AN ABSTRACT OF THE THESIS OF

Xin Li for the degree of Master of Science in Electric and Computer Engineering presented on September 26, 2016.

Title: Don't Fool Me: Detecting Adversarial Examples in Deep Networks.

Abstract approved: _____

Fuxin Li

Deep learning has greatly improved visual recognition in recent years. However, recent research has shown that there exist many adversarial examples that can negatively impact the performance of such an architecture. Different from previous perspectives that focus on improving the classifiers to detect the adversarial examples, this work focuses on detecting those adversarial examples by analyzing whether they come from the same distribution as the normal examples. An approach is proposed based on spectral analysis deeply inside the network. The insights gained from such an approach help to develop a comprehensive framework that can detect almost all the adversarial examples. After detecting adversarial examples, we show that many of them can be recovered by simply performing a small average filter on the image. Those findings should provoke us to think more about the classification mechanisms in deep convolutional neural networks.

Don't Fool Me: Detecting Adversarial Examples in Deep Networks

by
Xin Li

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented September 26, 2016
Commencement June 2017

Master of Science thesis of Xin Li presented on September 26, 2016

APPROVED:

Major Professor, representing Electric and Computer Engineering

---

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Xin Li, Author

ACKNOWLEDGEMENTS

First, I would like to thank my major advisor Dr. Li, Fuxin for providing me support and guidance during my study. I really appreciate an opportunity to work with him and I considerably explored the filed of study of my interest with his help. Great thanks for being especially supportive, understanding and patient during the my final term in OSU. Every single question and confusion I have was answered by Dr. Li, Fuxin in a rather detailed way. He is the person who wisely and tremendously provided help that enlightened me during my research and my learning process.

Second, I would like to thank Dr. Ramsey, Stephen, Dr. Bobba, Rakesh and Dr. Jansen, Henri J. F. for agreeing to serve on my graduate committee. Thank you for giving me this opportunity.

Third, I want to thank the Department of Electrical Engineering and Computer Science at Oregon State University for supporting me during my study.

Finally, I want to thank my dear parents for my life, their love, support and encouragement. I want to thank my Graduate Coordinator Thompson, Nicole and my Administrative Program Assistant Miller, Darcy. Both of them did great contribution to my defense application process. I also want to thank my friends Zheng Zhou who helped me a lot in some technical issue during my research. And I would show great thanks for all my friends in Corvallis who stayed with me during my graduate study.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

To the memory of my Papa, and to my Mama for your support and encouragement;
to all my friends in Corvallis for your care and company.

Don't Fool Me: Detecting Adversarial Examples in Deep Networks

1 Introduction

1.1 Overview

Recent advances in deep learning have greatly improved the capability to recognize images automatically. State-of-the-art neural networks perform better than human on some difficult, large-scale image classification tasks in complex datasets. Krizhevsky et. al. in 2012 proposed **AlexNet**[4] which has a similar but deeper architecture compared the classic CNN, **LeNet-5**. Equipped with highly-efficient **2-D** convolution algorithm, which employ GPU parallelism, this model can train on huge scale dataset in a fast way. With their dropout and data augmentation method, they avoided the overfitting problem of the huge architecture. In 2014, Simonyan et. al. explored how the depth of a convolutional neuron network affects its performance. Using small 3×3 filter size, hey proposed their **VGGNet**[2] with 19-layer version and 16-layer version architecture, which significantly increased the accuracy and is generalized to multiple datasets. Their work shows that the performance will increase when the number of layers is increasing. Later in 2016, He et. al. proposed **ResNet**[6]. This artifact is a countermeasure for accuracy loss cause by exceedingly adding layers of neurons to a network. This problem is addressed by introducing a residual learning architecture to a plain convolutional neural network. And, using stochastic gradient descent, their network can be easily trained with backpropagation.

However, an interesting discovery has been that those networks, albeit very resistant to overfitting, would have completely failed if one perturbs some of the pixels in the image via an adversarial optimization algorithm. In 2013, Szegedy et. al. has lauched thier adversarial optimization altorithm[1] to generate image perturbation

that can mislead the networks to assign incorrect category to a given image. Those perturbations make changes to images, which is indistinguishable from the original to a human observer. (Fig. 1.1). In 2014, Goodfellow et. al. proposed a linear explanation to Szegedy's work, and introduced their fast gradient sign method that can generate adversarial examples efficiently to facilitate the their adversarial training which helps increase the generalization of the model[7]. After that, many related researches[8, 9] are conducted and get more insights into the adversarial examples.

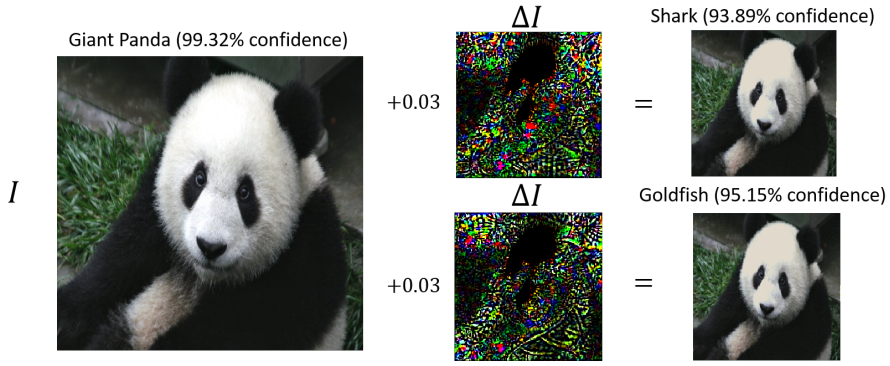


FIGURE 1.1: An optimization algorithm can find the adversarial example where, with almost negligible perturbations to human eyes, will completely distort the prediction result of a deep neural network [1]. This algorithm is quite universal, having been successfully tested on many different networks and the user can direct the network to output any category with adversarial optimization.

Those adversarial examples are dangerous if one would put a deep network into any crucial real application. For example, in autonomous driving, adversarial example would fool the driving system to let it consider a part of the road as clear, but, in fact, some pedestrians are walking across. This would result in serious traffic accident. In precision line, Robot fooled by adversarial examples would cause severe financial loss or even produce the products that not safe for customers. Security issue

would arise when applying adversarial example to automatic recognition for confirming identity (face, iris, speech, etc.). If the result of the network can be hacked at the will of a hacker, crashes, wrong authentications and other devastating effects would be unavoidable.

Therefore, there are ample reasons to believe that it is important to identify when is a prediction highly confident. Such knowledge if available will help significantly to control behaviors of robots employing deep learning. If a reliable procedure exists and is hard-wired into the robot, it can prevent the robot from behaving in manners undesirable from human because of the false perceptions it made about the environment.

This is similar to many previous theoretical work such as [10], however has never quite been applied in computer vision because of the immensely high dimensionality and sparsity of data. It is well related to a basic dichotomy of machine learning, or every function estimation problem in mathematics, statistics and physics: *interpolation vs. extrapolation*. It is well-known that if the data to be predicted lies within a subspace, convex hull, or manifold spanned from the training data, then it is more likely to succeed as the prediction problem can be characterized as interpolating from some training data nearby. Extrapolation is much harder, since it is extremely difficult to estimate data labels or statistics if the data comes as extremely different from any known or learned observations. Therefore, in many cases it might be better to abstain an extrapolation prediction. In statistical machine learning terminology, the *i.i.d.* assumption is commonly used, so that the testing examples are assumed to be drawn independently from the same distribution of the training examples. In general, this precludes extrapolation, since there is a decent probability that some training examples exist in some neighborhood of a testing example, given the *i.i.d.* assumption.

Although these concepts are well-known, the difficulties of understanding and applying them lie in the high-dimensional spaces that are routinely used in machine

learning and especially deep learning. Is it even possible to define interpolation vs. extrapolation in a $4,000$-dimensional or $40,000$-dimensional space? It looks like almost everything is extrapolation since the data is inherently sparse in such a high-dimensional space [11, 12], a phenomenon well-known as the curse of dimensionality. The enforcement of the *i.i.d.* assumption seems impossible in such a high-dimensional space, because the inverse problem of estimating the joint distribution requires an exponential number of examples to be solved efficiently.

1.2 Contribution

Intuitions drawn from previous work [1, 7] show that deep learning finds manifolds of much lower dimensionality and utilizes those low-dimensional manifolds to build efficient learning machines. In this work, we use these trained networks and their highly nonlinear transformations to run a number of empirical studies deep inside the network, in order to visualize how the adversarial examples change the prediction of the deep network, and to conclude that the located adversarial examples are likely to be from a different distribution, using simple spectral dimensionality reduction techniques such as PCA.

From those intuitions, we develop statistical estimates of whether an image belong to the normal image distribution using sampling from inside the trained deep neural networks. It turns out that these confidence estimates can almost perfectly separate between the known normal and known adversarial examples. Those confidence estimates may have applications in controlling the network behavior and preventing disastrous effects, and hopefully pave the way for further empirical and theoretical research on self-aware learning. In this particular problem setting, we explore recovering the original prediction from the corrupted adversarial example after we detect it, and can achieve reasonably good accuracy even with a simple average filter on the image.

As a result, our contribution can be listed below:

- We run two different classification model, specifically **AlexNet** and **VGGNet**, on normal image dataset and adversarial example dataset that is generated using **L-BFGS** optimization algorithm by Szegedy et. al.. And them we visualized the activation layer by layer by applying PCA dimension reduction to find how adversarial examples differ from normal examples.

- By the intuition that normal examples and **L-BFGS** optimized examples belong to different probability distributions, we extracted the statistics of interest from the evaluated activation when data are given to the models. And construct an estimation of whether an example is a normal example or has been perturbed with the adversarial optimization result.

- We construct a detection mechanism for the networks that helps to draw statistics of interest from convolutional layer activation. Use them to compute the estimates and let networks be aware of the undesirable mistake.

- We explored a recovery strategy to recover the images that has been detected as problematic ones to the original images.

2 Background

2.1 Deep Neural Network

Deep neural network (DNN) is a computational paradigm inspired by biological neural networks, though recent study shows that the working mechanism of the DNN is fundamentally different from biological neural networks [13], which are used to estimate or approximately express any functions that can depend on a high-dimensional inputs that are generally unknown, for the purpose of feature extraction and classification.

2.1.1 Basic Structure Overview

A basic deep neural network consists of four basic components: Inputs, neurons, perceptron weights and outputs. As shown in **fig.2.1**, DNN structure has its input x_i and output $\hat{y}$ on the two ends of the network respectively. The basic computational units in an DNN is called neurons. They are, in **fig.2.1**, is denoted by $a_i^{(l)}$, which means the ith neuron on the $l-th$ layer. The neurons are connected by edges with perceptron weights denoted as $w_{i,j}^{(l)}$, which means the perceptron weights for layer l connecting the jth neuron on layer l to the ith neuron on layer $l+1$. When all the neurons at one layer connect to every single neurons at previous layer, we call this layer the fully connected layer.

2.1.1.1 Neurons and Perceptron

As mentioned above, the neurons are the basic computational units in the DNN. A single neuron is under taking a percrptron work. As shown in **fig.2.2**, when input for a layer come to one neuron, the neuron firstly multiply the inputs with with weights and sum them up, then apply to the sum an activation function ϕ. Hence the output of each neuron is:

$$o_j = \phi\Big(\sum_{i=1}^{n} w_{i,j}x_i\Big) + w_{0,j}^{(l)} \tag{2.1}$$

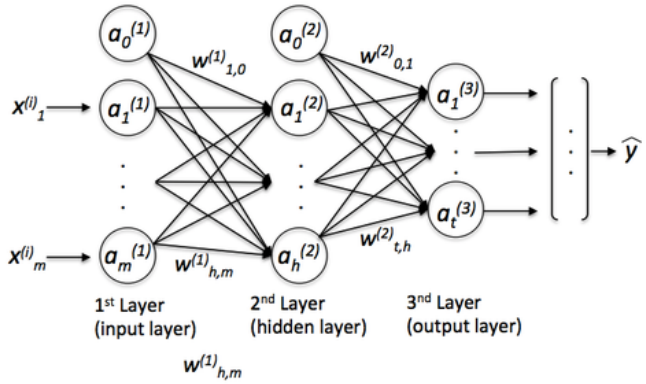


FIGURE 2.1: General Structrue of Neural Network

It is common to write (2.1) in matrix form:

$$o_j = \phi(W_j^T X) + w_0^{(l)} \tag{2.2}$$

where weights W and input X are of the same dimension, and $w_0^{(l)}$ is the bias term.

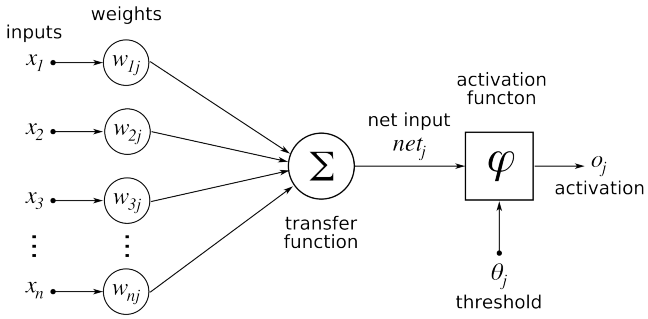


FIGURE 2.2: Perceptron for a Single Neuron

Neurons connected with inputs forms the inputs layer and neurons connected with outputs forms outputs layer. layers between those two are called hidden layers. And neurons in those layers are called hidden units.

2.1.1.2 Multi-dimensional Representation

Usually, one DNN has multiple neurons on one layer as shown in **fig.2.1**. So, the output of network layer l can be expressed as matrix representation for the ease of calculation:

$$\Phi_l = \Phi_l(W^{(l-1)T} X^{(l-1)}) + w_0^{(l)} \tag{2.3}$$

where Φ_l is a n-dimensional output, $W^{(l-1)T}$ is a $m \times n$ (n is the dimension of data) matrix representing the proceptron weight for all the neurons at layer $l-1$. And, also, DNNs have multiple layers of neuron, so we have network output:

$$\hat{Y} = \Phi_N(W^{(N-1)T}\Phi_{N-1}(...(W^{(2)T}\Phi_1(W^{(1)T}X + w_0^{(1)}) + w_0^{(2)}) + w_0^{(N-1)}) + w_0^{(N)} \tag{2.4}$$

2.1.1.3 Activation Functions

The perceptron mentioned in **section.2.1.1.1** calculates an analog output by applying an intended function to the linear combination of the inputs. This function is reffed to as activation function. There are multiple frequently used activation functions. Every Activation functions takes a scalar as its input and maps the scalar to another according to how the criteria is defined mathematically.

- **Sigmoid Function**

  The mathematical form of sigmoid function σ is:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

The curve for sigmoid function is shown in **fig.2.3**. This function takes a real valued input and maps it to a value that is between 0 and 1. It is shown that that input values with large absolute value will be map to approximately to 0 or 1, particularly for large negative and large positive scalars respectively.

A drawback of sigmoid function is that it will saturate when the absolute value becomes extremely large. The gradient of the function at that point will be infinitely

approaching to 0. When applying a back-propagation, computed gradient for the loss function will be very small. An extra attention will need to be paid when initializing the weights to prevent saturation.

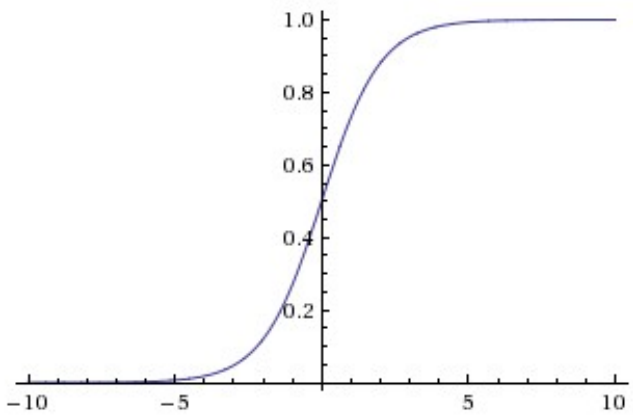


FIGURE 2.3: Sigmoid Function

• **Tanh Function**

The tanh function takes scalar inputs and maps them the value between −1 and +1. Its activation can saturate, but its output will not always be positive. So it is preferred than sigmoid. It is defined as:

$$tanh(x) = \frac{sinh(x)}{cosh(s)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1 \tag{2.6}$$

As we can see, *tanh* function is a scaled sigmoid function.

• **ReLU function**

ReLU[14] function is the rectified linear unit function. It is defined as:

$$ReLU(x) = \max(0, x) \tag{2.7}$$

And its curve is shown in **2.5**. RelU is take a thresh hold at zero and spamming all negative inputs and set corresponding inputs as 0. It becomes popular to be used

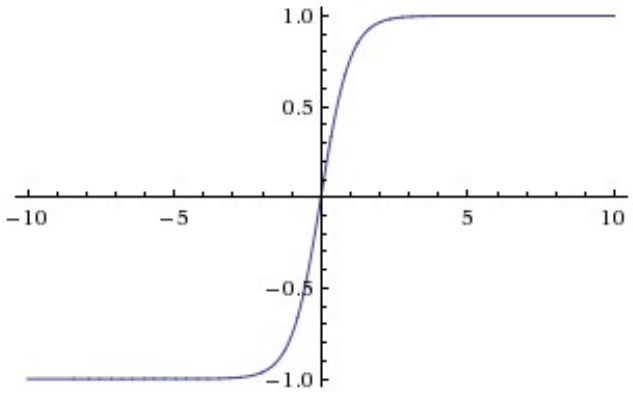


FIGURE 2.4: Tanh Function

in the non-linearity part of the networks because its piecewise-constant sub-gradient lead to easier computational implementation and fast convergence.

ReLU function can get a dramatically faster convergence when applying stochastic gradient descent (SGD), which is due to its linear and non-saturation property. Also, it has a simpler implementation than *tanh* and *sigmoid* function.

2.1.2 Training the Network

In training step, for a general classification or regression problem, we usually optimize a loss function. In definition, a loss function (also termed as cost function) is a function that maps an event or values of one or more variables to a real number. Intuitively, it represents the cost associated with one event. Particularly in machine learning problem, an evens can be abstracted to a set of parameters that form a function.

2.1.2.1 Optimization: Stochastic Gradient Descent

One of the simple ways for optimization is gradient descent. The idea of gradient is

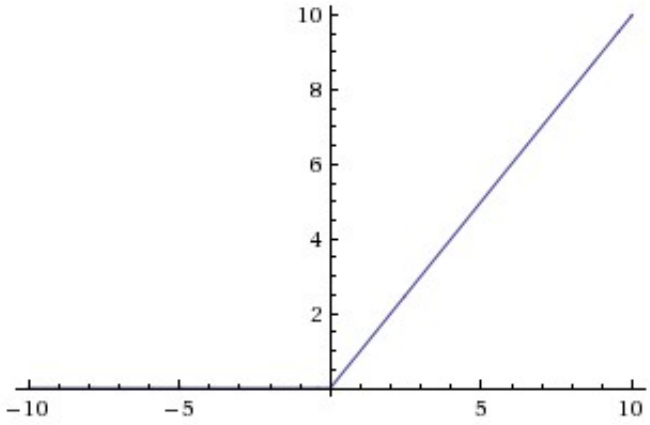


FIGURE 2.5: ReLu Function

that the gradient of a function at one point always has the direction where the value of the function is decreasing. One need to choose a initial position as a start point in the parameter space and calculate gradient at that point with respect to the parameters on which we would like to optimize the loss and update the standing point by adding to it the gradient to move to the next point where the value of the loss is less than that at previous point. One can also scale the gradient to adjust the step size for updating.

Modern implementation of gradient descent is usually batch methods, such as BFGS. At each iteration, it consume the whole dataset to compute the gradient and get converged well on local optimal. However, there are two issue for batch methods. First, it is slow to use whole dataset to compute the gradient. Second, it is not compatible for online learning. Stochastic Gradient Descent (SGD), motivated by a computationally expensive operation, backpropagation, gives the solution to both issue. Each iteration, it only requires one or a portion of data point to compute the gradient.

$$\theta = \theta - \alpha \nabla_\theta J(\theta; x, y) \tag{2.8}$$

where J is the loss function and α is the learning rate. Usually, SGD takes a smaller learning rate compared with the standard batch methods because SGD has much more variance in the update direction than that of a regular gradient descent.

2.1.2.2 Backpropagation

As we talked about in **2.1.1.2**, the output of a neural network has the form in **(2.4)**. When applying the gradient descent, one will have to take gradient to weights layer by layer so that weights on all layers can be updated. These operations require one to apply chain rule to the loss function which is a set of activation functions that are nested together.

So, when calculating the gradient on the ith layer. We have:

$$\frac{\partial J}{\partial W_i} = \frac{\partial J}{\partial f_N} \cdot \frac{\partial f_N}{\partial f_{N-1}} \cdot \ldots \cdot \frac{\partial f_{i+1}}{\partial f_i} \cdot \Phi_i(f_{i-1}(x)) \tag{2.9}$$

where:

$$f_k(x) = W^{(k)T}\Phi_k(f_{k-1}) + a_0^{(k)} \tag{2.10}$$

$$f_0(0) = x \tag{2.11}$$

As we can see in (2.9) that, if we need the gradient at ith layer, we need to have all gradients from above the ith layer calculated. That is, every time when we obtained the gradient from one layer, we need to store them for future use.

There are a few thing we need to notice. Backpropagation algorithm does not guarantees convergence. It is possible for the algorithm to oscillate around the optimal point or reach a local optimal point. To avoid stopping at the local optimal, one can run multiple experiments with different starting point, then choose the one with best training or validation performance, or build a committee of networks that can vote during testing process.

2.2 Convolutional Neural Network

In this section we will explain the basic concepts of a deep convolutional network that is commonly used in visual recognition and will be used in the experiments of this work.

The Convolutional Neural Networks (CNN) consists of neurons that are associated with one another to form one or more convolutional layers, pooling layers and fully connected layers (**fig. 2.6**). A convolutional neural network naturally exploits the locality structure in data. In an image, pixels that are located close to each other are more likely correlated than pixels that are far away, same holds for temporal data (video, speech) where objects (frames, utterances) that are temporally close can be assumed to be more correlated. This structure also makes CNNs easier to train because there are fewer parameters than those in a fully connected neural network with same number of hidden units. The reason is that the weights used for producing one type of feature in a convolutional layer are shared by the whole inputs. Most deep networks adopt similar principles while varying in the structural complexity of the system, such as adding more layers and smaller filters in each layer [2], multi-layered network within each layer [15], combining multiple networks [16], etc.

In my experiment, I used the AlexNet and 16-layer VGG network model in **section.6**. VGG with 16 layers contains 14 convolutional layers, each contains an increasing number of 3×3 filters **fig.2.6**. It also has 5 max-pooling layers in between the convolutional layers, and 2 fully connected layers before the classification layer. The task that it performs is ImageNet classification [17], to classify natural images in a photo collection into $1,000$ different categories. It was trained on a training set of 1.2 million images and has a testing top-5 error rate of 13.5% (Additional better results have been achieved by training a 19-layer network with multiple crops and batches of data, as well as combining multiple models by bagging). AlexNet has 5 convolutional layer, which is a shallower architecture, but is a great progress since LeNet-5.

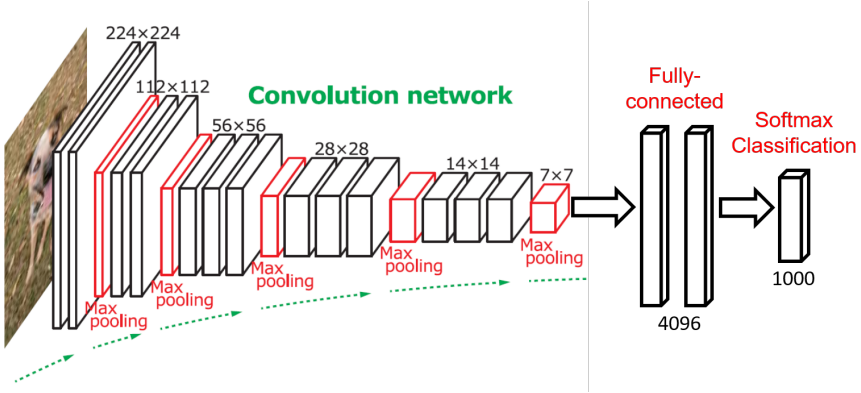


FIGURE 2.6: (a) A sample configuration of a state-of-the-art deep convolutional network architecture ([2], figure modified from [3]). The network consists of convolutional layers (convx-y indicates y convolutional filters of size x), max pooling layers (maxpool) [4] and fully connected layers (FC, where nodes are connected to every node from the previous layer. Finally, a soft-max layer computes a logistic loss for classification. ReLU nonlinearity is used for every layer.

2.2.1 Convolution

Convolution is an operation on two functions f and g, which produces a third function that can be interpreted as a filtered f by g. In this interpretation we call g the filter. If f is defined on a spatial variable like x, we call the operation spatial convolution. Formally, for functions $f(x)$ and $g(x)$ of a continuous variable x, convolution is defined as:

$$conv(x) = f(x) * g(x) = \int_{-\infty}^{+\infty} f(\theta)g(x-\theta)d\theta \tag{2.12}$$

Notice that, under the integral, function f and g are functions of θ, but the result is still a function of x. Function $g(x-\theta)$ is obtained by shifting to the right the reflection of $g(\theta)$ by x. Hence, the value of the convolution result will change while g sliding through the axis from $-\infty$ to $+\infty$.

However, most of the time, our inputs are discrete. The convolution of two

discrete function $f[x]$ and $g[x]$ is defined as:

$$conv[x] = \sum_{k=-\infty}^{+\infty} f[k]g[x-k] \tag{2.13}$$

The length of the function is usually finite. So, the upper and lower bound of the summation is usually from 0 to an integer N.

Convolution can be generated to be a 2-D operation, such as in images processing. The 2-D convolutions can be expressed as:

$$conv(x,y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(\theta_1,\theta_2)g(x-\theta_1,y-\theta_2)d\theta_1 d\theta_2 \tag{2.14}$$

The discrete form is:

$$conv[x,y] = \sum_{n_1=-\infty}^{+\infty} \sum_{n_2=-\infty}^{+\infty} f[n_1,n_2]g[x-n_1,y-n_2] \tag{2.15}$$

Similar to the interpretation of the one dimensional form, the value of the 2-D convolution will change while the filter matrix g sliding on the matrix f.

In digital image processing, input functions are defined 2-dimensional space and will be discrete. Convolution will be applied in the form of (2.15). It is useful in edge finding, feature detection, motion detection, image matching, and countless other tasks.

2.2.2 Convolution in Digital Image Processing

The statistical properties in one image is homogeneous. Hence we can extracted same features from all parts of the image to process. It means, if we have a $N \times N$ feature detector patch, we can apply it to any patches randomly sampled form the image. By applying convolution between the image and the feature detector patch, we can obtained desired feature from all over the locations.

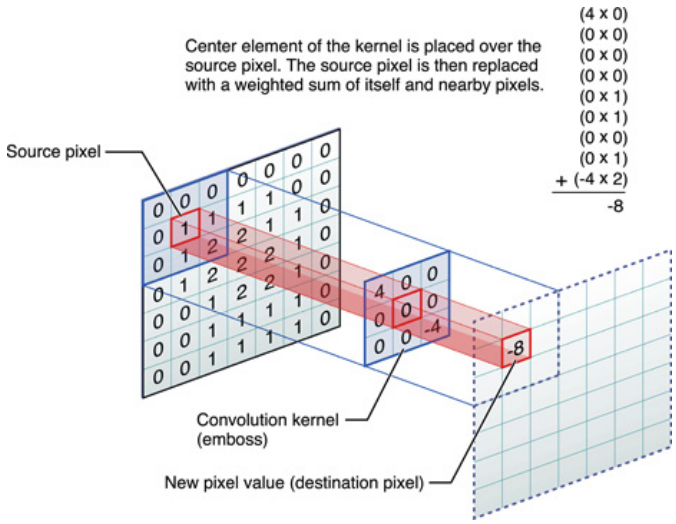


FIGURE 2.7: Convolution on Images

As an example shown in **fig.2.7**, we have a 3×3 feature detector patch, also called kernel or filter. The filter will sliding through the image with a predefined step size, also known as stride. Every time when the filter arrive at a location, we will calculate convolution for the pixel located at the center of the filter. For a $N \times N$ filter working on a gray-scale image, the convolution at pixel (x, y) can be calculated using (2.15). For simplicity, one can view the value of the convolution as the summation of the products of pixel value covered by the filter and the value on the filter at corresponding position. To prevent images from shrinking in their size too much, we usually apply padding zeros to the images on their edges.

2.2.3 Architecture of Convolutional Neural Networks

As shown in **fig.2.8**, Convolutional neural netoworks usually have a specialized connectivity structure between neurons. It stacks multiple layers in the feature extraction stage. Stages closer to the output will operate on more global level and more invariant

features. A classification stage is stacked at the end.

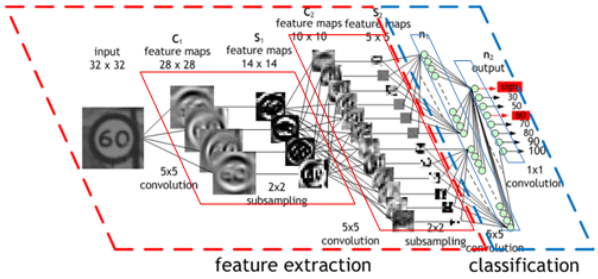


FIGURE 2.8: General Architecture of CNN

• **Convolutional Layer**

Convolutional layers are commonly used in computer vision that only connect spatial or temporally close nodes to the next layer. Compared with fully connected layers, convolutional layers are locally-connected and have additional weight sharing.

In **fig.2.8**, one convolutional layer usually contains multiple filters. They will apply convolution, which is in the form of (2.15) and visualized in **fig.2.7** will be apply to each input with one and each filter respectively. The inputs, like images, usually have the dimension of $m \times n \times r$, where $m \times n$ would be the size of image and r is the number of channels of a image. Adding to the convolution output some non-linearity, the output of a convolutional layer i can be:

$$Y_i = [T(f_1 * Z_i), T(f_2 * Z_i), \ldots, T(f_k * Z_i)] \tag{2.16}$$

where Z_i is the input of layer i and $f_1, ..., f_k$ are filters of size much smaller than the input on this layer and $*$ is the convolution operator. This network would apply the same convolutional filters at every location of Z_i, which means that the number of parameters is much smaller than if each filter is the same size as input.

The filters can be of size $x \times y \times z$ where $x \times y$ is the size of the filter that should be smaller than the image, and z can be less or equal to the number of size of the

input. If there are k filters on one convolutional layer, the operation of convolution will result in k new result images which is called feature maps because each filters is to extract one kind of feature.

• **Pooling Layer**

Theoretically, the output of the convolutional layer can be used for training and testing the classifier directly. However, the output of convolutional layer always has very high dimensionality that will make the model computationally expensive. For instance, we have a 224×224 images. After one layer of convolution with 90 filters to extract 90 types of features. The operation will give a result of dimension $224 \times 224 \times 90 = 4515840$. Working on the feature vector with such high dimension will make the classifier prone to over-fitting.

As mentioned in **2.2.2**, the images are usually statistically homogeneous. This also means one type of statistics would be useful for all parts of images. By aggregating the important statistics, we can reduce the dimensionality and also improve the results. This operation is called pooling. By the type of statistics to extract, they are named max pooling, min pooling or mean pooling. There is an example of max pooling in **fig.2.9**. Max-pooling selects the maximal output from a small region, in order to allow minor deformations for prominent features detected from the convolutional filter.

Also, pooling operation has a property which is termed as translate invariant [18]. If the pooling patches are contiguous area and statistics are only extracted from the same units, then, these pooling units will then be translation invariant, which means the same feature from the convolution output should still be active after the translation operation. So we can say operation in **fig.2.9** is translation invariant.

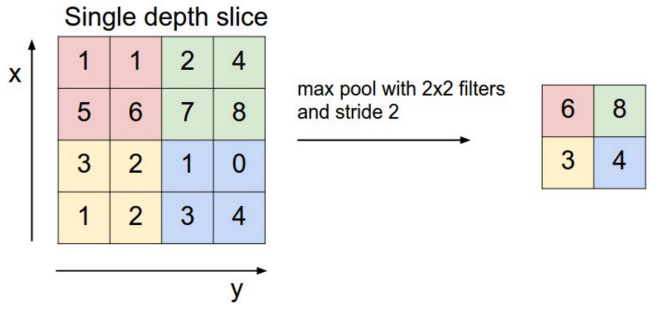


FIGURE 2.9: Max Pooling

Other types of layers are also used in a CNN, for example, the normalization layers [4] such as batch normalization layers[19]. If all the nodes in a layer are connected to all the nodes in the next layer, the network is called *fully-connected*.

2.3 State-of-the-Art CNNs

Firstly inspired by the biological visual perception mechanism[20], many architectures of the CNN had been proposed. Hubel and wiesel find the cells that act as the probe for detecting the lights in 1959[21]. LeCun et. al. formed the fundamental framework of CNN, LeNet [22][23]. LeNet-5 consists of 7 layers, which are convolutional layers C_i, pooling layers S_i and fully connected layers F_i, see **fig.2.10**. The convolutional C_1 and C_3 contains 6 and 16 filters respectively. And pooling layers should always generate same number of feature maps with its previous layer. The input of the network will be prepossessed to a 32×32 image which is significantly larger than the area occupied by its largest character to make the evident features like stroke end-points appear near the center of the receptive filed. The 32×32 input is also normalized before being processed to accelerate learning process.

They conducted multiple experiments on $100,000$ hand-written characters (up-

per case and lower case, digits and punctuation) which are from 95 classes. They have an pretty good overall performance error rates are 2.99% for upper case, 4.15% for lower case, 1.4% for digits and 4.3% for punctuation.

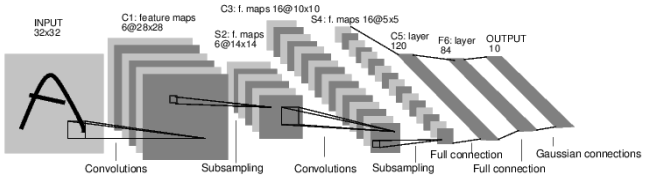


FIGURE 2.10: LeNet-5

Many new architectures has been proposed and developed since 2006 after LeNet:

• AlexNet

AlexNet is proposed by Krizhevsky et. al. [4] in 2012. It has a similar but deeper architecture compared to LeNet-5, see in **fig.2.11**. AlexNet contains five convolutional layers and three fully connected layers and pooling layers. Cross-GPU parallelism enable them to train a large scale network that cannot fit into the memory of a single GPU.

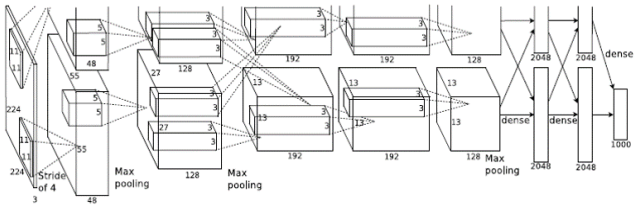


FIGURE 2.11: Architecture of AlexNet

Configured with non-saturating non-linearity, $ReLU$ function, in the non-linearity

part, AlexNet has a faster learning process on large dataset than that when configured with traditional saturating non-linearity, e.g., *Sigmoid* function and *tanh* function. Though *ReLU* does not require normalized input image because of the non-saturation properties, their local localization applied to the training process turned out facilitates the model to generalize.

They also configured AlexNet with overlapping pooling method throughout the network. This strategy enables them to reduce the error rate by up to 0.4%, and also prevents over-fitting, compared with non-overlapping pooling method.

• VGGNet

In 2014, Simonyan et. al. [2] in Visual Deep Group proposed VGGNet. Also in the area of computer vision, their attempt is to explore the influence that is brought by the depth of the CNN. They fixed other parameters of a CNN, such as the size of receptive filed[24], the number of filters on each convolutional layer or the scale under which the detection operates[25].

In their experiment, the network accepts the fixed RGB image of size 224×224×3 subtracted by the mean of RGB dimension obtained from training dataset. They used filters of size 3×3 filters with convolution stride equals to 1 for all convolutional layers. Spatial resolution is preserved by padding 1 pixel at the edge of the input of the convolution. They used max-pooling method in all pooling layers with pooling window of size 2×2.

After feature detection layers, they stacked two fully-connected layers with 4096 channels and one fully-connected layer with $1000 - way$ softmax for $1000 - categories$ classification task.

ReLU activation was used for all hidden layers to provide non-linearity. Local Response Normalization [26] does not increase the performance of their architecture and increase the memory consumption.

During their experiment, they applied multiple training and testing process to different configurations of their CNN. By adding or eliminating convolutional layers

between max-pooing layers, they increased or decreased the depth of their CNN. One of the benefits that comes from this is that multiple layers of stacked convolutional layer with filters of one size can express the function of one single convolutional layer with filters of another size. For example, three 3×3-receptive-filed convolutional layer can be stacked together to express a 7×7-receptive-filed layer. And, moreover, the stacked layers consists of less parameters than the single layer. If there are C channels for each layer, there will be $3 \times (3^2C^2)$ parameter for the stacked layers and 7^2C^2 parameters for a single larger-filter layer, which is much more than that of stacked layers.

The additional 1×1 convolutional layers are configured to some of the experiments and were found to be a way to provide more non-linearity without effect the effective receptive size provided by the stacked convolutional layers.

Tested on ILSVRC-2012 dataset and evaluated under different settings, such as different scale and different type of corps, they showed that the increasing in depth can bring benefits for classification accuracy.

• ResNet

Kaiming et. al., in 2015, explored to find whether one network is necessarily better to have more layers. The accuracy of a deep network will drop rapidly when it reaches the highest point. This is not an overfitting problem and error rate will increase when the network gets deeper. They finally proposed a solution which is called ResNet.

They proposed a residual learning structure (see **fig.2.13**) to address problem. A shortcut connection is set up between the input and output of a block of layers, which can be formulated as $H(x) = F(x) + x$. No extra parameter or computational complexity is introduced by this shortcut. Applying this structure to every two convolutional layers of a CNN, they obtained the solution network, ResNet.

Comparison was made when they implemented a ImageNet Classification task. **fig.2.14** shows, testing on ImageNet data with plain network and ResNet (each have 18-layer and 34-layer configuration), the top-1 error rate tends to increase for a plain

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

FIGURE 2.12: VGGNet Configurations: The only difference between configurations is the number of convolutional layers, which lead to the change of the CNN depth as shown in column A to E. conv stands for convolutional layers

network when it gets deeper while ResNet keep decreasing the error rate as it gets deeper.

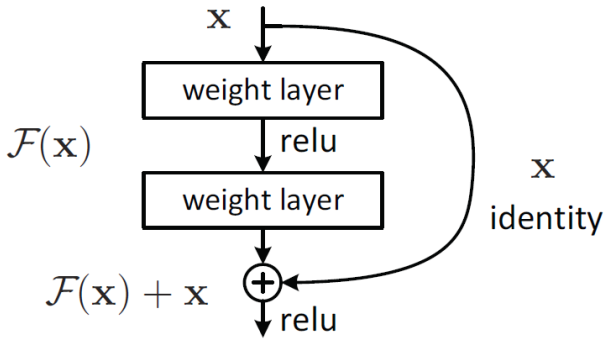


FIGURE 2.13: Residual Learing Framework

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

FIGURE 2.14: Top-1 error (%, with 10-corp testing) of plain network and Resnet

3 Adversarial Optimization of Deep Convolutional Networks

In this section, we will introduce the adversarial optimization and have an overview of the current state-of-the-art strategies for generating the adversarial examples. Finally, in this section, we will see some latest study of in the solution to countermeasure adversarial examples.

3.1 Adversarial Optimization

The famous result that deep networks can be broken easily [1] is an important motivation of this work. The idea is to start from an existing example and optimize for an example that will be classified to another category while being close to the original example. Namely, the following optimization problem is solved:

$$\begin{aligned} \min_{r} \quad & c\|r\|_1 + L(f_\theta(\mathbf{x}_0 + r, y)) \\ s.t. \quad & \mathbf{x}_0 + r \in [0,1]^d \end{aligned} \tag{3.1}$$

where $\mathbf{x}_0$ is a known example and y is an arbitrary category label, d is the input dimensionality. c is a parameter that can be tuned for trading off between closeness to the original example x_0 and the classification loss on the other category y. It has been shown, to the astound of many, that one can choose an r with very small norm while completely change the output of the algorithm (e.g. Fig. 1.1), this can be even done almost universally for almost all networks, datasets and all categories [1, 7]. This leads many people to question whether deep networks are really learning the "proper" rules for classifying those images.

3.2 An Overview of Related Works in Adversarial Example Area

In 2014, Szegedy et. al. discovered two properties of deep neural networks. They proposed that the semantic information in some layer is carried by the entire space of an activation instead of an individual unit. The projection to any direction in the space will make it indistinguishable, which means unit-level inspection methods are less useful. And that makes one choose a global, or network level, inspection method to explain how a model makes a decision on a category. This type of method can also helps to find the particular region in the input that encloses the information that greatly help the model to give a correct classification result. Then, under this method, they give an implicit argument that, normally, by adding to a given example

some small enough increment or decrements will not change the class to which it belongs to. But with an optimization proposed to find the small perturbation, the example will no longer be classified correctly. The examples that are added with this kind of perturbations were termed as "Adversarial Examples".

They conducted their experiment on multiple networks and on **MNIST** dataset. And they find that their "minimum distortions" can be generated using different network and can fool networks with different parameters, particularly different initialization, number of layers and ways of regularization, from which they are generated. These adversarial examples can also be misclassified even when the network are trained from a different set of data. Interestingly, all those perturbations are imperceptible by human eyes, see **fig.3.1**.

(a) (b)

FIGURE 3.1: Adversarial Examples: Normal images are on the left column, perturb images are on the right and the perturbation are in the middle.

(a) (b)

FIGURE 3.2: Adersarial Examples from QuocNet

In 2015, an explanation for the adversarial example generated by Szegedy in a previous year[7] was proposed by Goodfellow et. al.. Their explanation shows that the adversarial example can be produced leveraging the linear behavior of the deep network. The linear behavior can be seen as a linear module in the network and has a transfer function with a weights matrix W:

$$y = W^T x \tag{3.2}$$

An adversarial example is considered as $\tilde{x} = x+\eta$, where η is the perturbation optimized in [1]. When such adversarial example applied to a deep network, the output will be:

$$\tilde{y} = W^T x + W^T \eta \tag{3.3}$$

Because it is vector multiplication, increment $W^T\eta$ can be maximized by assigning η to be the vector with the same direction of the weights W and with all element value equals to 1. When solving a problem with large dimensional input, such as images, this increment term will grow dramatically. Some networks are designed to show most of it linear properties for the sake of easy training process. Hence, Goodfellow et. al. approximate this kind of network as a linear model. The perturbation η can be expressed as:

$$\eta = \epsilon sign(\bigtriangledown_x J(\theta, x, y)) \tag{3.4}$$

where $J(\theta, x, y)$ is the loss function to optimize in the training process of the network, θ is a set of parameters and ϵ is a small constant. Applying this algorithm for multiple networks, when $\epsilon = 0.25$, they obtain error rate 99.9% with 79.3% confidence for a shallow softmax classifier on MNIST dataset, 89.4% error rate with 97.6% for a maxout network. When $\epsilon = 0.1$, the convolutional maxout network will get 87.15% error rate with 96.6% probability on CIFAR-10 preprocessed by Krizhevsky et. al. in 2009.

Their result of perturbation gives a good generalization of the adversarial example. The intriguing phenomenon is that different models always classify one adversarial

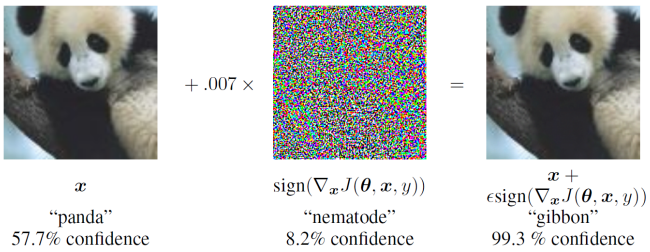


FIGURE 3.3: Adersarial Example Genterated by Fast Gradient Sign Method

example to the same class. They believe classifiers all resemble a linear classifier that can learn same set of parameters from different part of the dataset.

In 2016, Moosavi-Dezfooli et. al. proposed an efficient algorithm for computing the adversarial examples [27]. They defined an expression for the robustness ρ of a classier for a predicted label:

$$\rho(\hat{k}) = E_x \frac{\Delta(x; \hat{k})}{||x||_2} \tag{3.5}$$

where $\hat{k}$ is the label of interest, E_x is the expectation operator. And $\Delta(x; \hat{k})$, which means the robustness for label $\hat{k}$ at point x, has the definition:

$$\Delta(x; \hat{k}) = \min_r ||r||_2 \quad \textbf{s.t. } \hat{k}(x + r) \neq \hat{k}(x) \tag{3.6}$$

which means the minimum perturbation to make an example be misclassified by a classifier is the distance from the example to the hyperplane in the input space. The higher robustness intuitively means the data point is far from the hyper-plane for classification. By applying an iterative update rule, they iteratively push the point to another side of decision boundary. Thus make an adversarial example.

In 2015, Nguyen et. al. proposed another different way for fooling a deep network [28]. They produce images that is completely unrecognizable by human eyes (see **fig.3.4**), while a deep neural network can assign it to some category with an extremely high probability. Based on the traditional Evolutionary Algorithms (EAs),

they employed a new method called MAP-Elites that generate a population of such images that has a very high score and keep the one with best-so-far score, and then make a random permutation on the image. They used direct encoding and indirect encoding (two kind of way to represent a EA-adversarial image) to generate fooling images.

For images to match MNIST images (28×28 gray scale image), both direct and indirect way, with 200 generations or independent evolution, and get a median confidence of 99.99%. And for ImageNet images (256×256 images), they use 10 crop of size 227×227 in one images for network. Direct EAs for it only get median confidence 21.59% for $20,000$ generations. Only 45 classes can be successfully fooled with confidence of over 99%. Indirect EAs performs well after $5,000$ generations with confidence of 88.11%

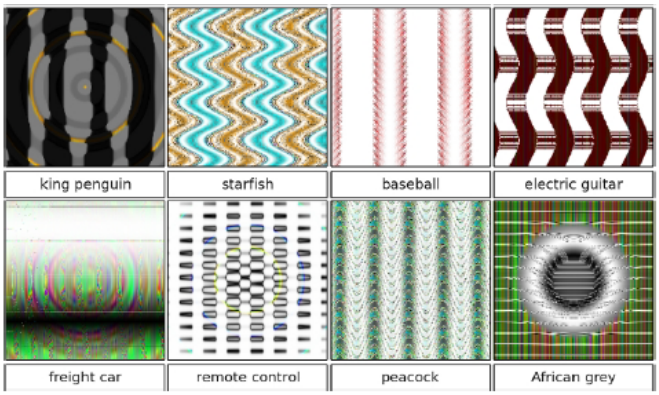


FIGURE 3.4: Evolved Images Unrecognizable to Humans

3.3 Overview of Countermeasures

Countermeasures also evolved in recent years [29, 30]. Goodfellow et. al. in [7], inspired by Szegedy et. al., proposed their adversarial training (meaning that

training on the adversarial example), which serves as a regularization part of the model. They believe that deep networks are able to represent functions that show less vulnerability to adversarial example, which is supported by the theorem of Universal Appropriator[31]. Their fast gradient sign method is low-overhead algorithm to generate adversarial example, which makes adversarial an efficient process. With adversarial training they obliged to reduce the error rate from 89.4% to 19.7%.

From (3.4), They managed to derive an modified objective function with second term as its regularization term:

$$\tilde{J}\theta, x, y = \alpha J(\theta, x, y) + (1 - \alpha) J(\theta, x + \epsilon sign(\nabla_x J(\theta, x, y))) \tag{3.7}$$

This regularization term produce the increase the ability of perturbation of the adversarial example, which in turn an efficient way. They applied this method to a maxout network with dropout regularization, they are able to obtain the error rate 0.94% (without adversarial training) to 0.84% (with adversarial training). After that, noticing that the training error cannot reach zero, they modified the network with more units on each layer (1600 units per layer instead of previous 240 per layer), they trained this network with early stop. The result dropped from 1.14% without adversarial training to on average 0.79%.

An 2016 work by Bendale et. al. introduced an new model layer[32], which is called OpenMax. They were inspired by the previous work about open set, which indicates that some of the data may belong to the categories that has not been learned by the CNN. In their work, they termed the activation output of the penultimate layer as Activation Vector (AV), denoted as $v_i(x)$, $i \in [1, N]$. Applying extreme value fitting to the mean of AVs of different category, they are able to obtain a hyper parameter ρ to revise the original AV and obtain an logical activation $v_0(x)$ for the openset. Similar to the $Softmax$, the $OpenMax$ is definde as:

$$P(y = j|x) \frac{e^{v_j(x)}}{\sum_{i=0}^{N} e^{v_i(x)}} \tag{3.8}$$

where $v_i(x)$, $i \in [0, N]$, is the output value of units in the activation layer, which represent $N+1$ confidence values for N categories and an additional openset category. They conducted their experiments on $80,000$ test images set($50,000$ validation data in ILSVRC2012 dataset, $15,000$ fooling images and $15,000$ unknown images) using pre-trained AlexNet and *OpenSet* did outperform $SoftMax$ in terms of F-measure, see **fig.3.5**. They also visualized the probabilities given by $OpenMax$ and regular

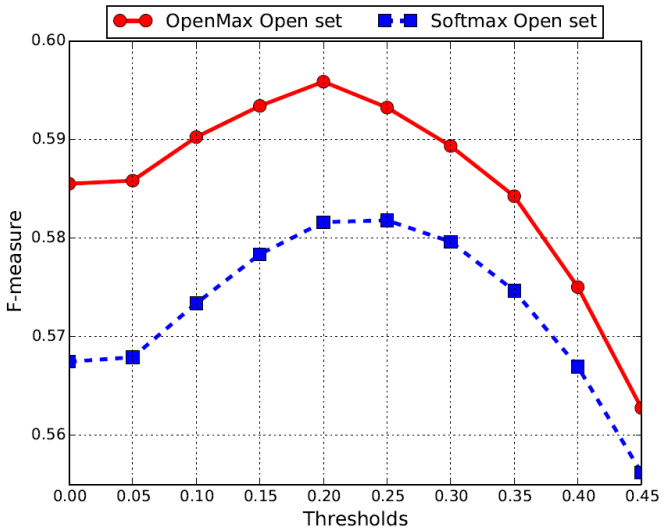


FIGURE 3.5: OpenMax outperforms Regular Method

$SoftMax$, which shows that $OpenMax$ will assign low confidence on unknown instances of images, see.

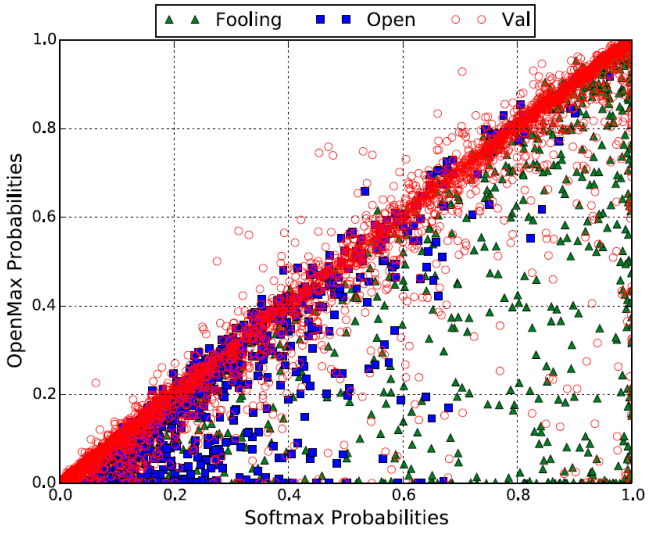


FIGURE 3.6: OpenMax outperforms Regular Method

4 Understanding the Trained Deep Classifier Under Adversarial Optimization

The term Adversarial Example [1] was firstly proposed by Szegedy et. al. in 2014 when they were analyzing the counter-intuitive properties of deep neural networks. The general property of this kind of examples is that it can mislead a well-trained network anomaly. Specifically, in the process of classification, it makes networks assign to it a category to which it does not belong.

4.1 Deep Networks Are Robust to Random Noises

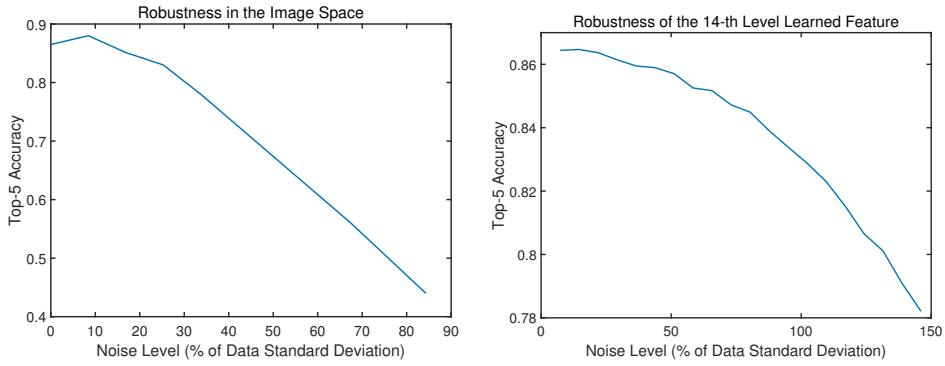
Despite astounding findings in **3.1**, deep learning is relatively robust to conventional perturbations on many different levels. We perform some experiments using a standard 16-layer VGG network [2] as mentioned previously.

In Our experiment, we added Gaussian perturbation to the input image. Two

experiments are conducted, the first add various levels of perturbation on the original image. The second adds perturbations on the 14-th layer (the first fully-connected layer with $4,096$-dimensional feature). This layer is chosen because it is sufficiently close to the final prediction while not at the final layer, and it avoids complications from the convolution layers (which we will get to in Section 4). One could think of the convolution layers as doing *feature extraction* and the fully-connected layers as doing *machine learning* in the traditional sense: given input x and predict an output y. Under this line of thought, the first fully-connected layer would contain the features for the learning.

As Fig. 6.8 shows, the prediction degrades as one adds stronger and stronger perturbation. However, deep networks can handle noise with standard deviation under 15 percent of data standard deviation without much trouble, which is significantly larger than the adversarial perturbation it can handle.

At the 14-th layer, deep learning is more resilient. In fact, it is extremely resilient that even if one enters half the noise w.r.t. to the signal, it can still perform admirably with only a 1% drop in classification accuracy. This basically shows that, although the previous papers on adversarial training might give the impression that deep learning is not robust, it is not the case. If one does not specifically aim at attacking it, it would be difficult to degrade a deep learning classifier, and the features it learned are even more robust. For a perturbation to break the network, it would have generated substantial variation on the learned feature (e.g. in the 14-th layer) to make the classification fail. We believe this is important to note, since then it is clear that the adversarial examples are finding very specific and non-random perturbations to corrupt the prediction. Especially, those perturbations have caused significant changes to the deep features (such as the ones at the 14-th layer) such that an otherwise extremely robust feature has been corrupted completely.



(a) (b)

FIGURE 4.1: Robustness of Deep Networks. In the image domain, it requires random noise of about 30% of the signal to make CNN start performing worse. b) In the feature domain, it requires a whopping amount4 of random noise to make CNN perform even a little worse.

4.2 Adversarial Behavior

As we have shown, deep networks degrade nicely in almost the entire feature space, thus cannot be broken easily from random vectors, and they are more robust in the very deep layers.

In order to gain a deeper understanding of the behavior of a deep network, we utilize spectral analysis. As a starting point, we perform principal component analysis (PCA) [33] at the 14-th layer of the network (the first fully-connected layer). The rationale behind using a principal component analysis is that each deep learning layer is a nonlinear activation function on a linear transformation, hence a lot of the learning process lies within the linear transformation, for which PCA is a standard tool to analyze.

A linear PCA is performed on the entire collection of $50,000$ images from the ImageNet validation set collected using the approach in (3.1), starting from random images in the collection. The result shows very interesting findings (Fig. 4.2) and

shed more light on the internal mechanics of those adversarial examples. We show the effects via a number of figures. In Fig. 4.2(a), we show the PCA projection onto the first two eigenvectors. This cannot separate normal and adversarial examples, as one could possibly imagine. The adversarial examples look exactly like an interpolation, rather than an extrapolation. However, it does seem that the adversarial examples reside mostly in the center while the normal examples occupy a bigger chunk of space.

Interestingly, as we move forward to the tail of the PCA projection space, the picture start to change significantly. In Fig. 4.2(b), we can see that there are a significant amount of adversarial examples that has extremely large values w.r.t. to the normal examples in the tail of the distribution. We chose to print the projection on the $3,547$-th and $3,844$-th eigenvector, but similar distributions can be found all over the place. As one can see, at such a far end on the tail, the projection is very similar to a Gaussian distribution. An explanation for that could be that under this "uninformative" weighted direction most of the weighted features are nearly independent w.r.t. each other, hence the distribution of their sum is similar to Gaussian, according to the central limit theorem[1]. However, although normal examples behave similarly to a Gaussian, some adversarial examples are having projections with a deviation as large as 5 or 10 times the standard deviation, which are extremely unlikely to occur under a Gaussian distribution.

[1]Note this is directly after the final convolution layer without a ReLU transformation, since ReLU destroys the negative part of the data distribution, the data no longer looks like a Gaussian after ReLU. However, some tail effects can be observed even in the distribution after ReLU.

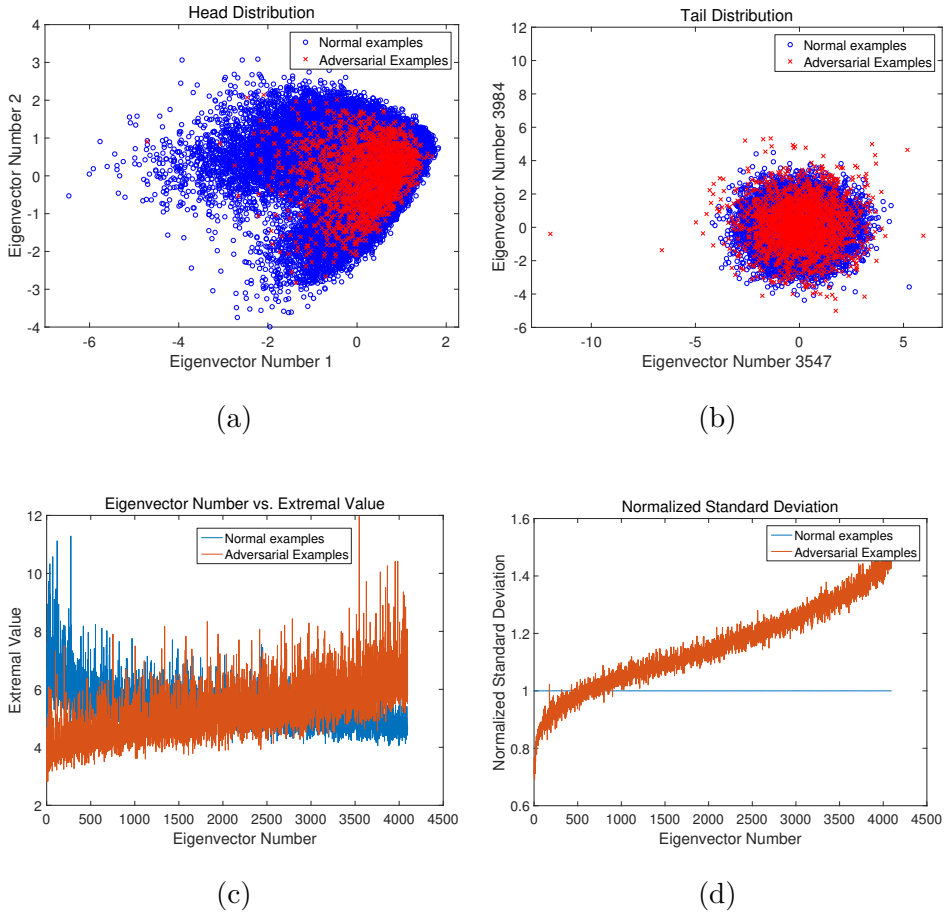


FIGURE 4.2: Blue indicates normal examples and red/orange indicate adversarial examples. (a) The projection of the data at layer 14 onto the 2 most prominent directions; Adversarial example cannot be identified from normal ones. (b) Projection of the same data to the $3,547$-th and $3,844$-th PCA projections, some adversarial examples are having significantly higher deviation to the mean; (c) The absolute value of the most extremal value in the projection to each eigenvector, normalized by the standard deviation of the projections on the normal eigenvectors; (d) The average normalized standard deviation of normal and adversarial examples on each projection.

Fig. 4.2(c) and Fig. 4.2(d) shows that there are two distinct phenomena:

- The extremal value and standard deviation on the projections onto the first $500 - 700$ eigenvectors are decidedly lower in the adversarial examples than the

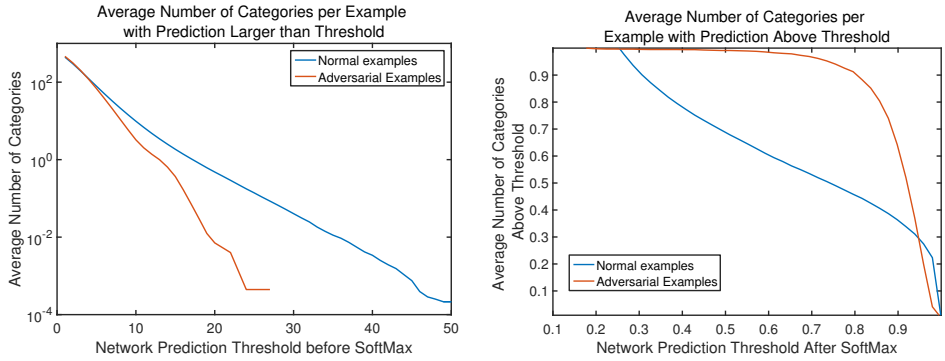
normal ones.

- The extremal value and standard deviation on the projections onto the last $1,000-1,500$ eigenvectors are decidedly higher in the adversarial examples than the normal ones.

It is interesting to reflect about the causes and consequences of those properties. One deciding property is that there is a strong regularization effect in adversarial examples on almost all the informative directions. Hence, the predictions in adversarial examples are *lower* than those of normal examples, rather than higher as it might seem in Fig. 1.1. In Fig. 4.3, we show the number of categories with a prediction higher than a threshold. The result shows strong regularization effects in adversarial examples: before the softmax transformation, normal examples have on average one category prediction that is more than 20, however adversarial examples have only 0.01 category predictions more than 20, meaning one in 100 examples with even one strong prediction. The reason that those adversarial examples appear more *confident* after softmax is because that the predictions on all the other categories are regularized *even more*. Hence the normalization component of softmax has decided that the single prediction, although much less strong, should be assigned a probability of more than 90%. This could be seen as an artifact of softmax that requires the normalization term, however, there is no good alternative that handles simultaneously multi-class classification. One could utilize one-against-all to compute per-class probabilities for each yes/no decision, however that is well-known to be difficult to calibrate among different one-against-all classifiers in order to generate a unique classification result.

The second property also offers important insight to the problem. We can hypothesize that some additional dependencies that the adversarial examples have utilized to lower the prediction values have made its way into the tail of the distribution, and produced large deviations from the mean which is only possible with high interdependency. While the tail is not immensely useful during classification, it does have some contributions. The main insight we can gain from this is that it is possible

to use the deviation of PCA projections to the mean to determine whether the data come from the normal distribution or the adversarial distribution. This provides a basis for us to devise confidence estimates of deep learning that can reject predicting on those adversarial examples.



(a) (b)

FIGURE 4.3: Average number of categories per example with predictions higher than a threshold. (a) Before softmax; (b) After softmax. As one can see, in normal examples, there are on average about 1 category with a prediction score of more than 20 (before softmax), while with adversarial ones, only 1% examples have a category with a prediction score more than 20. However, since predictions on almost all categories have dropped, after softmax adversarial examples have much higher likelihood on a particular category.

5 Identifying and Countering Adversarial Examples

5.1 Self-Aware Learning with an Abstain Option

Previous work on adversarial examples has mainly focused on re-training the network to gain better separation on adversarial examples. In this work, based on the previous findings, we instead advocate an approach that would first *identify* adversarial examples. This would help us to *avoid* classifying those examples, which do not come from the input distribution we are trained on and are likely adversarial.

This puts us into the framework of self-aware learning [10] where the learning algorithm has an abstain option of saying "I don't know", instead of making actual prediction on the example. We define a framework that is slightly different than [10], avoiding the KWIK requirement of never making a mistake.

We assume that the training input is drawn i.i.d. from a distribution $P(\mathbf{x}, y)$, where $\mathbf{x}$ is the input and y is the output. Assume that the testing input is drawn from a mixture distribution between $P(\mathbf{x}, y)$ and $Q(\mathbf{x}, y)$:

$$P_m = \Omega P(\mathbf{x}, y) + (1 - \Omega) Q(\mathbf{x}, y) \tag{5.1}$$

, where $\Omega \in \{0, 1\}$ is an unknown mixture weight, and $Q(\mathbf{x}, y)$ is an adversarial distribution. Assume that we have a classifier that includes a function $f(\mathbf{x})$, and a boolean strategy a_i between `predict` and `abstain` that can be chosen for each individual $\mathbf{x}_i$. Assume that the expected error from our classifier on the adversarial distribution is e_q (which could be assumed, if no other prior is present, as the random guessing error of $\frac{C-1}{C}$ for a C-class classification problem). Further assume that abstaining always incur a fixed cost e_a. As long as $e_a < e_q$, abstaining would be better than predicting on the example drawn from the adversarial distribution, however, e_a should be set sufficiently large so that it still makes sense to predict if the classifier is confident about it.

For each testing input, the testing of the self-aware classifier is then trying to

optimize $\min_a E_{P_m} L_a(x, y)$ where

$$L_a(x_i, y_i) = \begin{cases} P(y_i \neq f(x_i)), & \text{if } a_i = \texttt{predict}, (x_i, y_i) \sim P(x, y) \\ e_q & \text{if } a_i = \texttt{predict}, (x_i, y_i) \sim Q(x, y) \\ e_a & \text{if } a_i = \texttt{abstain} \end{cases} \tag{5.2}$$

hence the classifier needs to select between making a prediction using its function $f(x)$ and risk paying e_q versus abstaining. It is then not difficult to see that the optimal strategy is:

$$\begin{aligned} &a_i = \texttt{predict}, \quad \text{if } P(\Omega = 1|x_i)P(y_i \neq f(x_i)) + P(\Omega = 0|x_i)e_q < e_a \\ &a_i = \texttt{abstain}, \qquad\qquad\qquad \text{otherwise} \end{aligned} \tag{5.3}$$

which leaves the question of estimating $P(\Omega = 1|x_i)$, whether the image x_i comes from the training distribution or the adversarial distribution. Our main contribution is an algorithm defined in the next subsection to solve this subproblem of estimating $P(\Omega = 1|x_i)$.

5.2 Estimating whether a Testing Image Comes from the Training Distribution

We seek evidences that can separate adversarial examples from normal ones. A simple idea would be to proceed to the fully-connected layer and look for extremal value patterns on the tail, as in Fig. 4.2(b), where an SVM classifier on the normalized standard deviation can correctly classify more than 99% of the adversarial examples. However, since the tail distribution corresponds to only very small differences in the input, an adversarial optimization algorithm can defend against this by explicitly minimizing the projection on the tail PCA components, without heavily affecting the prediction results.

Instead we turn to early convolutional layers, where the tail distribution is not obvious and easy to combat against, and that would achieve fastest prediction

between normal and adversarial examples. Convolutional layers are very interesting to consider, since they work with many small correlated images and generate many different outputs, each for a small window in the input. The output of different locations in an image under one convolutional filter can be thought of as a sample from a distribution on the responses w.r.t. a convolutional filter output. For example, if the filter is detecting a left-right boundary, the distribution would capture the statistics of how likely such a left-right boundary would be in an image. Similar ideas have been used in computer vision ever since the SIFT BoW model [34]), however, we are hoping to compute statistics out of it to decide whether the image looks like a normal distribution or an adversarial one, rather than extracting meaningful features for classification.

Suppose the output at a convolutional layer m is an $W \times H \times K$ tensor, where W and H represent the width and height of the tensor, and K represents the number of convolutional filters. Before max-pooling, W and H would be the same as the image size, and after max-pooling it will be smaller. Such a tensor can be considered as a K-channel image where each pixel has a K-dimensional feature. We consider the feature on every pixel to be a random vector drawn from the distribution D_m of convolutional pixel outputs, a K-dimensional distribution.

The statistics extraction in the training phase can be summarized as **algorithm.1**. (**1** represents the all-one vector). And the testing phase works as **algorithm.2**

The simple algorithm collects the variances on each PCA component in each image and input those into an SVM classifier. As one can see from the empirical results in Fig. 4.2(c) and Fig. 4.2(d), the footprint of the adversarial examples are very eminent in those variances on principle directions, hence we expect the SVM classifier to be able to capture those footprints and achieve good classification results.

Algorithm 1 Single Layer Statistics Extraction

1: Draw examples from the D_m of normal images in a training set to form an example matrix $\mathbf{Z}$.

2: Compute the mean $\mathbf{m}$ and PCA projection matrix $\mathbf{W}$ of $\mathbf{Z}$.

3: Compute the standard deviation $\mathbf{s}$ on each dimension in the PCA projection $\mathbf{W}^\top(\mathbf{Z} - \mathbf{m}\mathbf{1}^\top)$.

4: For each image I, draw many examples (many pixels) from this image $\mathbf{Z}_I$, project them using PCA: $\mathbf{W}^\top(\mathbf{Z}_I - \mathbf{m}\mathbf{1}^\top)$, and normalize them by dividing **standarddeviation** on each respective dimension.

5: Collect the statistic for each image as $\mathbf{x}_I = \mathbb{E}(\|\mathbf{Z}_I\|_1)$, where L_1 norm is the vector L_1 norm on each column, it is used just for convenience reasons, in principle any statistic can be used.

6: Collect 25, 50 and 75 percentile point from the filter maps on each convolutional layer. The resulting vector would also be a K dimensional feature vector for each percentile.

7: Collect minimum and maximum extremal value from feature maps on each of the convolutional layers.

8: Train an linear logistic regression (or SVM) classifier on a training set with normal and adversarial examples obtained from step 4 to 7. Label normal examples as -1 (negative example) and adversarial examples as $+1$ (positive example).

Algorithm 2 GetStatistics

1: **for** each *layer* **do**

2: For each image I, draw many examples (many pixels) from this image $\mathbf{Z}_I$, project them using PCA: $\mathbf{W}^\top(\mathbf{Z}_I - \mathbf{m}\mathbf{1}^\top)$, and normalize them by dividing $\mathbf{s}$ on each respective dimension.

3: Collect the statistic for each image as $\mathbf{x}_I = \|\mathbf{Z}_I\|_1$, where L_1 norm is the vector L_1 norm on each column.

4: Collect 25, 50 and 75 percentile point from the filter maps on each convolutional layer. The resulting vector would also be a K dimensional feature vector for each percentile.

5: Collect minimum and maximum extremal value from feature maps on each of the convolutional layers.

6: Use the trained linear logistic regression (or SVM) classifier to predict the likelihood whether the example belongs to the normal or the adversarial distribution.

7: **end for**

5.3 A Cascade of classfiers

The ROC curve from a single layer shows that the detection task with respect to a single convolutional layer will not give a good result, as shown in **fig.5.1**. On

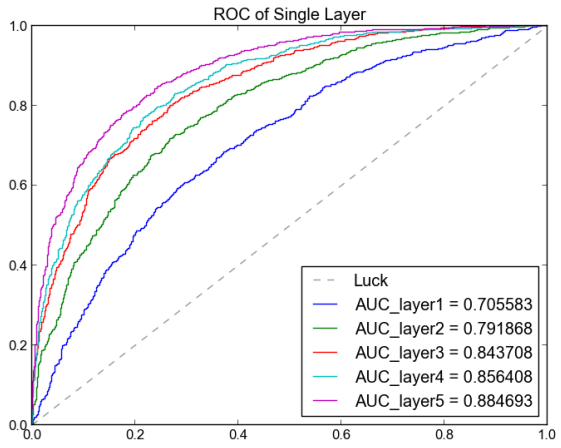


FIGURE 5.1: RCO for each of the convolutional layers in AlexNet

each convolutional layer, the value of area under ROC is no more than 0.9. So we cannot trust the detection result from only one single convolutional layer.

Viova et. al. in their precious work proposed a strategy to detect patches of pixels that represent human faces[35]. The cascade classifier is designed based on *AdaBoost* and is composed by a sequence of base classifiers. Each classifier is responsible for operating on data from the corresponding convolutional layer.

A cascade classifier is usually used in a binary classification setting, see **fig.5.2**, such as detecting the existence of a positive example. SC_N represents the classifiers corresponding to each convolutional layer. X is the input of the cascade classifier. In general, in the training process in cascade classifier, a detected positive example will trigger the classifier on the next stage continue training on it. And, as soon

as one example is assigned to be negative, the cascade classifier system will discard it immediately. This training behavior conforms our reality that, in the real world, normal data are far more than defected data so that images can be considered to be normal most of the time.

Instead of using this method directly, we constructed a modified version of cascade classifier, considering working on single layer data will result in a poor result. The base classifiers will not solely consider statistics from their own stage, instead, after one stage of training, the remaining positive examples will be concatenated to the corresponding features on the next stage. The training task is harder at the latter stages of the cascade.

The cascade classifier has pretty good property of their overall recall and false positive rate. The recall and false positive rate are equal to the, respectively, production of single layer recalls and false positive rate respectively, and they can be shown as **fig.5.2**.

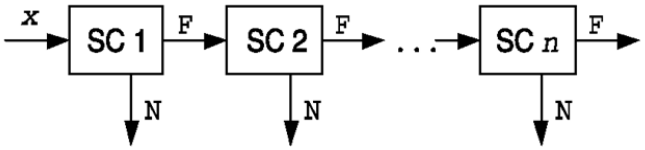


FIGURE 5.2: RCO for each of the convolutional layers in AlexNet

The operations that represented by **fig.5.2** can also be summarized as **algorithm.3**.

Algorithm 3 Training Process of a cascade of Classifier

1: $N_{pool} \leftarrow$ Normal Example pool
2: $N_{train} \leftarrow \varnothing$
3: $P_{train} \leftarrow$ Training Perturbed Dataset
4: $P_{test} \leftarrow$ Testing Perturbed dataset
5: $L \leftarrow$ Number Of Convolutional layer
6: $s \leftarrow sizeof(P_{train})$
7: **while** current layer $\leq L$ **Or** $N_{pool} \neq \varnothing$ **do**
8: $svm \leftarrow$ LinearSVM.instantiate()
9: $N_{train} \leftarrow$ draw sebset of size s from N_{pool}
10: $T \leftarrow P_{train} \cup N_{train}$
11: $Data_{train} \leftarrow$ statistics of interest corresponding to images in T calculated by **algorithm.1**
12: svm.train($Data_{train}$)
13: $Data_{pool} \leftarrow$ statistics correspond to N_{pool}
14: $N_{negative}, N_{positive} \leftarrow svm$.predict($N_{pool}$)
15: Eliminate $N_{negative}$ from N_{pool}
16: **end while**

The overall false positive rate of a K stage cascade classifier can be represented as:

$$F = \prod_{i=1}^{K} f_i \tag{5.4}$$

where f_i is the false positive rate at each layer. And similarly the true positive rate can be represented in the same form:

$$T = \prod_{i=1}^{K} t_i \tag{5.5}$$

where t_i is the true positive rate at each stage. Given the goal of the true positive rate (recall) of the classification problem, we can maintain a pretty high true positive rate while having the false positive rate attenuate layer after layer in the process of

training. This can be done by selecting the classification threshold which corresponds to a relatively high true positive rate. For example, in AlexNet, the when we choose the threshold that yields 0.97 true positive rate, the false positive rate will range from 0.85 to 0.35, so the overall true possitive rate 0.86 and false positive rate 0.05. This result gives a pretty good ROC curve.

6 Experiments

6.1 Data Preprocessing

We have two source of data source for defected data, data generated using L-BFGS by [1] and data generated by EA by [28]. LBFGS-adversarial dataset are generated from For ILSVRC2012 validation dataset. In total, $4,400$ L-BFGS data were extracted from this dataset. For the purpose of contrast, we also generated $5,000$ EA-adversarial images using the algorithm in [28].

In the data preparation process, we extracted statistics of interest, mentioned in **section 5.2**, from feature maps in the lower 9 convolutional layers for both normal example and LBFGS-adversarials in VGG-16 convolutional neural networks, and in all 5 convolutional layers for normal, LBFGS-adversarials and EA-adversarials respectively in AlexNet.

All input images had been re-shaped into $224 \times 224 \times 3$ dimensions for VGG-16 model and re-shaped into $227 \times 227 \times 3$ dimensions. And they were normalized before input into the networks for evaluation.

6.2 Experiment Settings

We use MatConvNet from VLFeat to capture the feature maps of interest for all normal dataset, LLBFGS-adversarials and EA-adversarials. And we selected to use pre-trained model **imagenet-vgg-verydeep-16** as our VGG-16 CNN model and use **imagenet-caffe-alex** as our AlexNet model.

We selected **LinearSVC** from **scikit-learn** toolkit in the python2.7 as our base classifier in our cascade classifier.

6.3 Experiment for LBFGS-Adversarials Detection

Our classifier has been run on multiple convolutional layers in the network. Experiments show that we are very capable of identifying LBFGS-adversarials. We show that, although in each single layer base classifier does not performs perfectly with an accuracy no more than 80% and poor ROC curve, the overall performance of a cascade classifier composed with classifiers from each layer can reach as high as over 90%, which showcases the effectiveness of the spectral analysis and the idea of using statistics on the pixel distributions from each image.

6.3.1 AlexNet Model Experiment on LBFGS-adversarials

We performed our design of cascade classifier on the data generated from ImageNet ILSVRC2012 validation images using AlexNet network. In **table.6.1**, we show the single layer performance when experimenting with AlexNet, and we can see single layer performance, as we expect, is not very high:

TABLE 6.1: Classification Result for Normal vs. LBFGS-adversarials

Network Layer	2nd	3rd	4th	5th	6th
Accuracy	57.5 ± 0.69	67.3 ± 0.73	70.9 ± 0.57	74.9 ± 0.87	78.95 ± 0.63

However, when we configured all layers of base classifier together, the performance increased greatly. The base classifier is selected to be SVM, specifically, scikit-learn LinearSVC. Tuning on the parameter C and true positive rate (recall), we found it best performs with $C = 0.005$ and with the threshold corresponding with 0.97 true positive rate. The average accuracy reaches 83.337% over 20-fold cross-validation. To compare the performance with another detection algorithm, we also applied the OpenMax method mentioned in **section.3.3**. To learn the Weibull distribution, the

EVT is applied on the same set of training dataset. The ROC only get 0.81 AUC from the testing data on which our detector have just performed. And its accuracy is poor, see **fig.6.1**.

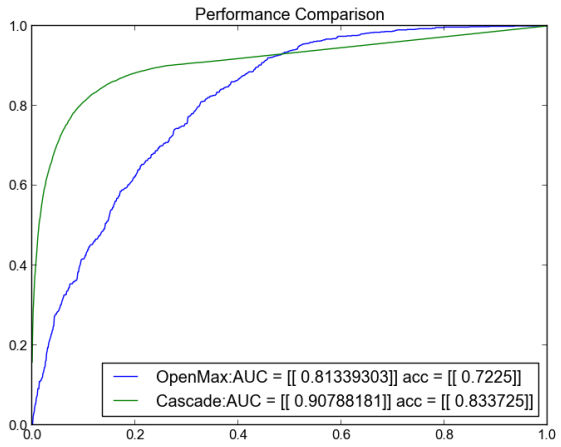


FIGURE 6.1: Comparison Between OpenMax detection Methods and Cascade Classifier: The blue curve represents the performace of OpenMax Method, and green curve represents the perfornace for Cascade Classifier.

6.3.2 VGG-16 Model Experiment on LBFGS-Adversarials

For comparison, we also performed this same classification task experiment on the data generated from the same dataset as in **section6.3.1** but using VGG-16 network. The single layer performance is shown in **table.6.2**

TABLE 6.2: Classification Result for Normal vs. LBFGS-Adersarials

Network Layer	2nd	3rd	4th	5th	6th
Accuracy	72.1 ± 0.69	84.1 ± 0.73	80.3 ± 0.57	81.4 ± 0.87	74.3 ± 0.63
Network Layer	7th	8th	9th	10th	
	73.9 ± 0.61	74.2 ± 0.65	71.2 ± 0.72	74.3 ± 0.79	

With the same set of base classifier, with parameter C = 0.005 and with classification threshold corresponding to 0.98, the accuracy of the classifier reaches on average 90.665% over 20-fold cross-validation. The accuracy is higher than the result from AlexNet, see **fig.6.2**.

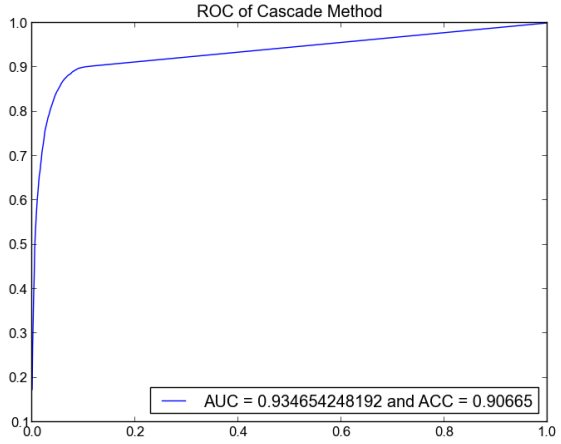


FIGURE 6.2: Overall ROC Performance Curve of Cascade Classifier Trained on VGG-16 Network

x

6.4 Testing Cascade algorithm on EA-Adversarials

To test the generalization property of our detector. We performed 20-fold corss-validation, with the same set of parameters as **section.6.3.1**, on data generated from EA-adversarials in [28] using AlexNet. The result shows that EA-adversarials are much more detectable than LBFGS-adversarials by our detector.

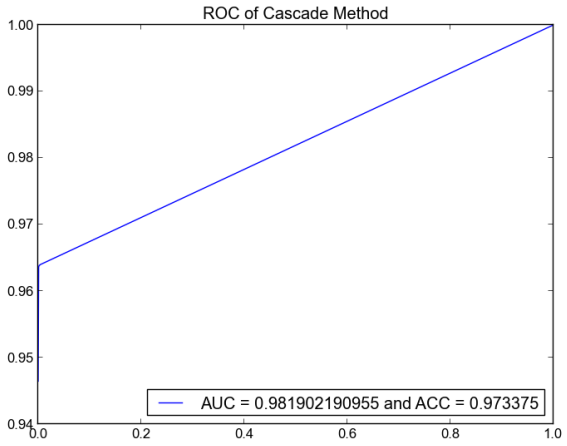


FIGURE 6.3: Overall ROC of data generated from EA-adversarials dataset on AlexNet

Single stage classifier performance when working on detection of EA-adversarials is also much more higher than that when working on detecting LBFGS-adversarials, see **table.6.3**. We only need the data generated from the first three convolutional layer to reach the overall 97.34% detection rate.

TABLE 6.3: Classification Result for Normal vs. EA-Adversarials

Network Layer	2nd	3rd	4th
Classification Accuracy	93.45 ± 0.69	98.3 ± 0.73	97.9 ± 0.57

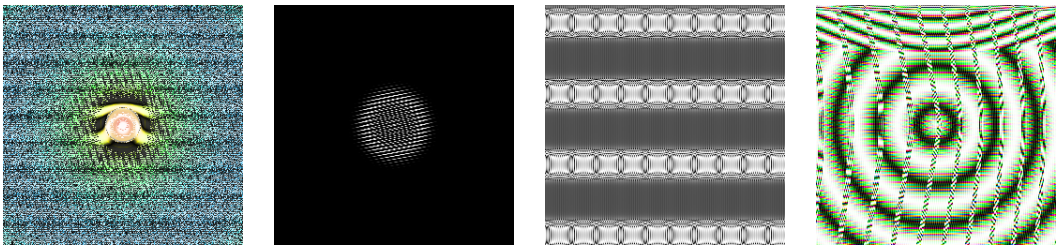
6.4.1 Images Classified Correctly and Incorrectly

(a) (b) (c) (d)

FIGURE 6.4: Some of Misclassification on L-BFGS images by Our Classifier. (a) and (b) are from normal dataset. (c) and (d) are from LBFGS-Adversarial dataset, which is misclassified to category n02408429(water buffalo) and n01518878(ostrich, Struthio camelus).

(a) (b) (c) (d)

FIGURE 6.5: Some of Correctly Classified on L-BFGS images by Our Classifier. (a) and (b) are from normal dataset. (c) and (d) are from LBFGS-Adversarial dataset, which is misclassified to category n04209133(shower cap) and n02328150(Angora).



(a) (b) (c) (d)

FIGURE 6.6: Some of Misclassfied EA images by Our Classifier. From left to right, they are misclssified to category n03220513 (dome), n01749939 (green mamba), n04118776 (rule, ruler) and n03935335 piggy (bank, penny bank)

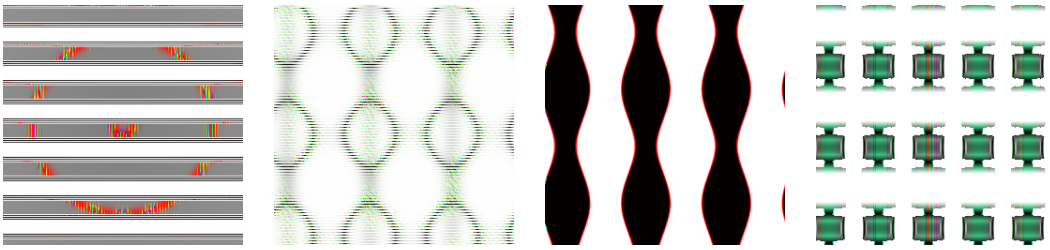
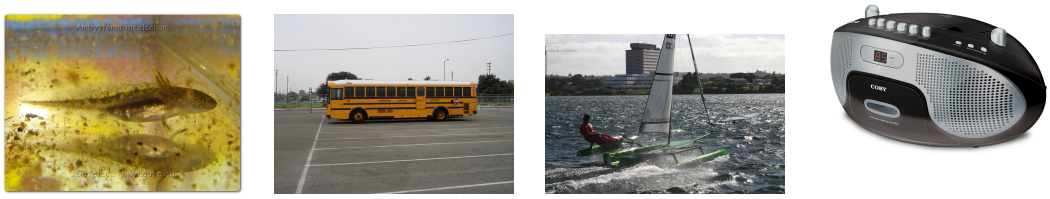


FIGURE 6.7: Some of Correctly Classified EA images by Our Classifier. From left to right they are misclassified to n06874185 (traffic light, traffic signal, stoplight), n03443371 (goblet), n04522168 (vase) and n03742115 (medicine chest, medicine cabinet)



(a) (b) (c) (d)

FIGURE 6.8: Images Miscalssified by OpenSet Method but Correctly Classified by Our Classifier. (c) and (d) are from LBFGS-Adversarial dataset, which is misclassified to category n02133161(American black bear) and n02328150(Agona).

6.5 Discussion

6.5.1 Performance And Depth of a CNN

We compared the over all results from **section.6.3.1** with **section.6.3.2**. The ROC curve of VGG-16 outperforms that of AlexNet experiment. The difference is that AlexNet has fewer number of convolutional layers. In the process of training and test, every time when we move forward to the next level, we will concatenate the feature from the previous layer to the next. This means, having more layers would help to gain more information for detection task. From **table.6.1**, **table.6.2** and **table.6.3**, we can see, as we move to the higher level of layer, the performance is in an increasing trend.

6.5.2 High Detectability of EA-adversarials

In **section.6.4**, the experiment result shows EA-adversarials are highly detectable by our detector. To gain more insight into this result, I made a few comparisons between the statistics of interest extracted from normal images, perturbed images and EA-adversarials.

We visualized the mean of the statistics that used for the detection task from the

first layer of the AlexNet on all its dimensions. As we can see in **fig.6.9**, from left part to the right part of the curve, the difference of the mean value of the PCA projection on layer 1 in AlexNet extracted from EA-adversarials and that of the normal ones is increasing dramatically, while, compared to the EA-adversarials, the statistics from LBFGS-adversarial have much less difference from the normal data and the difference does not change very much as we see from left to right.

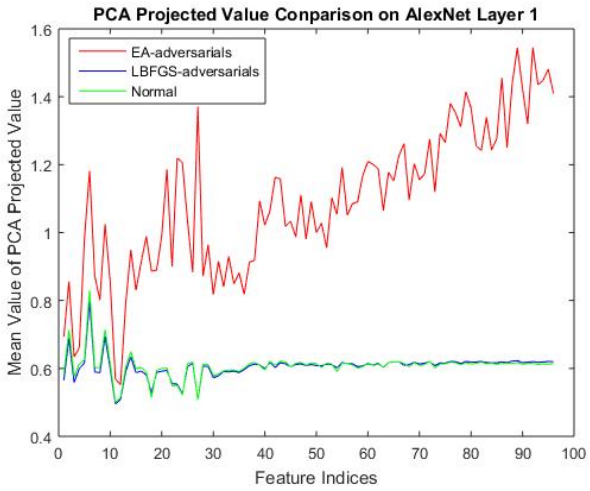


FIGURE 6.9: PCA Projection Comparison

Also, from **fig.6.10** and from **fig.6.11**, extremal statistics (minimum value and maximum value in each feature map) from LBFGS optimization result gain more difference from normal data. LBFGS-adversarials have smaller maximum values and greater minimum values. This may implies that the effect of the LBFGS optimization is to push the original data more compacted together. However, we can also see statistics drawn from EA-adversarials have more severe extent of fluctuation, which is also the reason why EA-adversarials are more distinguishable.

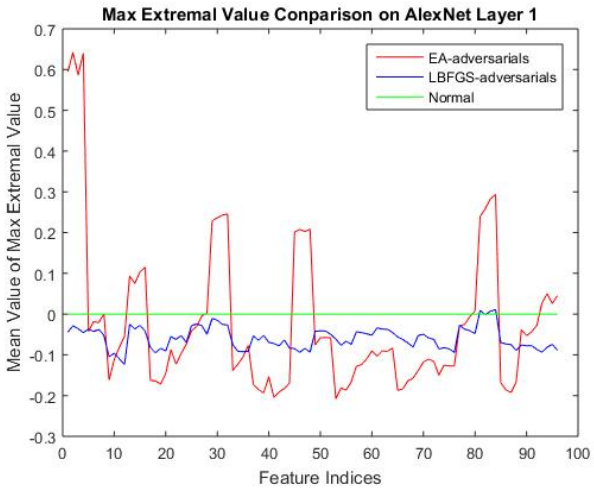


FIGURE 6.10: Maximum Feature Map Extremal Value Comparison

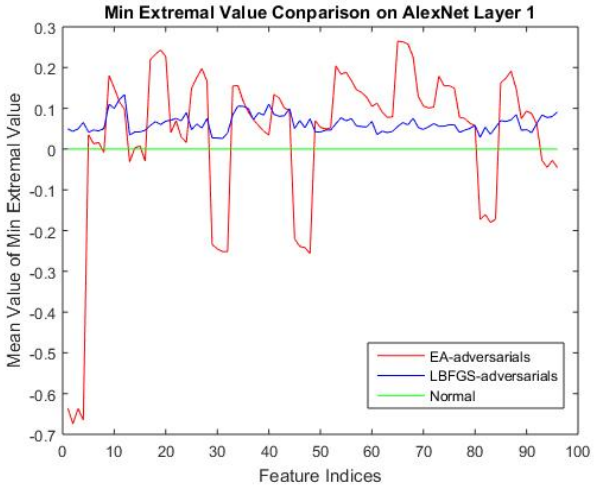


FIGURE 6.11: Minimum Feature Map Extremal Value Comparison

From **fig.6.12**, **fig.6.13** and **fig.6.14**, we still see the EA-adversarials evidently differ from normal images. The EA-adversarial statistics fluctuate much more severely than LBFGS-adversarial data from the normal data.

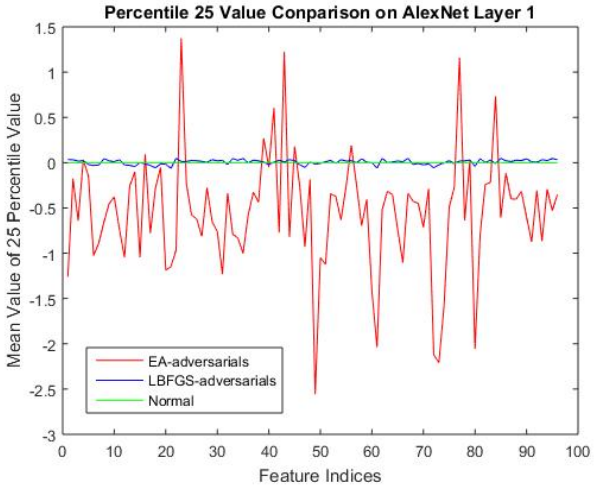


FIGURE 6.12: Percentile 25 Value Comparison

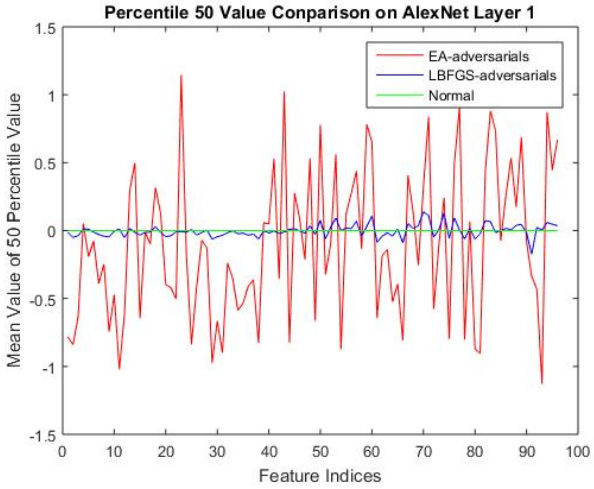


FIGURE 6.13: Percentile 50 Value Comparison

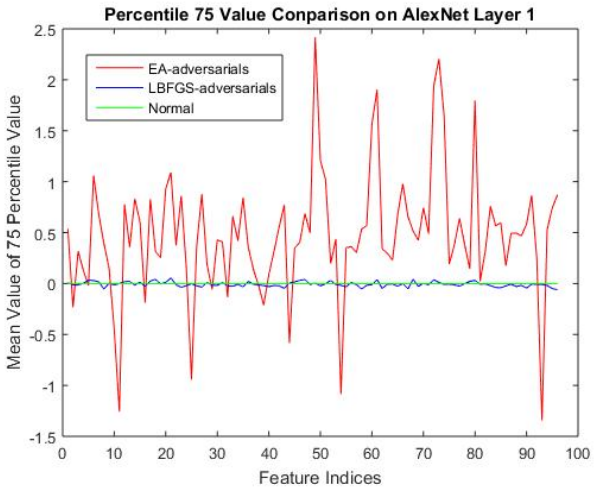


FIGURE 6.14: Percentile 75 Value Comparison

6.6 Image Recovery

Insights from [7] indicates that the adversarial mechanism is very specifically attacking vulnerable gradients starting from the first convolutional layer. Insights from the previous experiment also suggests that the adversarial effect could be very obvious even after the first convolutions. Therefore a natural idea would be to destroy the adversarial effects in the first convolutional layer to try to recover the original image. We try a very simple approach: applying an small (e.g. 3×3) average filter on the adversarial image before using CNN to classify it. The positive and negative adverse gradients will average out in this approach, and making the masked features from the normal images to show up a bit. In Table 6.4 we show such recovery results: after using a 3×3 average filter on identified adversarial examples, the classification accuracy grows from almost 0% to 73.0%, showcasing the effectiveness of this simple average filter.

TABLE 6.4: Recovery Results. Simply using a 3×3 average filter we can recover a large proportion of adversarial examples after detecting them using the algorithm described previously. More complex cancellation approaches such as foveation in [5] that utilizes cropping can achieve better results.

Approach	Top-5 Classification Accuracy on Recovered Images
Original Image (Non-corrupted)	86.5%
3×3 Average Filter	73.0%
5×5 Average Filter	68.0%
Foveation (Object Crop MP) [5]	82.6%

Those results show that we can both detect and recover from adversarial examples with very high accuracy. However, the authors believe that the main issues with the current deep convolutional networks is that they are too locally focused. As one can see, some adversarial examples that can be cancelled by a simple 3×3 average filter could corrupt the output of the deep convolutional network. For human with a large receptive field, they will not even care about what happens within a 3×3 area. Therefore, the authors believe that future deep learning approaches should focus on enlarging the receptive field in order to reduce the chance of being fooled by adversarial examples. Another potential direction is to research classification approaches that do not require a softmax-type normalization, in order to avoid regularizing attacks such as the ones used in the adversarial optimization in (3.1).

The reason for the behavior of EA images should be that they are not close to any of the training normal training examples in the input space. From the comparison, we can also interpret that the main contribution for the detection may lie in the exremal statistics when we try to detect L-BFGS optimization result.

7 Conclusion and Future Works

7.1 Conclusion

This paper proposes to deal with adversarial examples in deep learning using an approach that detects the adversarials. This is achieved by making empirical observations on the spectral properties of normal and adversarial examples and designing relevant algorithms that examine spectral statistics to identify normal and adversarial examples.

From the fact we see in the experiment in previous section, we saw that, for adversarial example detection, the performance detector we trained is correlated with the number of convolutional layers we used. We extract those statistics of interest from 9 layers of feature map in the VGG-16, and from 5 layers of feature maps in AlexNet. The classifier will discard the example points and stop working on them as soon as they are considered to be negative (statistics generated from normal images). Remaining examples will still be under tested until it reaches the end stage of the classifier or detected to be negative. The high recall value we selected for the base classifier brings the high false positive rate. This means even though the an example are detected to be positive, more stages of tests are needed to eliminate the false positive examples.

Another insight we have is that fooling images are much more detectable than adversarial examples in **section.6.4**. The reason is adversarial images is generated from normal images by adding an non-saturating increment to the normal ones. Normal inputs and adversarial inputs are very close by the distance in the input space. As for Fooling examples, they are generated to be with no semantic meaning for human eyes and they are not geometrically close to any examples in the training example distribution. Namely, the probability for fooling examples of being an outlier is high.

The experiments in **section.6** shows that, after we successfully train a cascade of classifiers, we can detect the incoming adversarial examples on-fly. the empirical

statistics of interest will be extracted during the time each feature map is evaluated. However, the performance is subject to the number of layers in a network. One of the possible future work direction is to design a classifier architecture that performs independent with the number of layers.

Bibliography

1. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)

2. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

3. Noh, H., Hong, S., Han, B.: Learning deconvolution network for semantic segmentation. In: International Conference on Computer Vision (ICCV). (2015)

4. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems. (2012) 1097–1105

5. Luo, Y., Boix, X., Roig, G., Poggio, T.A., Zhao, Q.: Foveation-based mechanisms alleviate adversarial examples. arXiv preprint arXiv:1511.06292v3 (2016)

6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition. (2016)

7. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)

8. Fawzi, A., Frossard, P.: Manitest: Are classifiers really invariant? arXiv preprint arXiv:1507.06535 (2015)

9. Sabour, S., Cao, Y., Faghri, F., Fleet, D.J.: Adversarial manipulation of deep representations. arXiv preprint arXiv:1511.05122 (2015)

10. Li, L., Littman, M.L., Walsh, T.J., Strehl, A.L.: Knows what it knows: a framework for self-aware learning. Machine learning **82**(3) (2011) 399–443

11. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. (1998) 604–613

12. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer-Verlag, New York (2001)

13. Nguyen, A.M., Yosinski, J., Clune, J.: Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. CoRR **abs/1412.1897** (2014)

14. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10). (2010) 807–814

15. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. arXiv:1409.4842 (2014)

16. Zeiler, M., Fergus, R.: Stochastic pooling for regualization of deep convolutional neural networks. In: ICLR. (2013)

17. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: IEEE Conference on Computer Vision and Pattern Recognition. (2009)

18. Yang, J., Yu, K., Huang, T.: Supervised translation-invariant sparse coding. In: Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, IEEE (2010) 3517–3524

19. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)

20. Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G.: Recent advances in convolutional neural networks. arXiv preprint arXiv:1512.07108 (2015)

21. Hubel, D.H., Wiesel, T.N.: Receptive fields and functional architecture of monkey striate cortex. The Journal of physiology **195**(1) (1968) 215–243

22. Le Cun, B.B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Handwritten digit recognition with a back-propagation network. In: Advances in neural information processing systems, Citeseer (1990)

23. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11) (1998) 2278–2324

24. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: European Conference on Computer Vision, Springer (2014) 818–833

25. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229 (2013)

26. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. (2012) 1097–1105

27. Moosavi-Dezfooli, S., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. CoRR **abs/1511.04599** (2015)

28. Nguyen, A., Yosinski, J., Clune, J.: Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In: IEEE Conference on Computer Vision and Pattern Recognition. (2015)

29. Gu, S., Rigazio, L.: Towards deep neural network architectures robust to adversarial examples. arXiv preprint arXiv:1412.5068 (2014)

30. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. arXiv preprint arXiv:1511.04508 (2015)

31. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural networks **2**(5) (1989) 359–366

32. Bendale, A., Boult, T.: Towards open set deep networks. arXiv preprint arXiv:1511.06233 (2015)

33. Jolliffe, I.: Principle Component Analysis. Springer-Verlag (1986)

34. Leung, T., Malik, J.: Representing and recognizing the visual appearance of materials using three-dimensional textons. International Journal of Computer Vision **43**(1) (2001) 29–44

35. Viola, P., Jones, M.J.: Robust real-time face detection. International journal of computer vision **57**(2) (2004) 137–154