Understanding Deep Networks with Gradients

Henry Z. Lo, Wei Ding
Department of Computer Science
University of Massachusetts Boston
Boston, Massachusetts 02125–3393
Email: {henryzlo, ding}@cs.umb.edu

***Abstract*—Existing methods for understanding the inner workings of convolutional neural networks have relied on visualizations, which do not describe the connections between the layers and units of the network. We introduce the prediction gradient as a measure of a neuron's relevance to prediction. Using this quantity, we study a relatively small convolutional neural network and make three observations. First, there exists a small number of high prediction-gradient units, which upon removal, severely impact the ability of the network to classify correctly. Second, this performance loss generalizes spans multiple classes, and is not mirrored by removing low-gradient units. Third, the distributed representation of the neural network prevents performance from being impacted until a critical number of units are destroyed, the number depending highly on the prediction gradient of the units removed. These three observations validate the utility of the prediction gradient in identifying important units in a neural network. We finally use the prediction gradient in order to generate and study adversarial examples.**

I. Introduction

Convolutional neural networks use serially stacked projections followed by a linear classifier [11]. In addition to learning a feature vector for classification, the network generalize features of the input space into higher-level concepts which are useful across data sets and tasks [2]. However, despite the flexibiliy of these features, understanding *how* they are internally represented is significantly hindered by their distributed representation.

We show that prediction gradients can be used to rate unit importance in a neural network, supporting this notion with qualitative experiments [10]. Using this measure, we also derive some insights into how relevance to prediction performance is distributed among units in a neural network.

Existing attempts to understand CNNs have focused on visualizing units, layers, and classes; for example, in figure 1Convolutional outputs for three images. Left shows three faces. Right shows filter outputs for each face image. Rows represent layers. figure.caption.1. This approach reveals little insight into how the neural network encodes the image as a function of its units, and how these units relate to each other, both within and across layers.

We show that the prediction gradient can be used to understand what happens inside a CNN in two ways. First, we use it to visualize activation patterns in the net, and show that similar classes often induce similar unit patterns. Second, we demonstrate the method's effectiveness in identifying features in a network. By removing identified 'strong' units, we show that performance degrades much faster than 'weak' units.

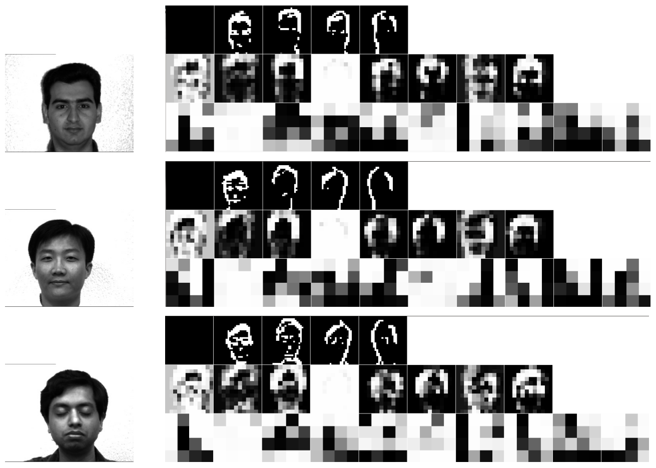


Fig. 1: Convolutional outputs for three images. Left shows three faces. Right shows filter outputs for each face image. Rows represent layers.

Finally, we demonstrate that the gradient can be used to generate images which look like one class, but are misclassified as another. Unlike previous work which only generated high-confidence misclassifications [15], we show that similar perturbed images can be made for almost any target class. We study the generalizability of these adversarial examples across classifiers, training sets, and across models, as in [15].

This last section of the paper investigates whether adversarial examples can truly be abused by an adversary. Models are already being proposed which are robust to, and indeed improve from adversarial examples [5], [6]. Regardless, given the demonstrated strengths of DNNs in computer vision, being able to use adversarials reliably may pose severe security vulnerabilities in domains such as image-based biometrics.

Our contributions are:

1) An efficient technique for measuring the importance of any unit in a deep neural network, using the *prediction gradient*.
2) Experiments showing the consistency and validity of the prediction gradient by relating the score of units to network performance.
3) Empirical evidence showing the robustness of distributed representations, and the existence of high-scoring units which are crucial for overall network performance.
4) A study on the feasability of adversarial image gen-

eration to fool neural networks, using the prediction gradient to generate such images.

II. Related Work

Existing work seeks to understand CNNs in terms of pixel space. To our knowledge this is the first paper to investigate how to understand network concepts in terms of its units.

Erhan et al. visualized CNNs by finding the pixels in input space which maximize the activation of a given unit [4]. The authors also propose a sampling method from deep belief networks.

Zeiler and Fergus visualized the concepts learned in each unit of a network using deconvolutional net [16]. These networks are optimized to reproduce the input given a unit's activation [17].

Simonyan et. al. visualized object classes by backpropagating class predictions back to the input pixel space. They also proposed a method for identifying the pixels of a given image relevant to its object class [14].

Donahue et. al. compared methods to reduce dimensions on the feature space at any given layer of a deep network, and proposed a method which selects image segments from pixel space to visualize the network [3].

Szegedy et. al. first showed that minute perturbations in images can cause high-confidence misclassifications [15]. Others have since shown that such misclassifications can occur for human-unrecognizable images, and that these misclassifications are a result of linearities in the activation functions of neural networks [12] [5].

Erhan and Zeiler's methods both map a single unit back to pixel space. Simonyan's methods visualize object classes, also in pixel space. Our proposed method seeks to understand object classes in terms of units in the CNN.

We propose a fast method for ranking and evaluating units in a neural net, which requires no additional optimization. By providing a single number for each unit, our method quantitatively measures each unit's contribution to a class prediction. In contrast, existing methods only visualize the contribution of a single unit by mapping back to pixel space, and several require nonconvex optimization. Furthermore, our measure is flexible, as it allows unit / feature importance to be calculated with respect to a given class input, prediction, or overall.

III. Prediction Gradient

The notation used in this paper is shown in Table INotation used in this paper. Superscripts and subscripts may be omitted when only considering one copy of a symbol. table.caption.2.

A. Prediction Gradient

We use the *prediction gradient* to quantify a unit's relevance for a given prediction:

$$\frac{\partial \mathbf{y}}{\partial o} = \left\langle \frac{\partial y_1}{\partial o}, \ldots, \frac{\partial y_{|T|}}{\partial o} \right\rangle \tag{1}$$

This vector measures how much the unit o contributes to the final output $\mathbf{y}$ for a given image. The ith element of the prediction gradient refers to the change in predicting class i due to varying o.

Symbol	Meaning
o_a^ℓ	Output of ath unit in ℓth layer
$\mathbf{y}^j$	Output vector of neural network for jth input
w_{ab}^ℓ	Weight from ath unit in $(\ell-1)$th layer to the bth unit in the ℓth layer
t^i	Identity of ith image
X_t	Set of all images sharing the label t
L	Loss function

TABLE I: Notation used in this paper. Superscripts and subscripts may be omitted when only considering one copy of a symbol.

In our analyses, we focus what makes the network predict correctly, so we investigate the element y_i where $i = t$ is the face label. The other elements of the prediction gradient provide insight into how unit o contributes to misclassifying the image. This may be of interest in understanding why the network makes certain mistakes.

B. Calculation

The prediction gradient can be calculated very quickly; most calculations are done as an intermediate step during backpropagation. Thus, obtaining all the prediction gradients of a dataset is relatively simple, and no slower than one epoch of training.

Let L be an loss function which depends on $\mathbf{y}$. Given one image, online backpropagation will update weight w_{ab}^ℓ using this equation:

$$w_{ab}^\ell = w_{ab}^\ell + \lambda \frac{\partial L}{\partial w_{ab}^\ell}$$

The weight w_{ab}^ℓ feeds into unit o_b^ℓ, and thus the error attributable to o_b^ℓ must be calculated before the error attributable to w_{ab}^ℓ can be determined. Mathematically, this becomes apparent after applying the chain rule:

$$\frac{\partial L}{\partial w_{ab}^\ell} = \frac{\partial L}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial o_b^\ell} \frac{\partial o_b^\ell}{\partial w_{ab}^\ell} \tag{2}$$

Thus, the prediction gradient is already calculated in the course of learning, and requires no additional effort to obtain.

C. Class Gradient

To quantify unit relevance to each class label, we use the *class gradient*, which is the average prediction gradient over all images of the same class t:

$$\frac{1}{|X_i|} \sum_{j \in X_i} \frac{\partial \mathbf{y}^j}{\partial o} = \frac{1}{|X_i|} \frac{\partial}{\partial o} \sum_{j \in X_i} \left\langle y_1^j, \ldots, y_{n_c}^j \right\rangle \tag{3}$$

Equation 3Class Gradientequation.3.3 shows that the average prediction gradient is equivalent to the gradient of the average prediction. In practice, we found that calculating the class gradient in this way is easier to parallelize.

For some datasets, averaging may be problematic if the elements of a class are not sufficiently smooth, that is, if different images of the same class have very different activation patterns. We show later that this is not the case for our data set.

The class gradient is useful in understanding how the neural network perceives all images of each class, rather than how it predicts one specific image.

D. Convolutional Gradient

In this paper, we focus on the class gradients of the convolutional layers in a CNN. Each layer can be thought of as a mode-4 tensor W^ℓ:

- Mode 1 corresponds to the different feature maps.
- Mode 2 corresponds to the different channels of the layer's inputs (e.g. one channel for each convolutional output in the low layer).
- Mode 3 and 4 correspond to the 2-d convolutional filter location in the image.

Sharing this weight tensor W^ℓ are many units: the amount is the number of valid input pixels times the number of feature maps. All units of a feature map arguably correspond to a single feature, so we quantify the importance of this entire set of units by aggregating all units in a map using the Frobenius norm. We call this the *convolutional gradient*. δ_a^ℓ is the ath convolutional gradient in the ℓth layer.

By aggregating, the convolutional gradient removes spatial information in a feature map. However, as we want to quantify the importance of a feature map, there is no way to avoid this. If we wanted to take into account spatial information, we could use prediction or class gradients instead.

The convolutional gradient is not a true gradient, and so is not directly comparable to the non-convolutional prediction gradients. However, it is a useful quantity to compare among feature maps.

IV. Experimental Setup

We use the gradients discussed to investigate an intentionally small network and data set to aid interpretability of results. Using a learned CNN, we do the following:

1) Identifying activation patterns relevant to correct class prediction, and show that these are relatively stable for different images in the same class.
2) Ranking and rating units in a neural network, to observe how the network understands face identities.
3) Validation of the measure of unit contribution by removing the strongest / weakest features, and observing the effect on performance.

Our goal is to demonstrate a measure of relevance of each unit to correct prediction. Our activation patterns, feature rankings, and knockout experiments support one use of the gradients, but there may be others. The purpose of the analyses is to demonstrate the utility of the class gradient, and hence we use a simple data set and neural network to maximize interpretability.

A. Model

We use a CNN similar to LeNet [9], but with the following parameters:

- 3 convolutional layers, with 5, 8, and 10 feature maps.
- One hidden layer with 20 units.
- Hyperbolic tangent activation functions.
- 5×5 convolutional filters for all convolutional layers.
- 2×2 max pooling for all convolutional layers.

Input images are resized to 60×60 pixels. The learning parameter is set to 0.1.

B. Data

To facilitate learning on our network, we use the Yale face database[1], which consists of face images of 15 identities in 11 different conditions [1]. 75% of the data set is used for training, and 25% for testing. The network was trained on 15 classes, one for each identity.

After training for 250 epochs, the network achieved an error rate of 4.88% on the data set. Weights for the network were frozen after this.

V. Gradient Analysis

We demonstrate the utility of the prediction gradient by example. Using on a CNN trained on a face recognition data set, we show how the prediction gradient can be used to understand the inner workings of a neural network, and to identify useful and non-useful features relevant to class labels.

A. Prediction Gradient Consistency

If prediction gradient within the same class are too different, then class gradients would be meaningless. We verified that this was not the case in Table IIPrediction gradients for three images each from three identities. The diagram in the network column the three convolutional layers of the neural network. Each circle is a feature map. Brighter circles indicate a higher prediction gradient for that feature map. Note the patterns in prediction gradients within each identity. table.caption.3; three different identities, under three different conditions. Units with the highest gradients stay quite similar across different faces of the same identity. In other words, each person has a distinct gradient profile of which units contribute most to detecting that face. This justifies the use of the class gradient.

B. Gradient-Based Feature Selection

The convolutional gradient can be used as a measure of a unit's effectiveness in producing a certain network outcome. Hence, it can be used rank the internal features of a neural network in terms of importance.

In table IIIFive identities with their images, class gradients across the whole network, and top 8 strongest and weakest features, as ranked by the class gradient. These five identities were used in the knockout experiments.table.caption.4, we use the convolutional gradient to identify the top 8 strongest and weakest features for each of five identities. We average these

[1]http://vision.ucsd.edu/content/yale-face-database

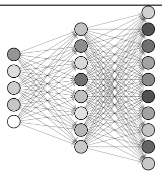
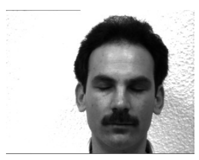
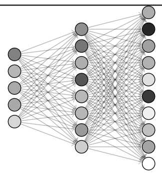
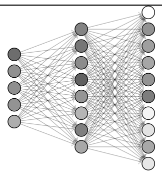
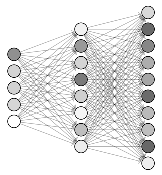
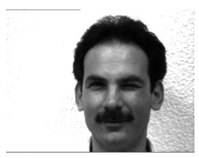
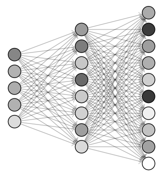
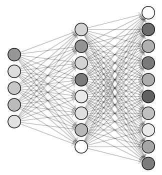
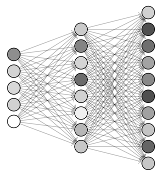
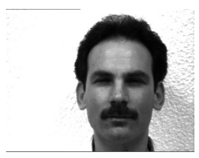
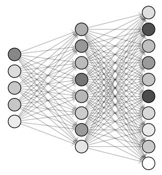
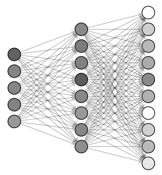
Identity 1		Identity 2		Identity 3	
Image	Network	Image	Network	Image	Network
					
					
					

TABLE II: Prediction gradients for three images each from three identities. The diagram in the network column the three convolutional layers of the neural network. Each circle is a feature map. Brighter circles indicate a higher prediction gradient for that feature map. Note the patterns in prediction gradients within each identity.

features across different images of the same identity. Note that as the gradients are different for each identity, each identity has a different set of strongest and weakest features.

The values for these convolutional gradients are shown, and all convolutional gradients are visualized in the network diagram in the third column. We note that there is a high variability in the value of the gradient. For example, identity 15's gradients are much higher than identity 14's. Thus, in our network visualizations, we normalize based on the highest gradient in the network.

C. Gradient-Selected Feature Evaluation

To verify the feature rankings produced by the network, we *knock out* these units and observe their effects on network performance.

For each identity (we use identities 7, 10, 12, 14, and 15), we find the top 10 strongest features. Then for each of these features, we set the output of the corresponding unit to 0 (knocking it out). We then run the modified network to evaluate its performance. The effects of the knockout are cumulative; e.g. at the first iteration we knock out the best feature, then we knock out the best 2 features, etc.

We evaluate network performance in two ways. First, we measure class error by only testing the network on the identity whose strongest features were extracted. Second, we measure overall error by testing the network on all identities.

Finally, we do this same experimental procedure for the 20 weakest features. We use the entire Yale data set, not just the test set.

D. Gradient-Selected Feature Results

Knockout experiment results are shown in figure 2Charts show the effect of removing an identity's strongest features on network performance. Two types of error are considered - error within the identity, and error among all images. Left column shows the removal of strongest features one-by-one, and right column shows the removal of weakest features. Features are rated based on class gradient. figure.caption.5.

Removing three or more of the strongest features impacts network performance. There seems to be some robustness to the network, in that it can handle minor damage and still classify well. This robustness depends on the identity; e.g. the network fails to recognize identity 12 after removing 4 units, but only fails on identity 10 after removing 8.

Overall performance seems to degrade slower than performance on a single identity, at least for the top features for identities 10 and 15. Regardless, all of the strongest feature near 100% error after removing 10.

Removing the weakest units gradually increases error as features are knocked out, but the pattern is not certain. For example, error for identity 12 increased drastically after removing the 7 weakest features, but then dropped again after removing two more. This suggests that some weak features, actually hurt network performance.

Overall performance degrades very slowly when removing features from the weakest first. The network almost completely fails after removing the top 10 strong features for any identity. In contrast, after removing 10 weak features, the network still achieves less than 25% error. Even after removing 15 weak features, overall error for many of the weak feature sets is below 50%. This is remarkable, considering that there are only 23 convolutional features.

Misclassifications for the knockout procedure can be seen in figure 3How the neural net misclassifies as the strongest features for identity 7 are knocked out. Lines represent identities; rows represent average prediction. On the left, all

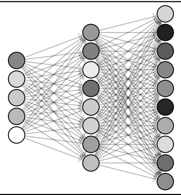
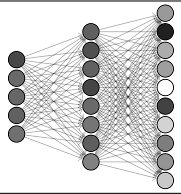
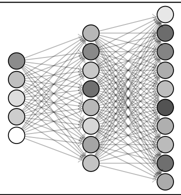
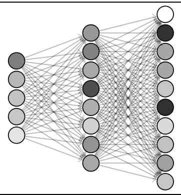
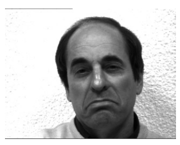
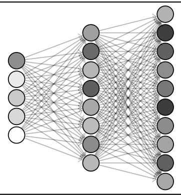
ID	Image	Network	Strongest Features			Weakest Features		
			Layer	Unit	Gradient	Layer	Unit	Gradient
15			1	5	0.25153	3	2	0.03227
			2	3	0.23269	3	6	0.03676
			3	8	0.21675	3	3	0.09039
			1	2	0.21362	3	9	0.10847
			3	1	0.20919	2	4	0.10995
			2	6	0.20612	2	2	0.12622
			2	5	0.19961	3	4	0.13164
			1	3	0.19855	1	1	0.13169
14			3	5	0.00202	3	2	0.00025
			3	7	0.00167	3	6	0.00052
			3	10	0.00158	2	4	0.00055
			3	3	0.00136	1	1	0.00057
			3	4	0.00125	2	2	0.00063
			3	1	0.00121	2	7	0.00075
			3	9	0.00117	2	1	0.00076
			2	8	0.00102	2	3	0.00077
7			1	5	0.02247	3	6	0.00728
			3	1	0.02029	3	2	0.00951
			1	3	0.01951	3	9	0.00977
			2	6	0.01890	2	4	0.00983
			1	4	0.01802	3	3	0.01157
			2	3	0.01760	2	2	0.01199
			2	8	0.01726	1	1	0.01228
			1	2	0.01683	2	7	0.01453
10			3	1	0.00369	3	6	0.00071
			1	5	0.00328	3	2	0.00072
			3	7	0.00316	2	4	0.00114
			2	6	0.00301	1	1	0.00182
			1	4	0.00287	2	2	0.00189
			3	5	0.00287	2	7	0.00212
			3	10	0.00286	2	1	0.00219
			1	3	0.00282	3	9	0.00231
12			1	5	0.08089	3	6	0.01925
			1	2	0.07403	3	2	0.01964
			1	4	0.06849	2	4	0.03013
			1	3	0.06410	3	9	0.03074
			2	6	0.05959	3	3	0.03077
			2	8	0.05814	2	2	0.03345
			2	3	0.05761	3	5	0.03780
			3	1	0.05623	3	7	0.04417

TABLE III: Five identities with their images, class gradients across the whole network, and top 8 strongest and weakest features, as ranked by the class gradient. These five identities were used in the knockout experiments.

subjects are correctly classified as no units are knocked out. As units are removed, the ability to distinguish between faces is lost. figure.caption.6. The strongest features for identity 7 were used. The colored lines in this image show the true class label, while the rows show the mode predicted class label. Interestingly, 7 is not the first identity to be grossly misclassified. As the network degrades, identities become more difficult for the network to distinguish.

VI. Observations

With the convolutional gradient and our experiments, we observed the following:

- Stability of representation. The neural net learns a sparse gradient pattern "profile" for each identity.
- Robustness of representation. Even when removing one or two of the strongest features in a neural network, performance remains high.
- Feature relevance imbalance. Removing three strong features gives the same performance reduction as removing the weaker half of all features. Strong units are useful across classes.

This analysis was made possible using prediction gradients and knockout analysis. Prediction gradients have the benefit of being efficiently computable, unlike modern methods for understanding neural networks, yet non-linear, unlike traditional measures such as correlation. This method is applicable not only to convolutional layers as demonstrated in this paper, but to any output in a neural network.

VII. Adversarial Image Generation

We now shift gears to apply the prediction gradient to understanding adversarial images.

First, we determine whether or not adversarial image masks generalize across elements of the same class.

Methods of generating adversarial examples such as the fast sign gradient method rely on creating a single perturbation from a single example [5]. This perturbation is then applied to that same example to induce misclassification. As generating such examples is difficult without access to the Jacobian of

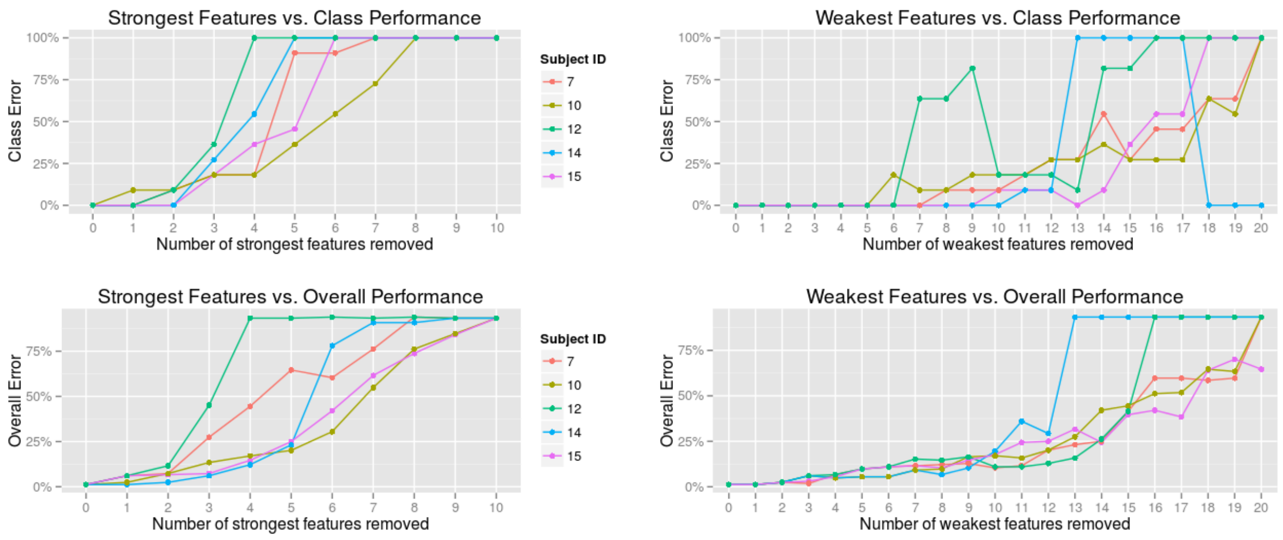


Fig. 2: Charts show the effect of removing an identity's strongest features on network performance. Two types of error are considered - error within the identity, and error among all images. Left column shows the removal of strongest features one-by-one, and right column shows the removal of weakest features. Features are rated based on class gradient.

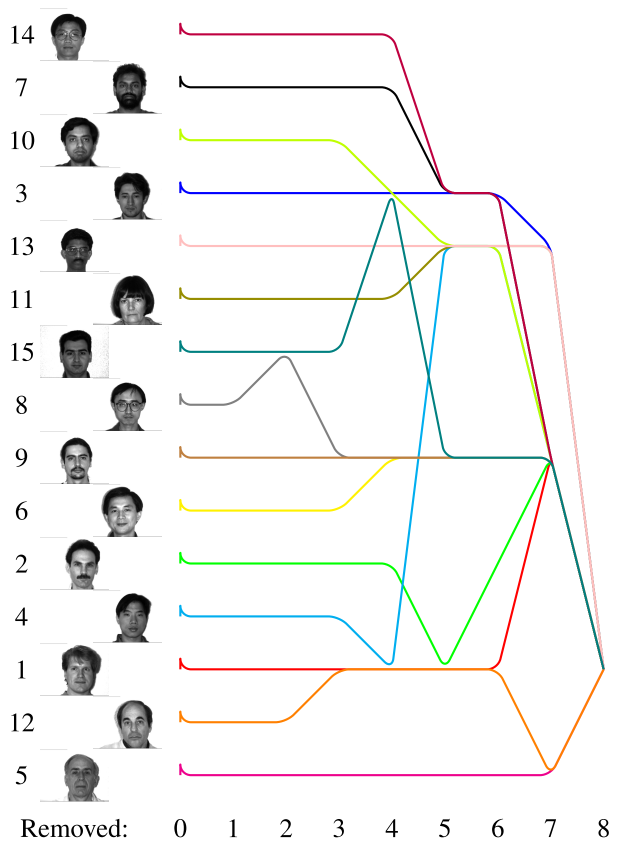


Fig. 3: How the neural net misclassifies as the strongest features for identity 7 are knocked out. Lines represent identities; rows represent average prediction. On the left, all subjects are correctly classified as no units are knocked out. As units are removed, the ability to distinguish between faces is lost.

the network, it would be ideal if an adversarial mask can be learned to consistently mask any example as a target class.

Second, we generate adversarial masks from a training set, and determine how well they deceive a network. To this end, we use a variant of the gradient sign method [5], and a DNN to which we have access to the gradient.

Third, we determine the generalizability of these masks. We apply them to different data sets, and to other classifiers, and observe the ability to target misclassification.

A. *Experimental Details*

We use a CNN with three ReLU, max-pooling convolutional layers, with 32, 32, and 64 feature maps, and a softmax classifier. We also used a momentum term of 0.9 and a learning rate of 0.01. There were no hidden or local response layers, and no average pooling. Because of these differences, we call our network pseudo-AlexNet [8].

The network trained for 10000 epochs on the CIFAR-10 [7] training set, and achieved 32.77% validation and 34.19% test error.

The choice of CIFAR-10 was intended to balance real-world usability and training feasibility. As CIFAR-10 consists of color images, its input space is much less structured than MNIST and closer to full-sized color images.

VIII. Generalizability within Classes

A. *Learning Deceptive Masks*

We use an additive mask W as a perturber. The cells of W add to cells of the input image. This mask is attached as input to the pseudo-AlexNet, and the pseudo-AlexNet is then run through its training set, but with all class labels changed the target class. The pseudo-AlexNet's weights are not adjusted

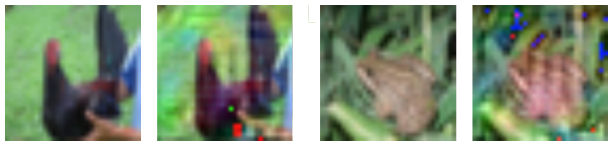


Fig. 4: Class-general perturbations learned using the gradients of the pseudo-Alexnet. Randomly selected. Left is a bird, right is a frog, and the target class for both is dog.

(they are already learned), but the error gradient propagates back to the image mask to adjust its weights.

Figure 4Class-general perturbations learned using the gradients of the pseudo-Alexnet. Randomly selected. Left is a bird, right is a frog, and the target class for both is dog. figure.caption.7 shows the results of a mask for a given class (dog). These images were randomly chosen. Error (with respect to the target class dog) was very high (88%), suggesting that masks do not generalize across samples in a class. Different classes did not perform any better.

B. Gradient Smoothness

The gradients of the mask are visualized in figure 5Visualization of gradients of three image predictions with respect to each pixel. Note that gradients are roughly normal with 0 mean and short tails. Columns of (a) correspond to columns of (b).figure.caption.8.

In 5Visualization of gradients of three image predictions with respect to each pixel. Note that gradients are roughly normal with 0 mean and short tails. Columns of (a) correspond to columns of (b).figure.caption.8a, the original image is shown, and gradients are shown in the noisy images to the right of them. These gradients have no clear pattern visually, thus challenging the notion that a classifier can learn to perturb images to fool another.

In 5Visualization of gradients of three image predictions with respect to each pixel. Note that gradients are roughly normal with 0 mean and short tails. Columns of (a) correspond to columns of (b).figure.caption.8b, the distribution of gradients is shown with respect to final-layer network outputs. The shape of this distribution suggests that perturbations do not happen as a result of a few pixels. It is due to small changes in many pixels.

IX. Targeted Misclassification

A. Linear Additive Masks

Unlike previous work, we generate adversarials with targeted misclassifications [15]. To do this, we calculate the gradient of the target output unit with respect to the input. This provides an additive image mask W which linearly approximates the best direction to perturb the image, and is scaled up or down with a coefficient η.

Using the pseudo-AlexNet, we calculate this mask for all images in the training set (essentially the Jacobian for all training images). We perform a line search for various values of η, ranging from 0 to 12750.

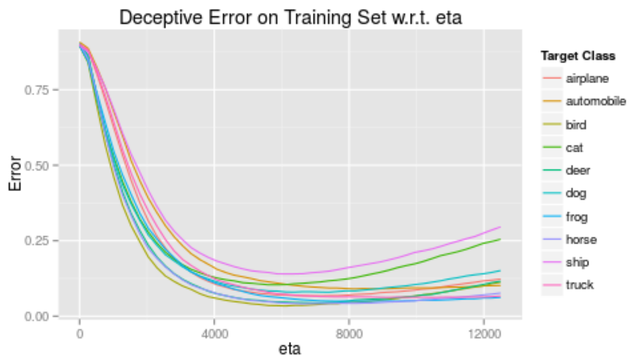


Fig. 6: Error for pseudo-AlexNet on CIFAR-10 training set as a function of η. Error for a target class is the proportion of samples successfully masked as the target class.

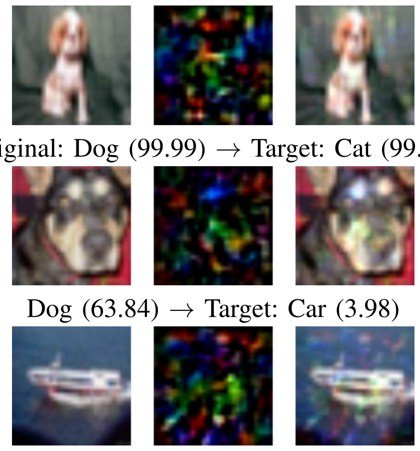

Original: Dog (99.99) $\rightarrow$ Target: Cat (99.99)
Dog (63.84) $\rightarrow$ Target: Car (3.98)
Ship (99.99) $\rightarrow$ Frog (0.00)

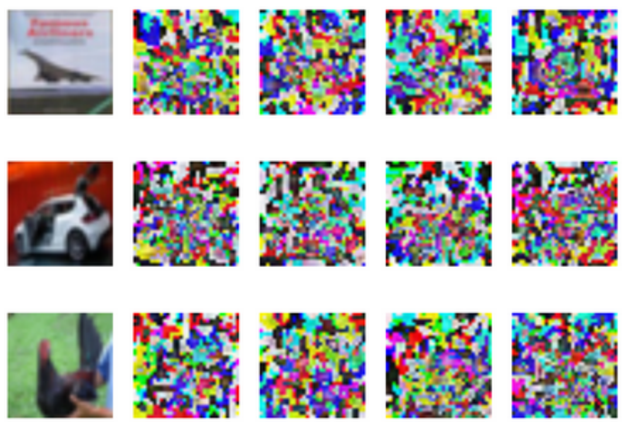
Fig. 7: Top: image and its mask in pseudo-Alexnet training set. Middle: image from validation set and its closest mask. Bottom: training set mask applied to MLP.

As shown in figure 6Error for pseudo-AlexNet on CIFAR-10 training set as a function of η. Error for a target class is the proportion of samples successfully masked as the target class. figure.caption.9, given the right value for η, *these image masks can perturb any image in the training set to be misclassified as any other class with very little error.*

The optimal values for η vary beetween 5000 to 10000 for each target class, and are usually around 7000. We call these the optimal η values.

B. Example Masks

Figure 7Top: image and its mask in pseudo-Alexnet training set. Middle: image from validation set and its closest mask. Bottom: training set mask applied to MLP.figure.caption.10 shows three examples of image masks obtained in the previous step. Even without explicity bounding distortion as in [15], the optimal distortion is rather low for most images. The top row image is typical of many learned masks.


(a) Column by column: three images; pixel gradients with respect to prediction; pixel gradients with respect to three final-layer unit outputs.

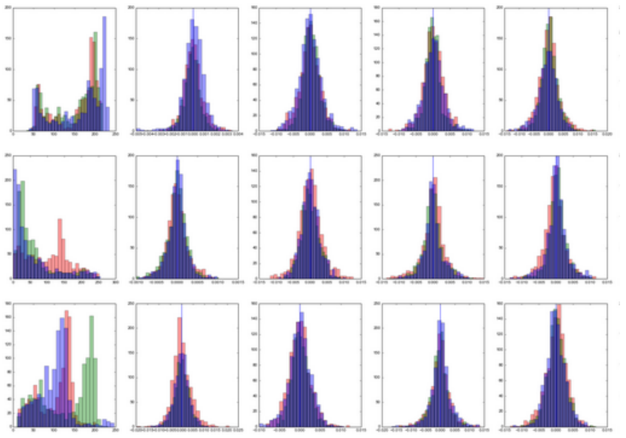

(b) RGB histogram of three images; histogram of prediction gradients for each color; gradients for three network outputs. Line denotes 0.

Fig. 5: Visualization of gradients of three image predictions with respect to each pixel. Note that gradients are roughly normal with 0 mean and short tails. Columns of (a) correspond to columns of (b).

Some masks induce more distortion compared to those on ImageNet [13], but similar to those on MNIST [15]. ImageNet images are higher resolution, and thus have more pixels (directions) to perturb towards misclassification. This suggests that adversarials may be more insidious for large-scale problems.

X. Perturbation Transferability

A. Dataset Transferability

Adversarial masks do not generalize across the same class, as seen in Section 2. In order to be useful for other data, we must be able to obtain masks for new data without the gradient. Our approach:

1) Given a new input image q, find its nearest image r in Euclidean distance from the training set.
2) Given the target class, use the pre-computed gradient mask that class with respect to r to perturb q.
3) Scale up q with the learned optimal η value.

This model was tested using the pseudo-AlexNet on the Cifar-10 test set. Results are shown in the upper half of figure 8Error for pseudo-AlexNet on CIFAR-10 its validation set, using 1-nearest-neighbor matching, and error for MLP on its training set. figure.caption.11. Though some target misclassifications can be generalized, e.g. frog and truck, in general the technique does not induce the target misclassification reliably.

B. Model Transferability

To test whether adversarials transfer across models, we learn an MLP on the same training data. The model has 500 hidden units, Tanh activation functions, and a learning rate of 0.1. As with the pseudo-AlexNet, 10000 iterations were used, leading to 55.17% validation and 54.22% test error. Testing was done on the training images, and masks from the pseudo-AlexNet were used for the MLP.

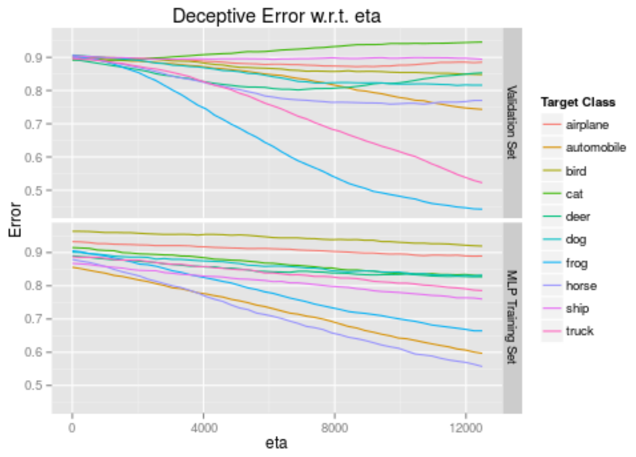


Fig. 8: Error for pseudo-AlexNet on CIFAR-10 its validation set, using 1-nearest-neighbor matching, and error for MLP on its training set.

The bottom of figure 8Error for pseudo-AlexNet on CIFAR-10 its validation set, using 1-nearest-neighbor matching, and error for MLP on its training set. figure.caption.11 shows that the effectiveness of the masks gradually increase with larger η, but the masks transferred to MLPs are not nearly as effective as on the original pseduo-AlexNet. However, we note these masks seem moderately effective on the pseudo-AlexNet with the validation set, and the MLP on the training set.

XI. Discussion

Overall, we have observed the following about adversarial examples:

- Adversarial perturbations often result from many small changes, rather than a few large changes.

- Adversarials tend to be close to the original data; hard constraints are not required [15].
- Adversarials can be targeted.
- Adversarials (using masks) are specific to an image, and not very generalizable to new data and different models.
- Adversarials are less noticeable in larger images.

It is possible to use more complicated procedures than nearest neighbor to obtain a suitable gradient mask for a unit. Though adversarial examples are still a long ways from being utilized effectively, investigating their generalizability may reveal common threads between deep neural networks and other models which share their quirks.

References

[1] Peter N. Belhumeur, João P. Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(7):711–720, July 1997.

[2] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[3] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *CoRR*, abs/1310.1531, 2013.

[4] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. Technical Report 1341, University of Montreal, June 2009.

[5] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. *ICLR*, 2015.

[6] Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. *CoRR*, abs/1412.5068, 2014.

[7] Alex Krizhevsky. University of toronto. Technical report, 2009.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*. 2012.

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[10] Henry Z. Lo, Joseph Paul Cohen, and Wei Ding. Prediction gradients for feature extraction and analysis from convolutional neural networks. In *International Conference and Workshops on Automatic Face and Gesture Recognition*, pages 1–6, 2015.

[11] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[12] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *CVPR*, 2015.

[13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, 2014.

[14] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *ICLR*, abs/1312.6034, 2013.

[15] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2015.

[16] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *ICLR*, abs/1311.2901, 2013.

[17] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. Deconvolutional networks. In *In CVPR*, 2010.