this method in general converge to a local optima in 1/2-1/3 fewer steps while learning to generalize on previously unseen adversarial images with accuracy on par or better than the control. This is done while exhibiting a more robust memory for the original, expected image space.

1. Introduction

The machine learning renaissance of the current decade has been fueled by neural networks' superhuman performance in tasks such as image classification (e.g.[2],[6]) and game play (e.g.

[9]), and as a result, these systems are being incorporated into critical security apparatus, medical devices, and are the keystone in autonomous transport systems currently under development. While the performance of neural networks on naturally occurring data is impressive, the way that they learn information is opaque and their key weaknesses poorly understood. Industry is making the key assumption that large amounts of training data (thus the term big data) is the necessary and sufficient condition to enable variations of current network designs to perform with satisfactory reliability for mass production.

Figure 1: The adversarial image (right) was generated using an L-BFGS optimizer to find a specific set of perturbations (center) which, when added to the original, correctly classified image (left), cause the network to believe with high confidence that the image is some arbitrarily selected second class (in this case a cardigan). The picture was generated using code adapted from [13].

However in 2014, Christian Szegedy et al identified a key problem, namely that while networks may achieve incredible performance on most images, the narrow prediction manifold that is learned can be exploited by an adversary who, by applying nearly imperceptible perturbations to a correctly classified original image, may fool a network to classify the image with an arbitrarily selected second label [11]. Figure 1 shows an example which was generated for this paper by strategically selecting the perturbed layer in the center such that when added to the original image, the neural network is fooled into believing that the resulting image is a cardigan. Note that the magnitude of perturbation added to each individual pixel is so small as to prevent all but the most astute observer from noticing any alterations. This additive distortion layer is re-rendered in figure 2 to depict the form of the noise.

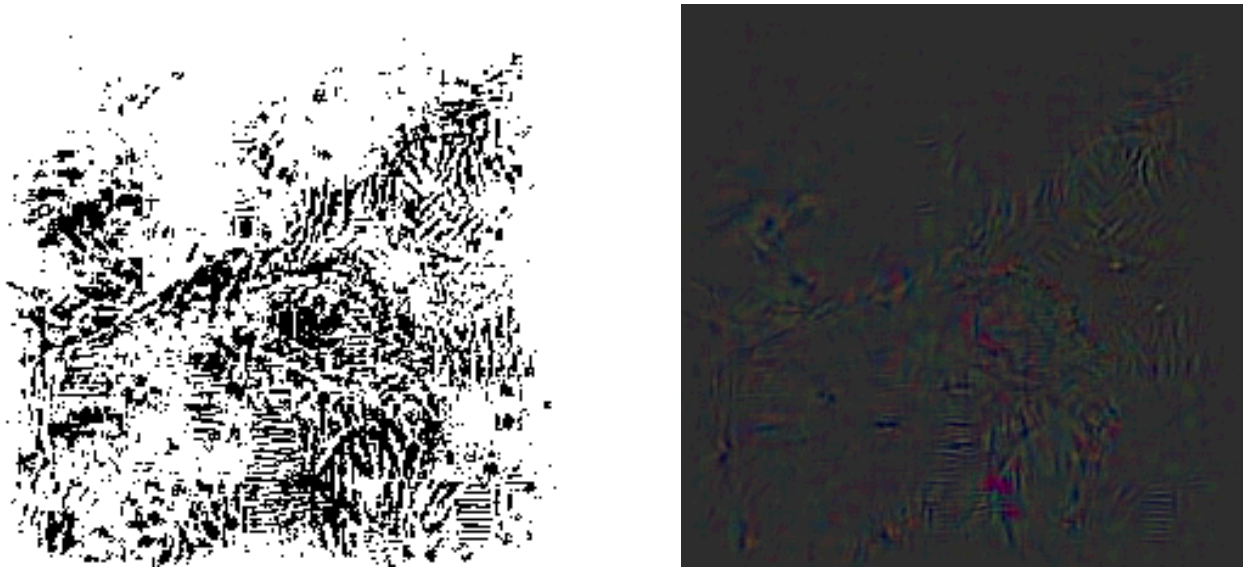

Figure 2: The perturbation or distortion layer is re-rendered here to make the exact form of the additive noise apparent by mapping the RGB pixels to a single color gradient and thresholding (left) and by adding 80% contrast to the image (right). Note that the noise bears no resemblance to the adversarial label (cardigan).

What is more, they noted that images generated to fool a network of a specific architecture which was trained on a certain portion of the data set will fool (cause the network to incorrectly classify) another network of a different architecture trained on a different portion of the data set with high probability ($p > 70\%$). This "cross-model generalization" property of the images implies that an adversary may create images designed to fool a test network that with high probability will fool any state-of-the-art network of unknown characteristics. Naturally, this is an undesirable characteristic for systems meant to diagnose patients, identify criminals, or safely transport humans through public spaces that additionally prevents the use of simple ensemble methods to catch potential adversarial inputs.

Furthermore, the problem is difficult to understand or resolve due to lack of a general method to completely interpret the information deep neural networks have learned. Because of this, current research primarily relies on empirical methods to safeguard existing architectures against adversarial images. This semester's work sought to reproduce tests and analyses of this problem and lay a groundwork for further research in computer vision. This paper first will introduce the reader to key structures and terminology critical to understanding the neural network implementation. Then

relevant research will be discussed in section 3 to further elucidate that which this paper seeks to reproduce and understand. Finally, the three primary components of this semester's work will be detailed along with their results.

2. Technical Background

2.1. Neural Networks

An artificial neural network (ANN) is a computational structure that automatically learns to encode nonlinear probability spaces and that computes using a relatively small number of largely parallel computations. ANNs are composed of thousands of artificial neurons or perceptrons. These nodes are analogous to logic gates in a digital computer. They are defined by a set of n weights w_i which act on the input a_i, $(w_i, a_i : i \in \{1, 2, ..., n\})$, a bias b, and a nonlinear activation function f (i.e. the heavyside, logistic, or rectified linear function). The perceptron's output is found by composing the activation function with the sum of the product of each corresponding value a_i, w_i added to the bias (alternatively called the *threshold* value). This takes the form $f(\sum_{i=1}^{n} a_i w_i + b)$. One may note that the sigma notation may be avoided by marking (a, w) as vectors $(\hat{a}, \hat{w})$. Furthermore, we may generalize the bias as an additional dimension of the weight vector of index 0 by appending b to the head of $\hat{w}$, taking care to append a corresponding dimension to $\hat{a}$ valued at 1. The perceptron then computes its output by first taking the inner product $\hat{x}^T \hat{w}$ and then passing it through the activation function. This can be written as $f(\hat{w}^T \hat{x})$. Vector notation is preferred due to its compact nature and simplicity. A linear perceptron normally takes f to be the heavyside step function $f(x) = \begin{cases} 1 : x \geq 0 \\ 0 : x < 0 \end{cases}$. If the function is a logistic sigmoid $\sigma = \frac{1}{1-e^{-x}}$, then the perceptron is referred to as a sigmoid neuron. A diagram of just such a neuron is shown in figure 3. It can be proven that for proper values of $\hat{w}$, a linear perceptron may draw a decision boundary dividing any set of linearly-separable data.

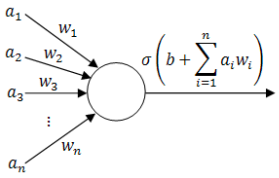

Figure 3: Artificial neurons are the basis for modern neural networks. Source: blog.manfredas.com

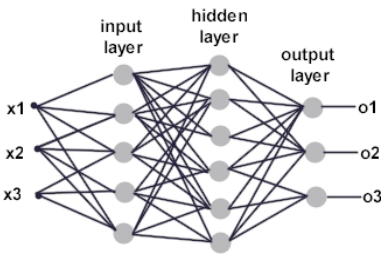

Figure 4: An example MLP of depth 3. The hidden layer is of width 6 (i.e. has 6 neurons in parallel). Source: http://dms.irb.hr

Though in general perceptrons can be connected in any way, most ANN architectures (and indeed the net used in this paper) are modelled after the multi-layer perceptron (MLP). These consist of many layers of neurons in which each neuron in layer i is connected to each neuron in layer $i+1$ (i.e. fully-connected layers). Additionally, most non-recurrent architectures do not connect any neurons within a given layer. The network's weights are normally initialized to bounded random values. The input to the network is taken to be a vector of values which is passed into layer 1. The MLP computes the inner product of the input and the weight vectors corresponding with each neuron in the first layer, passes the values through an activation function, and then outputs these values to each neuron in layer 2. In this way, the data is *fed forward* through each layer until it is eventually passed to the output layer (taken to have k neurons, each representing a particular class). In most networks, the output neuron of the highest activation is taken to be the prediction of the network. It can be proven that these types of *feed-forward* nets with one intermediary, "hidden" layer (i.e. a network of depth 3) can, under mild assumptions about the activation function, approximate any function.

This classical MLP is trained using a form of supervised learning, where a correct label is supplied along with the inputted training example. The prediction of the network is compared to this correct label, and if the prediction is wrong, a value is computed to quantify the degree of error (a simple example being -1 if incorrect, 1 if correct). This quantification is called the cost function (or conversely, the reward function). To train the MLP to make better predictions, the weights of the entire network are updated with respect to the error of the prediction. This is done by taking the derivative of the cost function with respect to each weight in the network. The weights are then updated by taking the value at time t and then stepping in the direction of this computed gradient. In other words, $w_i^t = w_i^{t-1} - \eta \frac{\partial C^t}{\partial w_i}$ where η is a hyperparameter which determines the *learning rate* and is usually set as some value close to 0.001. C is the cost of the network at prediction t.

Thus the probability space of the input is approximated by iteratively changing the weights

throughout the network through a process called backpropagation. Just as computations are done in parallel for each layer and sequentially from the input to the output layer when making a prediction, backpropagation allows us to compute the δ values to update weights (and biases) in each layer starting from the output layer and then propagates the delta values from layer $i+1$ to layer i to compute δ associated with the previous layer. Thus predictions and updates may be calculated in time linear with respect to the number of layers and take advantage of the parallel computing capacities of modern GPUs to speed up calculations within each layer. In this way, trained weights approximate an unknown probability space by finding the gradient of the space at each time t and incrementally descending until the network reaches a local optimum. This learning process is known as *gradient descent*.

Additional optimization techniques are employed while training most modern networks. Building off the image of network learning as exploring the optima of a probability space, *momentum* is employed to smooth out the path taken on the way to a stationary point (usually taken to be a local minimum). Exponential decay is frequently applied to both the weights and the learning rate for different reasons. The learning rate decay λ assists the speed and likelihood of convergence by allowing an initially large learning rate (to speed up training) to approach zero at a rate which parallels the rate of the network's approach to a minimum. Weight decay is a form of regularization (in this paper taken to be ℓ_2 regularization) which forces the network to store predictions across terms, curbing the network's tendency to over-fit the training set. ANNs are normally trained over batches of training data to further smooth the path of descent, and training is divided into many *epochs* marking the completion of training over the entire training set.

2.2. Convolutional Neural Networks

The MLP described above already achieves a surprisingly high classification accuracy when trained on a large enough data set. However one might find it odd that the 2D image is computed as a single vector with each pixel evaluated without regard to its relation with other nearby pixels. To address this problem (and additionally reduce the number of learnable parameters within the network), most

vision networks replace the fully-connected layers with *convolutional layers*.

While the full description of convolutional networks is beyond the scope of this paper, a general understanding of their unique properties is vital to understanding the rest of the paper. Convolutional neural networks (CNNs) were developed to work much like the retina of a living organism. They seek to maintain the local relationships within the input space thereby introducing a certain degree of translational invariance.

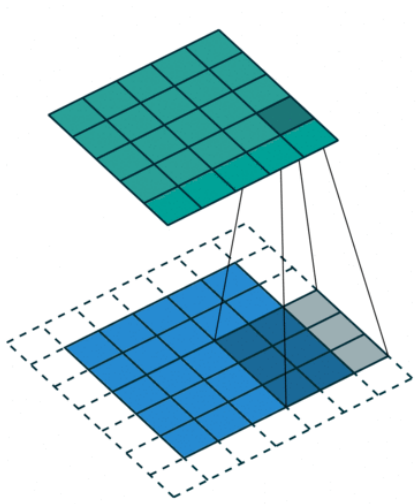

Figure 5: Convolutions can be understood as sweeping matrix multiplications. Source: https://github.com/vdumoulin/conv_arithmetic

The convolution operation is conducted using a small (usually taken to be 3x3, 5x5, or 7x7) convolution matrix which is swept over the 2-dimensional input space (sometimes taken to be three-dimensional if the input is a depth-3 RGB image with corresponding R, G, and B pixel layers). At each step, the convolution matrix is multiplied by the sub-matrix of the input corresponding to the convolution matrix's current location. The matrix product is then mapped to a single value in the output matrix (which is sometimes called a *feature map*). The values of the feature map are filled in by stepping over the entire input (sometimes padded on the sides with zeros in order to maintain or extend the size of the space). These feature maps are then passed through an element-wise activation function to enable classification of non-linearly separable data. Feature maps are analogous to the perceptrons or artificial neurons of the previous section. Different convolution matrices are used to learn and extract different pertinent features and are trained using a variation of backpropagation (where multiplications are replaced by convolutions).

Since the same convolution matrices are used over the entire input space (rather than being associated with a particular pixel of the input), the weights can be thought of as being "shared." This reduces the number of parameters to be learned vis-á-vis a fully-connected network (thus speeding

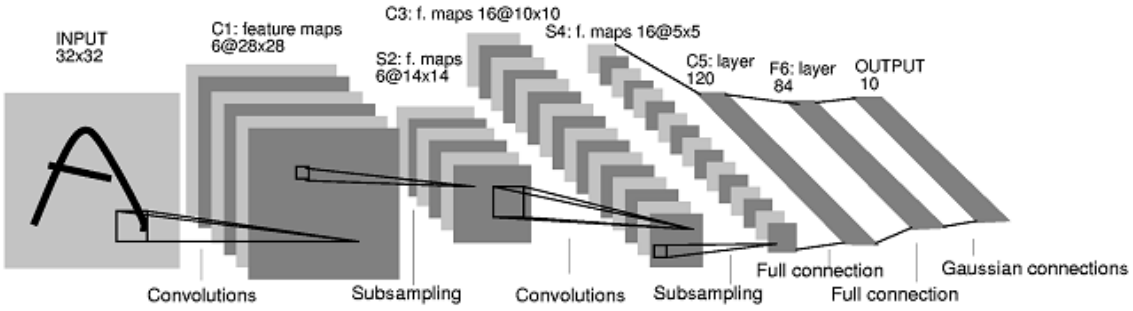

Figure 6: An example CNN. A series of convolutional layers are normally followed by one or two fully-connected layers to balance feature extraction and classification tasks. Source: https://i.stack.imgur.com/oUwMk.png

up training time) and makes the network more robust to certain types of transformations of the input space.

The CNN used in this paper was chosen to be relatively shallow to increase interpretability and speed of training. It has two convolutional layers, each of which uses a rectified linear (relu) activation function ($f(x) = x > 0?x : 0$) and is succeeded by a maxpool layer (where the dimensions of feature maps are reduced by selecting the maximum value in each *nxn* sub-matrix). n was chosen to be 3,2 for operations following the first and second convolutional layers respectively. The outputs of the second convolutional layer were passed through a single fully-connected layer (to mimic the common practice in state-of-the-art networks) also with a relu activation function before being passed through the output layer. The widths of the input and output layers are determined by the task, the former being the dimensions of the input vector and the latter being the dimensions of the prediction space (i.e. total number of labels for a classification problem). The hidden layers were chosen to be of widths 32, 64, 200 for the two convolutional and fully-connected layers respectively.

3. Related Work

The adversarial example problem is first raised in [11]. This paper formally states the assumption that, given an input image x which is correctly classified by the network (i.e. $f(x) = l_t$ for true label l_t), one would expect $f(x+r) = l_t$ for disturbance r bounded by some small ε. However, they found that, using the box-constrained L-BFGS algorithm, one may generate disturbance r which, when added to the original image, causes the network to classify the image as some arbitrarily chosen

adversarial label l_a. The algorithm is used to perform a line search and approximate the 2-norm of the disturbance subject to the constraint that $x + r \in [0,1]^n$, where n is the dimensionality of the input.

This paper sought to reproduce the results of [11] and so used the L-BFGS algorithm as a means of generating adversarial images to be visualized and used for training. The algorithm will be explained in more detail in section 5.1.

3.1. Adversarial Images and Garbage Images

In their 2014 paper on fooling convolutional networks, Nguyen, Yosinki, and Clune identify a phenomenon related to adversarial images [8]. Specifically, evolutionary algorithms may be used to generate "TV static" which is classified with high confidence as an arbitrary label. This class of mistakes highlights differences between human and artificial perception (at least in how the two disparate systems learn) and indicates the need for further development of the theoretical framework for neural networks before systems are blindly implemented.

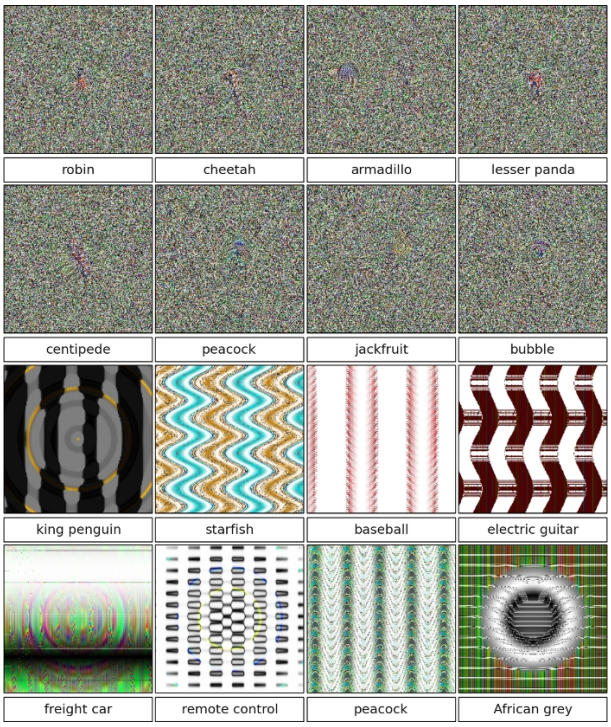

Figure 7: Example garbage images from [8]. State-of-the-art deep neural networks trained on ImageNet believe with $\geq 99.6\%$ certainty to be a familiar object.

Building on this and previous work, Goodfellow, Shlens, and Szegedy sought to explain modern networks' susceptibility to these images as a consequence of their relative linearity [3]. Recognizing that the dynamic range of an image is bound by the storage structure (i.e. most images limit pixel values to a vale $v \in [0,255]$), the p-norm of perturbation η added to an image is also bound by some small ε ([3] uses $p = \infty$). The pre-activity values of a neural layer when analyzing an adversarial image can be simplified to

$w^T \bar{x} = w^T x + w^T \eta$ where $\bar{x}$ is the adversarial input, x is the original input, and η is the perturbation. Then even though $||\eta||_\infty$ is bounded by ε (so that the magnitude of each element in the perturbation layer is small enough so as to be imperceptible), the pre-activity can be manipulated by a magnitude linearly proportional to the dimensionality of the input. By adding small perturbations parallel to the gradient of the cost function with respect to the input, one may maximize the prediction error using perturbations with norms of bounded magnitude.

3.2. Establishing Bounds

[12] further outlines the instability of the space of correctly classified images through spectral analysis. They show that the instability may be explained using the Lipschitz constant L_k for each layer $k = 1...K$:

$$\forall x, r, ||\sigma_k(x; W_k) - \sigma_k(x + r; W_k)|| \leq L_k ||r||$$

where σ_k is the composition of the activation function with the inner product of the weight matrix of layer k with the values x from layer $k-1$, and r is the bounded perturbation. The constant L for the entire network can then be bounded as $\Pi_{k=1}^{K} L_k$. The authors further qualify the likelihood of the existence of adversarial examples by computing the operator norm for each layer (using Parseval's formula to find the supremum of a function proportional to the 2-d Fourier transform of a function of the dimensionality, label, and step size for a given layer). In essence, this places upper bounds on the adversarial image space, where a small Lipschitz constant indicates that the creation of such images is difficult.

3.3. Adversarial Space

Tabacof and Valle seek to explore the adversarial image subspace of a pre-trained network by testing the classification when perturbed using differing total magnitudes of noise in [13]. The inclination of this paper differs by recognizing that the existence and bounds of the adversarial subspace have already been clearly defined in [12]; however, the framework which Tabacof and

Valle have developed serve as a useful starting point upon which to create tools which generate adversarial images, analyze deep hidden layers, and train networks using selective dropout.

3.4. Dropout

Much research has been done to prevent neural networks from over-fitting the training set. One widely adopted methods for doing so was proposed by Srivastava et al in [10]. Entitled "dropout" by its authors, the method independently selects a subset of neurons to be unused (or "dropped out") for each training example. In this way, the network trains a different sub-architecture on each example which inhibits excessive optimization to the training set. Dropout is characterized by the hyper-parameter p, defined as the probability that a given neuron will be dropped during a training step.

The method this paper chooses to call *selective* dropout is quite different than its namesake in that it is not a truly randomized algorithm. However, its intent is similar, namely to keep the network from over-fitting a training set (in this case taken to be the set of generated adversarial images, which as we have seen do not tend to appear naturally).

4. Data Set

The MNIST (Modified National Institute of Standards and Technology) data set was developed in order to assist the automation of the distribution of mail based on postal codes [7]. It consists of handwritten digits (each image corresponding to a labelled value $l \in [0-9]$) written by high-school students and employees of the US Census Bureau. The data set is divided into training and test sets of of 60,000 and 10,000 images respectively. The size of each image is 28x28 pixels. The images have been normalized and centered, a fact which improves the accuracy of many classifiers. The MNIST data set was chosen for these experiments because it is well studied and easy to train on, and the low-dimensionality of the input space aids the interpretation of adversarial images and their impact on the network.

5. Methods

Evaluation was done based on training error (which is not a good metric, and it is no wonder the control beat selective dropout given this method).

Building off of the code developed in [13] designed to test the critical points of the predictions by applying noise using Monte-Carlo methods, analysis was done using the Torch7 deep learning framework using a Lua wrapper.

A 3-layer convolutional neural network of architecture described in section 2.2 was first trained for 100 epochs over the MNIST training set (initializing learning rate η as $1x10^{-4}$ with decay rate of $\lambda = 5x10^{-7}$, a momentum term of $m = 0.9$, and a weight decay parameter $\alpha = 1x10^{-3}$). By the end of 100 epochs, the CNN has an accuracy rate of 99%. We refer to this pre-trained CNN as the base.

5.1. Adversarial Image Generation

The BFGS algorithm is a quasi-Newton method which approximates the inverse Hessian of a matrix to search for a local optimum in a non-convex space [5]. L-BFGS is a limited-memory variant. In a simple, low-dimensional space, we may perform perform a 2^{nd}-order descent in search of a stable point by computing the gradient and hessian of the cost function and use this

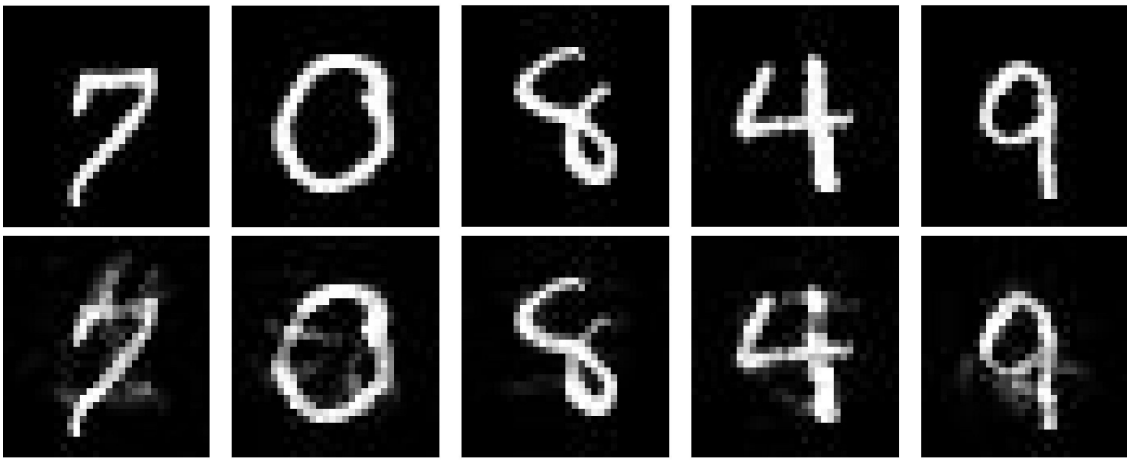

Figure 8: Adversarial images (bottom) generated using the L-BFGS algorithm on the original, correctly classified images (top). Note that due to the low resolution of the MNIST data set, the added disturbance is actually quite noticeable.

quadratic approximation to step in the correct direction (using step sizes δx which minimize the approximation). The minimization is then done by computing the product of the inverse Hessian and the gradient $H^{-1}g$. However the parameter space of modern neural networks can be upwards of 1 billion so computing the inverse Hessian normally is intractable. Because of this, the inverse Hessian is approximated by maintaining two properties: the secant condition and symmetry. Symmetry is necessary since the Hessian, defined as $\partial f/\partial x \partial y$ does not depend on the order of operation. The secant condition requires that $H(\delta x_n) = \nabla f(x_n) - \nabla f(x_{n-1})$. In other words, the gradient at time $n-1$ should be equal to the sum of the gradient at time n plus the Hessian of step between times $n-1$ and n. BFGS then computes an approximation of the inverse Hessian at time k by minimizing the Frobenius norm of the difference between the inverse of this and the inverse Hessian at time $k-1$. Formally, BFGS computes the following:

$$min_{H_k^{-1}} ||H_k^{-1} - H_{k-1}^{-1}||_2 \text{ bounded by}$$

$$H_k^{-1}(x_k - x_{k-1}) = \nabla f(x_k) - \nabla f(x_{k-1}$$

$$H_k^{-1} = (H_k^{-1})^T$$

Then since the recurrence relationship can be defined using vectors $(x_k - x_{k-1})$ and $(\nabla f(x_k) - \nabla f(x_{k-1})$, the inverse hessian need not be computed (except for an initial value). L-BFGS approximates this by truncating the update using the last n deltas.

This paper uses the L-BFGS algorithm as a means of producing adversarial examples because it was the method selected in [12]. In order to reproduce the results of [13], the Fortran implementation of L-BFGS written by Nocedal was used [1].

208 adversarial images were generated by randomly selecting correctly classified training examples and running the L-BFGS algorithm with a cost function defined by the classification error on the randomly selected adversarial label (i.e. penalize the system if it predicts any label that is not one we wish the network to incorrectly predict) bounded by a maximum normed value of the added disturbance. Figure 8 depicts five randomly selected images and their corresponding adversarial examples.

5.2. Visualization

Now that adversarial examples are generated, one may visually explore the manner by which the network is fooled. Though a set of theoretical rationales for adversarial images have been outlined [3], there is not a strong understanding of how to control the disturbances which propagate through the network. Since the dimensionality of the feature maps output by the the first convolutional layer is close to that of the inputs and the outputs are still a nonlinear output of a linear combination of pixels in the image (rather than a nonlinear output of a linear combination of nonlinear features derived from the image), the feature maps are relatively easy to interpret. However, after the first layer, outputs and feature maps become quite difficult to understand, since they represent high level learned encodings of abstract semantic information.

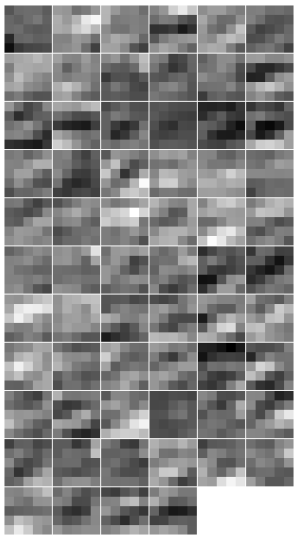

Figure 9: Direct analysis of feature maps in deep hidden layers is generally indecipherable.

Visualization techniques were adapted from those proposed by Zeiler and Fergus in [14] to compare activations of individual neurons over a given layer (both convolutional and fully connected) in the network when stimulated by original and adversarial image pairs. In order to decipher the encodings and selected features, the authors proposed that neurons might be visualized on an individual basis by isolating associated signals and reconstructing them in the sample space. This is done by connecting what they call a "deconvolutional net" to the layer under examination. The deconvolutional net maps the signal of the neuron in question back (inverting each activation function and flowing back through fully connected and convolutional layers) onto the original image space. This is done by first selecting an image to be input as stimulus. The feed-forward process proceeds as normal, with computed values propagated through the network. Then by masking over all neurons in a particular layer (by zeroing them out) except for the one in question, the observer may isolate the signal of an individual component and view the features it selects in the original input.

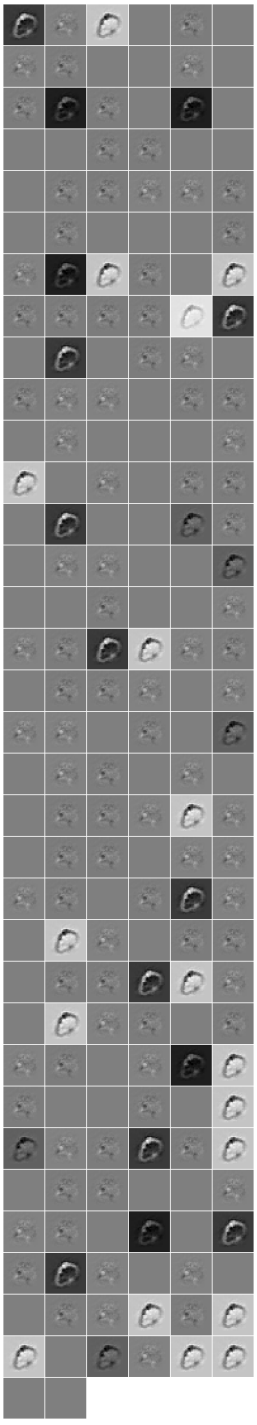

Figure 10: Concentration of blame becomes evident in deep, fully connected layers. Many activations are invariant to adversarial pairs, indicating that excessive training could be counter-productive.

Zeiler and Fergus employed this method to explain how invariances within the network translate to accurate classification of images transformed using elementary operations (e.g. rotations, translations, occlusions, etc). Conversely, one may use these methods to learn how these invariances are broken.

This method was adapted to examine this phenomenon. Projections were made without need for a second network by reversing the signal through the original CNN, taking care to properly account for gradients in shallow layers.

This form of analysis suggests a manner by which the signal might be appropriately managed. The correlation between neuron subsets and particular features is noted in [14], which prompted this paper's use of these visualization techniques to determine whether blame may be assigned to strict subsets of the network.

Observations made on these images support this conjecture, as is demonstrated by the stark contrast in differences in activations of neurons in the fully connected layer (figure 10). Defining volatility as the greatest absolute difference between the Frobenius norm of a neuron's activity when stimulated by the adversarial image and that of the activity when shown the original, it may be understood that the most volatile neurons with create a "back door" by which adversarial examples of similar associations might be easily generated. To better characterize these associations, one may define *perturbation groups* using both the original and the adversarial labels (e.g. images of 0's which were perturbed to be classified as 9's, etc.). In subsequent analyses, these group definitions were relaxed to allow generalizations based on only the range (e.g. activations of all

images which were perturbed to artificially classify as 9's) or domain (e.g. images of 0's which were perturbed to be classified as some other arbitrary label).

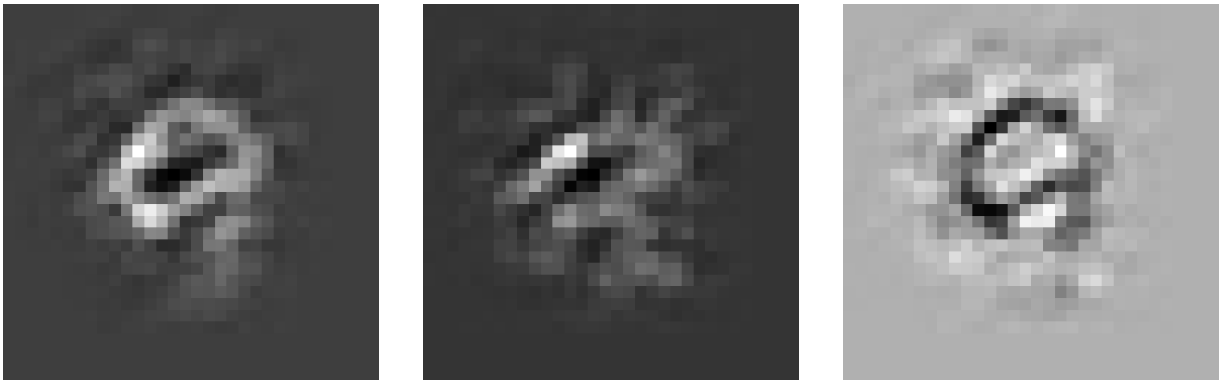

Figure 11: A comparison of averaged activations over the second convolutional layer when projected onto the original image space. The image on the right is calculated by taking the difference between the activations when analyzing the adversarial images (center) and their source images (left). Since over-linearity in the network may be associated with the existence of adversarial spaces [12], linear combinations over a specific layer may be used to judge influence at various depths.

Figure 12 highlights the differences between the activations of each neuron in the 2nd convolutional layer when stimulated by the original image and its adversarial counterpart. This uses the first definition of perturbation group, with different representations of 0 being transformed such that they are incorrectly labelled as 9's. Averaging over projections for a certain layer enables one to see a degree of semantic difference as encoded by that depth.

The original, adversarial, and differences of four members of the perturbation group (9, 1) are displayed in figure 11. One may notice that the difficulties of generalizing the characteristics of the adversarial noise by only directly

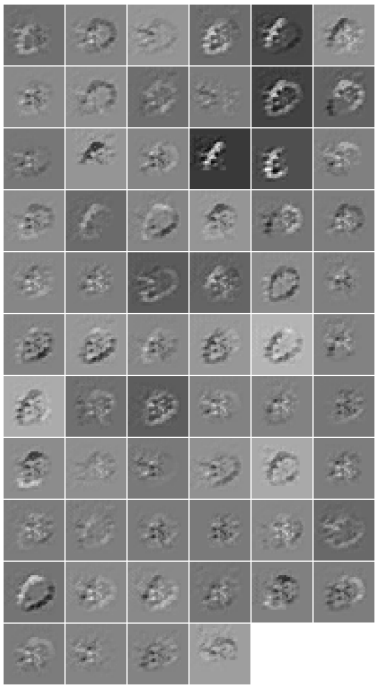

Figure 12: By projecting back onto the original input space, one may realize that certain neurons are impacted to a much greater degree than others.

examining the input are greatly reduced as we explore deeper in the network. This is a step towards understanding how classification errors are formed as the signal propagates through the network.

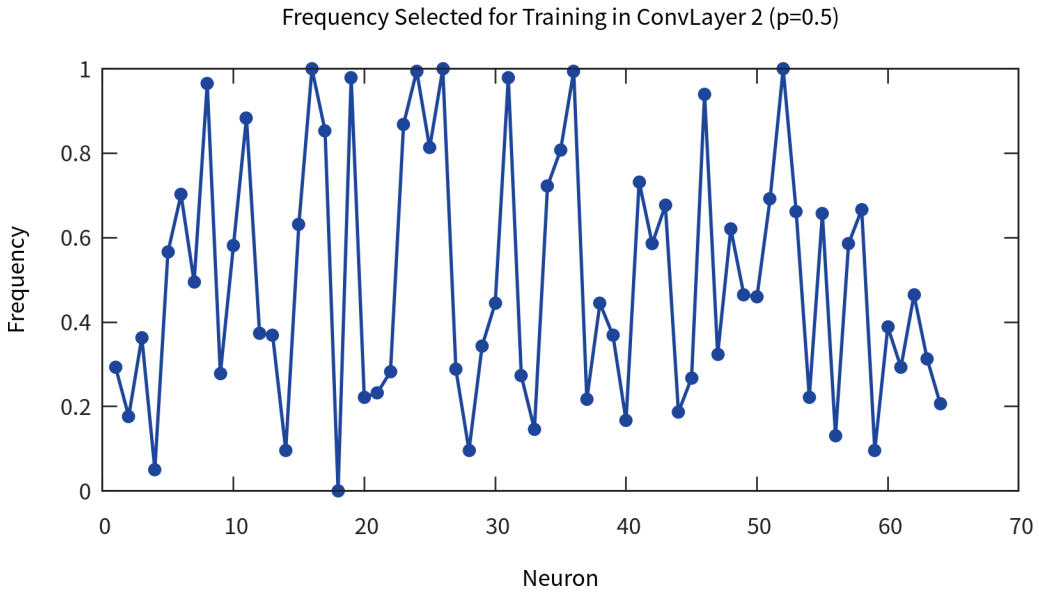

Figure 13: The fooling rate for artificial neurons varied greatly within a single layer. This supports the conjecture that selective dropout could be a valid method of boosting sub-architectures while maintaining the general accuracy of the net at large.

5.3. Selective Dropout

Goodfellow et al sought to interpret the robustness and accuracy of neural networks as a function of invariances of their sub-structures in [4]. Though it is noted in [12] that the features of the input space are best described using the entire basis of that space, the correlation between semantic classes and specific neuron pathways is an intriguing and observable phenomenon which suggests a more precise approach to training along particular pathways or subsets within the network. One difficulty of using neural networks to perform machine learning tasks is that their learned approximations for the real probability space are generally difficult to interpret. It is certainly naïve to assume that a nonlinear classification mechanism such as a deep convolutional neural network can be understood simply as a linear aggregation of its components. However, the proven of *randomized* dropout in preventing over-fitting and the interest in studying the excessive linearity of the network

as characterized by researchers such as Goodfellow and Ng provide adequate motivation for the empirical study which follows.

Visual observations over the activities of hidden layers led to the hypothesis that one may assign blame for incorrect classifications to a small subset of the network. Indeed further studies of the *instability* of neurons (defined as the rate at which certain neurons of a particular layer are fooled and thus repeatedly selected to be retrained over many iterations of selective dropout) varies from [0,1] inclusive. Figure 13 shows that throughout hard-negative training using selective dropout, some neurons were never determined to bear a significant responsibility (and thus are taken to be *stable* elements of the network and are never chosen to be retrained), whereas some volatile to the point of being selected every time. It was hypothesized that by selecting these neurons for targeted training, one might expand the original classification space to encompass these adversarial examples without injecting excessive bias in the system, thereby avoiding loss in classification accuracy on natural, normally behaving data.

The intuition behind selective dropout is simple. If a relatively small substructure of the network demonstrates a disproportionately high error rate when labelling a certain class of adversarial images, then most of the gains in classification accuracy through hard negative mining may be made from specifically training these components. Similarly, by masking a large portion of the network while conducting this training, less bias will be introduced, which in theory will protect the high accuracy of the network on the larger, naturally occurring test space.

5.4. Testing

The set of original-adversarial image pairs was divided into training and validation sets using three different methods of defining *perturbation groups* which each depended on the original and adversarial labels. It is important to validate the ability of the network to generalize on new adversarial images, some of which were generated from perturbations groups on which the networks have not been trained. For each method, a test set was selected by taking one random original-adversarial image pair from each group (including those of size 1, effectively preventing the network

from directly learning that group). The networks were then trained using the remainder of the 208 original-adversarial image pairs generated prior to testing. At time $t = 0$, the control and networks SD_p (where p is the proportion of neurons independently selected to be trained at each step), were identical copies of the CNN described in section 2.2.

Hard-negative training was then conducted on the network. In order to prevent all networks from learning the adversarial space while forgetting the original image space, stochastic gradient descent was performed, alternating between adversarial examples and their original counterparts. Using the majority of rational hyperparameters, this effectively reduced the nets' regression from perfect classification accuracy on the original images after learning an adversarial space. Cross validation using different sets of hyperparameters was performed in order to better understand the impact of selective dropout on the network. Important parameters which were tested include the learning rate, maximum number of epochs over the training set, grouping method, proportion of neurons dropped, and layer of focus. Networks were evaluated based on the classification accuracy on the withheld validation set.

5.5. Results

A summary of the final classification accuracy of the models tested is presented in tables 1, 2, and 3. Models are represented as M_p^l where $M = [C, SD]$ differentiates the control from those trained using selective dropout, p is the proportion of values retained to be trained at each iteration, and $l = [c, f]$ indicates whether the target layer is the 2^{nd} convolutional layer or the fully-connected layer respectively. It was discovered that for certain initial learning rates (i.e. 0.001), selective dropout with $p = 0.2$ for the second convolutional layer learned faster and converged to a better final generalization accuracy than the control even when doubling the number of epochs over the training set. Similarly, selective dropout with $p = 0.7$ on the fully connected layer significantly outperformed the control as well as networks trained while retaining fewer values. Initializing to a higher learning rate seems to reduce the effectiveness of selective dropout, as is evidenced by the relatively poor performance vis-á-vis other learning regimes. This can be explained once more by

recognizing the linearity of the network. If the weights of a subset of the network are excessively altered in a single step in time, the linear combination of those weights becomes too erratic, and the descent meanders in a jagged manner with no guarantee of convergence.

Table 1: Results using perturbation groups defined by both the source and target digit class.

Grouping = Both	C^c	$SD^c_{0.2}$	$SD^c_{0.5}$	$SD^c_{0.7}$	C^f	$SD^f_{0.2}$	$SD^f_{0.5}$	$SD^f_{0.7}$
η=0.001	0.4634	**0.5854**	0.4634	0.4756	0.4634	0.4390	0.4634	0.4878
η=0.003	0.5854	0.5610	0.4512	0.5000	0.5854	**0.5976**	**0.5976**	0.5488

Table 2: Results using perturbation groups only defined by the original digit label.

Grouping = Orig	C^c	$SD^c_{0.2}$	$SD^c_{0.5}$	$SD^c_{0.7}$	C^f	$SD^f_{0.2}$	$SD^f_{0.5}$	$SD^f_{0.7}$
η=0.001	0.1000	0.3000	**0.6000**	0.4000	0.1000	0.4000	0.4000	0.4000
η=0.003	0.7000	0.7000	0.7000	0.6000	0.7000	**0.8000**	**0.8000**	0.5000

Table 3: Results using perturbation groups only defined by the target (spoofed) digit label.

Grouping = Adv	C^c	$SD^c_{0.2}$	$SD^c_{0.5}$	$SD^c_{0.7}$	C^f	$SD^f_{0.2}$	$SD^f_{0.5}$	$SD^f_{0.7}$
η=0.001	0.4634	**0.5854**	0.4634	0.4756	0.4634	0.4512	0.4268	0.4634
η=0.003	0.5854	0.5610	0.4512	0.5000	0.5854	**0.5976**	**0.5976**	0.5488

5.6. Discussion

From the results above, it is difficult to conclude with certainty whether or not selective dropout on a single layer is an effective means of building a robust classifier *in general*. However, when paired with training on hard negatives, it has proven to be a surprisingly effective method of training a robust system which generalizes to images of similar perturbation groups on low-dimensional inputs. For nearly every perturbation grouping method and learning rate (including those not shown above), selective dropout with some value $p < 1$ (where $p = 1$ is simply the control) outperformed the baseline in the final analysis. Furthermore, in almost every case, the networks trained using selective dropout converged to their new local optimum in many fewer steps than the control (usually converging 2-3 times faster than the baseline). Though the final generalization accuracy still is not optimal, the test results indicate that selective dropout consistently performs better than existing methods of training on hard negatives.

While simultaneously training on the original images associated with generated ones keeps the accuracy of the network's predictions on the original images close to 100% for the control and all selective dropout groups, it was observed that the classification accuracy for the control net frequently experienced a minor reversal, ending around 98%. This never (or at least much more rarely) occurs for networks trained using selective dropout with $p \leq 0.7$. This supports the hypothesis that certain precautions must be taken when conducting hard-negative mining in order to make sure that adversarial images are correctly incorporated into the correctly classified space without forgetting the proper predictions for the original images.

6. Future Work

It is difficult *a priori* to determine the proper proportion $p \in (0,1]$ to choose in order to maximize network accuracy. However, it was noticed during training that approximate proportions could be determined based on the type of layer (convolutional vs fully connected) and the depth in the network. Fully connected layers tended to perform better when higher proportions of neurons are retained for training. This is most likely a result of the heavy reliance of this type of layer on large numbers of linear combinations to make certain predictions. Feature maps store much richer information when compared to a single perceptron, so it may be the case that these larger units actually do encode a non-negligible amount of learned information independently or in very small groups. While this observation may be sufficient for a blind cross-validated search, it is likely that p may be estimated based on the distributions of the Frobenius norm of difference of neuron activations over a layer using only a small subset of adversarial examples. By initializing p_0 as a function of the variance of activation volatility across a layer, one might linearly separate the neurons, thereby selecting an optimal p such that neurons retained for training bear the majority of the responsibility for misclassifications of a particular perturbation group. Given the convergence speed bonus when training using selective dropout, even if the use of selective dropout using $p_0 < 1$ converges to a sub-optimal stable point (when compared to the control), a hybrid approach to training could be effective using a negative biased exponential decay. The term p_t at time t can be

derived using the equation $p_t = p_0 + (1 - p_0)(1 - e^{-\alpha t} = 1 - e^{-\alpha t} + p_0 e^{-\alpha t}$, where p_0 indicates the initial proportion, and α is a hyperparemeter which controls the rate at which p approaches 1. The number of retained neurons would be taken to be $\lceil p * \text{width}_k \rceil$ for layer k, where $\lceil\ \rceil$ is the ceiling function. This could potentially reduce the time required to perform hard-negative training while strengthening the network by slowly distributing blame to a larger subset of the network. In short, selective dropout will boost the robustness of the network by balancing high accuracy and quick convergence at the beginning of training, and only after reinforcing the proper behavior of the most volatile neurons does the net begin to assign blame across the full span of the network.

6.1. Cross-Generalization

As noted in [11], adversarial images tend to fool networks of varying architectures and training experiences. Because of this, it is vital that selective-dropout is tested on larger and deeper networks to determine its scalability across diverse architectures designed for disparate tasks.

6.2. Pathway Identification

Parallel neurons in a single layer are normally understood as encoding similar levels of complexity of semantic information. Due to this and the linear relationship between elements in the image of a layer, neurons may be analyzed using naïve comparisons of the Frobenius norm of the difference in activity when introduced to different correlated stimuli. While this metric seemed to provide some level of clarity, a more mathematically rigorous approach ought to be taken in order to more accurately measure relationships within a layer, thereby identifying neuron *clusters* to be retained for training. Expanding on this work, one might adapt selective dropout (SD) to select subsets based on pseudo-Hebbian methods, where neurons are grouped based on relationships of activities under certain stimuli.

Since each layer computes a linear combination of the inner product (or convolution) of the weights and signal from the previous layer (passed through a nonlinearity), it is likely that pathways may be identified which tend to be activated with high probability compared to other subsets of the network. It would be informative to study correlations within and between layers in a more focused

and rigorous manner so that tests of SD t applied to larger sub-structures might by conducted. If sufficiently interpretable results can be drawn from these tests, recurrent connections might be introduced within layers and between clusters which may help make the net more robust to adversries using simple gradient methods to perturb images.

6.3. Darwinian Testing

An additional way of testing the robustness of networks trained using SD is by comparing performance of the network over time when pitted against an adaptive adversary. The set of perturbed images used in this paper was small and created prior to all tests, so it is difficult to draw conclusions about the manner in which the difficulty of generating such images evolves over time.

To address this, one might create an L-BFGS (or fast gradient) adversary to create perturbed images optimized to fool the network at time t. Allowing the network to train (via SD) on these inputs may teach the network to slowly shrink the total adversarial space, forcing the adversary to generate images using noise of increasing magnitude before being able to fool the network. Comparing difficulties in this way on control and SD networks would be informative both in characterizing selective dropout as well as to better understanding the way a CNN learns (and forgets) correct input spaces. .

6.4. Beyond MNIST

As was discussed in section 3.1, the size of the adversarial manifold increases in a manner linearly proportional to the dimensionality of the input space. Since most real-world tasks will require computer vision systems to process high-definition images of complex scenes, it is important that SD be tested on appropriately sized images. One may port the framework developed for the MNIST data set to train and test nets on the CIFR-10, CIFR-100, and ImageNet databases (using networks appropriately structured for the given task). It is likely that the benefits of SD will scale with the dimensionality of the problem and the size of the network, but it is possible that the increased complexity of the nonlinear relationships deep in the network would require a more advanced retention policy as mentioned in section 6.2.

7. Conclusion

The initial goal was simply to reproduce the findings of [11] and [13] by creating adversarial images on the MNIST database to fool a pre-trained, highly accurate convolutional neural network. Not only was this easily accomplished, but also visualization techniques were adapted to localize failures to maintain presumed invariances within the network to individual (and small groups of) neurons within a layer. This enables one to identify neurons which are particularly volatile and susceptible to adversarial manipulation. The study of these visualizations over particular perturbation groups inspired the development of a new training mechanism, *selective dropout* (after the common training method of dropout proposed by [10]), which encourages the network to effectively learn proper predictions on pre-generated adversarial images without forgetting the correct classification space of naturally occurring (i.e. non-adversarial) images. While further work must be done in order to determine whether the results of these experiments may be generalized to higher-dimensional problems and deeper networks, the results on the test set indicate that the stochastic nature of selective dropout makes it a useful tool for training more reliable computer vision systems.

References

[1] R. H. Byrd *et al.*, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[2] J. Deng *et al.*, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[3] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *ArXiv e-prints*, Dec. 2014.

[4] I. Goodfellow *et al.*, "Measuring invariances in deep networks," in *Advances in neural information processing systems*, 2009, pp. 646–654.

[5] A. Haghighi, "Numerical optimization: Understanding l-bfgs," Dec. 2014. Available: http://aria42.com/blog/2014/12/understanding-lbfgs

[6] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[7] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. Available: http://yann.lecun.com/exdb/mnist/

[8] A. M. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," *CoRR*, vol. abs/1412.1897, 2014. Available: http://arxiv.org/abs/1412.1897

[9] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan 2016, article. Available: http://dx.doi.org/10.1038/nature16961

[10] N. Srivastava *et al.*, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[11] C. Szegedy *et al.*, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[12] C. Szegedy *et al.*, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2013. Available: http://arxiv.org/abs/1312.6199

[13] P. Tabacof and E. Valle, "Exploring the space of adversarial images," *CoRR*, vol. abs/1510.05328, 2015. Available: http://arxiv.org/abs/1510.05328

[14] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. Available: http://arxiv.org/abs/1311.2901