AdjointBackMap: Reconstructing Effective Decision Hypersurfaces from CNN Layers Using Adjoint Operators

Qing Wan, Yoonsuck Choe
Department of Computer Science and Engineering, Texas A&M University
College Station, TX, 77843, USA

Abstract
There are several effective methods in explaining the inner workings of convolutional neural networks (CNNs). However, in general, finding the inverse of the function performed by CNNs as a whole is an ill-posed problem. In this paper, we propose a method based on adjoint operators to reconstruct, given an arbitrary unit in the CNN (except for the first convolutional layer), its effective hypersurface in the input space that replicates that unit’s decision surface conditioned on a particular input image. Our results show that the hypersurface reconstructed this way, when multiplied by the original input image, would give nearly exact output value of that unit. We find that the CNN unit’s decision surface is largely conditioned on the input, and this may explain why adversarial inputs can effectively deceive CNNs.

Introduction
Convolutional Neural Network (CNN) has seen great success in computer vision (CV). It is based on the receptive fields found in biological vision (Hubel and Wiesel 1968), implemented by convolution (Fukushima 1988), and trained by backpropogation (LeCun et al. 1989). Since 2009, hardware advancements like GPUs (Raina, Madhavan, and Ng 2009) enabled fast computation of CNNs, securing their dominance in the field of machine learning.

CNNs learn gradually more complex visual features through its multi-layer convolutional architecture. Deconvolution (Matthew and Fergus 2014) can illustrate what is represented in each layer: edge features at the lower layer, and object features are higher layers. CAM (Zhou et al. 2016) and its successor Grad-CAM (Selvaraju et al. 2017) can highlight specific features in the input image that contributes most to the decision made by the CNN. Especially, Grad-CAM, with guided backpropagation, locates input feature that directly influenced the output (Springenberg et al. 2014), indicating which specific feature the CNN used.

However, it is also known that CNNs are vulnerable to adversarial attacks. Usually, these adversarial patterns cannot deceive humans. For example, Goodfellow, Shlens, and Szegedy (2014) showed an adversarial patterns that makes a CNN to mistakenly recognize a panda as a gibbon by adding a low level of noise while the original panda image and its contaminated version are visually identical to a human. Also a recent research showed that physical adversarial attack (a T-shirt worn by a human with an adversarial pattern) can deceive a CNN based object detection model YOLOv2. Thus, the way CNN processes visual input may be profoundly different from those of humans.

These differences leave us with a fundamental question: how does a CNN make its decision? Furthermore, what does the decision boundary for a feature map unit in the CNN look like? However, CNNs employ a cascade of convolutional layers with nonlinear activations like ReLU (Hahnloser et al. 2000). A kernel in a specific layer only plays its hyperplane role on that layer. It doesn’t give us much intuition on its role with respect to the input space, if we take the kernel out and visualize it directly, especially as we go higher in the layer. Inverting the function performed by a feature map may be a better way. However, inverse of a CNN is an ill-posed problem because bijection does not exist in the mapping from input to CNN output.

In this paper, we introduce a novel algorithm (AdjointBackMap) that aims at precisely reconstructing an effective decision boundary of a CNN. We circumvent the difficulty of finding the inverse by mapping every convolution kernel (in higher layers) or weight vector (in the fully-connected [FC] layer) back to the input image space with the help of adjoint operators (Banach 1987). As long as two necessary conditions are satisfied (see the Algorithms section), our theory guarantees that,

• In any convolutional layer, AdjointBackMap maps any convolutional kernel from that layer back to the original input image space with $RM_4 \rightarrow RM_1$ reconstruction modes (RMs);
• In fully-connected (FC) layers, AdjointBackMap maps the final decision weights from the top layer back to the input image space through $RM_0$ reconstruction mode (RM);
• Any effective hypersurface reconstructed above by AdjointBackMap is mathematically precise;
• Any single hypersurface reconstructed by AdjointBackMap represents the whole decision process from that input to a unit in the feature map or the output value of the FC layer output.

From our results, we learn that:
• Although a kernel convolves on every receptive field lo-
cation, it may not utilize information from all those locations.

• All effective hypersurfaces mapped by AdjointBackMap on high-level kernels are not human-recognizable patterns in the input image space. Since these hypersurfaces decide a high-level feature map or FC output, they suggest that CNN’s decision is not alike that of human vision.

• Any reconstructed effective hypersurface is conditioned on the current input image $x$. This explains why adversarial examples are possible because two effective decision boundaries (the hypersurface) of a CNN corresponding to two human-indistinguishable images can be very different from each other.

Based on these results, we can say that the way CNNs predict is significantly different from how human vision works. Effective hypersurfaces of CNNs are brittle due to the input-dependence, and adversarial examples can take advantage of this property.

Related Work

Understanding the decision process of a CNN is an active research direction towards explainable AI. Attempts to invert the CNN model includes works like (Dosovitskiy and Brox 2016), (Shrikumar, Greenside, and Kundaje 2017). Basically, these methods invert a learned feature map in CNN back to the input space so as to visualize parts of the input image that contributed to the feature map’s output. A slightly different approach to the above is to explain a CNN with gradient or saliency (Zhou et al. 2016; Dabkowski and Gal 2017; Selvaraju et al. 2017; Kim et al. 2018).

Perturbation is another path to estimate feature importance inside a CNN. It treats a deep learning model as a black box and observes how prediction changes when input varies (Ribeiro, Singh, and Guestrin 2016; Zintgraf et al. 2017; Petsiuk, Das, and Saenko 2018; Ibrahim et al. 2019; Ancona, ¨Oztireli, and Gross 2019).

What seems to be missing from the works above is that, when an input feature is identified through these methods from a given CNN, when that specific input feature pattern is fed back into the same CNN, the output classification should be the same. Furthermore, the actual output layer activation values should be similar to the activation due to the original input image. For example, consider the example is illustrated in Figure 1 based on (Gildenblat June, 2020 (accessed). Guided-bp pattern (right), fed into the same VGG16, gives a different prediction. If VGG16 did use the highlighted area on guided-bp in its "Boxer" prediction, we would expect to see the same "Boxer" when feeding this highlighted area. However, VGG16 predicts it as "Balloon".

Contributions

Our contributions are as follows:

• Utilization of the concept of Adjoint Operators in the analysis of decision process inside a CNN.

• Work directly on conv kernels instead of feature maps from the dual of input image space;

• Precisely recover a complex decision process leading to the activation value of a feature map or an FC-layer output inside a CNN by visualizing a reconstructed effective decision boundary.

• Discovery that CNN’s decision boundary is largely conditioned on the current input.

• Visualize variations in CNNs decision boundaries under adversarial attack.

• Visual evidences from effective hypersurfaces show CNN’s decision process is significantly different from our human vision.

Model

Notation

We will use the following mathematical notations.

• $\odot$ denotes convolution.

• $\theta_X$ denotes the origin of the vector space $X$;

• $\langle x \mid y \rangle$ denotes the inner product of $x, y \in X$;

• $X^*$ denotes the algebraic dual of $X$, i.e. the space of all linear functionals on $X$;

• $\langle x, x^* \rangle$ denotes the value of a linear functional $x^* \in X^*$ at $x \in X$;

• $B(X, Y)$ denotes the space of all bounded linear operators from $X$ to $Y$;
Theory

Convolution distinguishes CNNs from other artificial neural networks and gives its power. The convolution process acts on the in-channel feature map with receptive-field-sized weights (the kernel). Below, we will consider convolution from an algebraic dual perspective.

With the help of Frechet differential \( \delta F(x; h) \) \cite{Berger1977} \cite{Luenberger1977} on increment \( h \), that

\[
\| F(x + h) - F(x) - \delta F(x; h) \|_\mathcal{Y} \to 0, \| h \|_\mathcal{X} \to 0, \tag{1}
\]

where \( F : \mathcal{X} \to \mathcal{Y} \) be an operator from an open domain \( D \) of a normed space \( \mathcal{X} \) to a normed space \( \mathcal{Y} \), we are able to approximate a convolution inside a CNN layer. Suppose \( F \) denotes a forward path from a small input image \( x \in \mathcal{X} \) to an in-channel \( r_1 \times r_2 \) receptive field represented by \( \mathcal{Y} \) on the feature map of a CNN which has no bias unit, i.e. \( F(\theta_X) = \theta_Y \), then

\[
\| F(x) - F(\theta_X) - J_F(\theta_X)x \|_\mathcal{Y} \approx o(\| x \|_\mathcal{X}), \tag{2}
\]

where \( J_F : \mathcal{X} \to B(\mathcal{X}, \mathcal{Y}) \) denotes the Jacobian Operator, and \( o(\| x \|_\mathcal{X}) \) denotes higher order of \( \| x \|_\mathcal{X} \). Therefore, we write a kernel \( w_{r_1 \times r_2} \) that convolves on \( F(x_0) \) (\( x_0 \) is a fixed nonzero image) as,

\[
c = F(x_0) \circledast w_{r_1 \times r_2} = \langle F(x_0), w_{r_1 \times r_2} \rangle \approx \langle J_F(z(x_0))x, w_{r_1 \times r_2} \rangle, \tag{3}
\]

where \( c \in \mathbb{R} \) represents a unit in the out-channel feature map; \( J_F(z(x_0)) \in B(\mathcal{X}, \mathcal{Y}) \). For \( \exists z(x_0) \in \mathcal{X} \) turns the last “\( \approx \)” to “\( = \)”. Later our experiment will show a replacement that \( z(x) = \frac{5}{M} \) for a large \( M \in \mathbb{N} \) will achieve this approximation when a CNN is activated by ReLU.

Considering the input image space \( \mathcal{X} \) as a Hilbert space with inner product defined element-wise as below:

\[
\langle x_{H \times W \times C} | y_{H \times W \times C} \rangle = \sum_{i=1}^{H} \sum_{j=1}^{W} \sum_{k=1}^{C} x_{i,j,k} y_{i,j,k}, \tag{4}
\]

by Adjoint Operator \( J^*_F(z(x_0)) \) of the Jacobian, we have

\[
c = \langle J^*_F(z(x_0))x_0, w_{r_1 \times r_2} \rangle = \langle x_0 | J^*_F(z(x_0))w_{r_1 \times r_2} \rangle = (\langle x_0 | J^*_F(z(x_0)) \rangle)^T w_{r_1 \times r_2}. \tag{5}
\]

Here \( w_{r_1 \times r_2} \in \mathcal{Y}^* \) and \( J^*_F(z(x_0)) \in B(\mathcal{Y}^*, \mathcal{X}^*) \).

From Riesz Representation theorem \cite{Rolland1999} (a hyperplane: \( \{ x \in \mathcal{X} | \langle x | J^*_F(z(x_0))w_{r_1 \times r_2} \rangle = c \} \)), we know \( \mathcal{X}^* \) is actually \( \mathcal{X} \) itself. So the adjoint operator will map a convolutional kernel sitting on any layer of a CNN (except the first convolutional layer) back to the input image space that serves as an effective hypersurface representing all decision boundaries forward from the input to the out-channel feature map in that layer, which enables our visualization.

One thing to mention is that \( J^*_F(z(x_0))w_{r_1 \times r_2} \) is in the dual space, \( \mathcal{X}^* \), of \( \mathcal{X} \) (Figure 2). We will view this

\[ J^*_F(z(x_0))w_{r_1 \times r_2} \] in the dual space instead of in the input image space if the Riesz Representation is not applied. In other words, Riesz Representation frees us from visualizing two distinct spaces.

Algorithm

Generally, we deploy our AdjointBackMap on two kinds of layers in a CNN (Figure 2).

- We map a filter from any convolutional layer (except for the first layer, from which a kernel has belong to the dual of input) back to input image space and visualize it as an effective hypersurface that represents all decision boundaries on the forward path from input to the activation value on the out-channel feature map.
- We map weight vectors in the last FC layer back to input image space and visualize a single reconstructed hypersurface that decides the prediction of CNN.

As we mention before, AdjointBackMap requires two necessary conditions to enable it. These two necessary conditions are:

- The CNN should not have any bias unit inside. Otherwise, Equation [2] would not be properly established. For example, a CNN using batch normalization \cite{Ioffe2015} may not be qualified for our technique.
- Approximation of a forward path \( F \) from input to a receptive field on an in-channel feature map should satisfy Equation [3]. In detail, there exists \( z(x) \) such that difference between the \( F(x) \) and \( J_F(z(x))x \) is relatively small. In this case, care should be taken when deploying our method on a CNN using activation functions whose derivative is not piecewise constant (like tanh).

Basically, our method shows that the effective hypersurface reconstructed by AdjointBackMap is able to reproduce a CNN unit’s activation given an input image through Equation [4] and the value of that activation should be the same as the one by propagating the same input through the CNN forward to the out-channel feature map or the FC layer. That is, if we dot-product this effective hypersurface to the input image directly, the returned value will precisely match the result obtained from the CNN itself. This is a crucial point that distinguishes our model from other methods developed in the field of explainable AI.

In practice, AdjointBackMap provides five reconstruction modes (RMs). Four of them acts on a convolutional layer and one on a FC layer. Conv layers have four RMs due to the two factors below:

- Whether to separate mapping along the stride preset during training. When a kernel moves by a stride, its corresponding receptive field on the feature map will change accordingly. In short, \( F \) (Equation [3]) will change when a kernel moves.
- Whether to separate mapping along in-channel feature maps or its corresponding in-channel kernels. This is because every in-channel feature map convolves with its corresponding in-channel kernel before merging to form an out-channel feature map.
Figure 2: Figure shows geometrical relationship between $J_F(z(x_0))$ and $J'_F(z(x_0))$. Besides, it depicts how we apply this to either conv layer or FC layer ($\approx$ refers to Equation [5]).

We explain with an example. Suppose a CNN takes a $32 \times 32 \times 3$ (height $\times$ width $\times$ channels) RGB image $x$. Its third convolutional layer has in-channel feature maps of shape $16 \times 16 \times 32$. That layer is equipped with convolutional kernels of $3 \times 3 \times 32 \times 64$ (height $\times$ width $\times$ in-channels $\times$ out-channels). The convolutional stride is 2 with padding 'same' (Abadi et al. 2016). We name the five RMs of AdjointBackMap as $RM_4$ to $RM_0$.

\[ \text{(RM}_4\text{)} \quad \text{Convolutional kernels are separated along in-channels. Or equivalently, each in-channel feature map is separated to convolve with its own kernel. AdjointBackMap works on one at a time. Also, each stride move is backward mapped independently. So the total kernels are } 32 \times 64 \text{ (range of } j, i \text{, respectively) and each kernel has its shape of } 3 \times 3. \text{ Each stride offset } s \text{ generates its own effective hypersurface of } 32 \times 32 \times 3 \text{ and total moves are } 8 \times 8. \text{ Therefore effective hypersurfaces reconstructed through } RM_4 \text{ are (from Equation [5]),}
\]

\[ H_{s,j,i}^{Adj}(z(x)) = J_{F,s,j,i}^*(z(x))w_{3\times3,s,j,i}, \quad (6) \]

where $s \in \{0, 1, ..., 63\}, j \in \{0, 1, ..., 31\}, i \in \{0, 1, ..., 63\}, F_{s,j,i}$ denotes a forward path from the input to the receptive field offset by stride $s$ on the $(j, i)^{th}$ kernel. In this case, the number of backward mappings is $8 \times 8 \times 32 \times 64$ and each one has its shape of $32 \times 32 \times 3$. Actually, this is the most basic application that exactly follows our theory. We are able to visualize what a single kernel is actually doing, from the perspective of input image space, on its effective receptive field with stride shift.

\[ \text{(RM}_3\text{)} \quad \text{Convolutional kernels are separated along in-channels. However, each stride sums together to form a single backward mapping. The total kernels are } 32 \times 64 \text{ and each kernel has its shape of } 3 \times 3. \text{ Each stride reconstructs its own effective hypersurface of shape } 32 \times 32 \times 3 \text{ and total moves are } 8 \times 8. \text{ Then all } 8 \times 8 \text{ moves sums together pixel-wise to form a single } 32 \times 32 \times 3 \text{ effective hypersurface. This is an unusual scope (usual scope for the layer right before global pooling (Lin, Chen, and Yan 2013)), which means we get an effective hypersurface } H_{s,j,i}^{Adj}(z(x)) \text{ in this way.}
\]

\[
\begin{align*}
8 \times 8 - 1 & \sum_{s=0}^{63} (F(x) \otimes w_{3\times3,s,j,i}) = \langle x, \sum_{s=0}^{63} H_{s,j,i}^{Adj}(z(x)) \rangle \\
& = \langle x, H_{s,j,i}^{Adj}(z(x)) \rangle. \quad (7)
\end{align*}
\]

First ”$=$“ due to Equation [5] and linearity in duality. Relationship to $RM_3$ is marked by the last ”$=$“. In this case, the number of backward mappings is $32 \times 32 \times 3$. Effective hypersurfaces reconstructed from $RM_3$ are

\[ H_{s,i}^{Adj}(z(x)) = \sum_{j=0}^{31} J_{F,s,i,j}^*(z(x))w_{3\times3,s,j,i} \]

\[ = \sum_{j=0}^{31} H_{s,j,i}^{Adj}(z(x)). \quad (8) \]

Similarly, relationship to $RM_1$ is marked by the last ”$=$“. This effective hypersurface decides a single unit’s activation in the out-channel feature map.

\[ \text{(RM}_1\text{)} \quad \text{Convolutional kernels are not separated along in-channels any more, but each stride is still backward mapped independently. So the number of backward mappings is } 8 \times 8 \times 64 \text{ and each one has its shape of } 32 \times 32 \times 3. \text{ Effective hypersurfaces reconstructed from } RM_1 \text{ are}
\]

\[ H_{s,i}^{Adj}(z(x)) = \sum_{j=0}^{31} H_{j,i}^{Adj}(z(x))w_{3\times3,s,j,i} \]

\[ = \sum_{j=0}^{31} H_{s,j,i}^{Adj}(z(x)). \]

\[ \text{In other words, inner product between input image and the effective hypersurface is equal to the pixel-wise summation of the out-channel feature map. Relations to } RM_3 \text{ and } RM_2 \text{ are listed as well.}
\]

\[ \text{(RM}_0\text{)} \quad \text{The backward mapping deploys on the weight vectors } \{w_k\} \text{ where } k \text{ denotes class index in FC layer of the CNN. Then the output value for class } k \text{ (before going through activation) is decided by effective hypersurface } H_k^{Adj}(z(x)) \text{ of shape } 32 \times 32 \times 3, \text{i.e.,}
\]

\[ H_k^{Adj}(z(x)) = J_{F}^*(z(x))w_k, \quad (10) \]

where $k \in \{0, 1, ..., (N - 1)\}$, $N$ denotes the quantity of classes.
Implementation: We use convolution to compute duality in Eq.5 as they are equivalent and duality can use hardware acceleration. Due to computationally expensive Jacobian, optimization is needed, summarized in Algorithm 1. Note conv2d, unstack, stack, expanddim, matmul are functions defined in TensorFlow (Abadi et al. 2016). Padding of conv2d is the same as training. Also conv2d has an ‘axis’ choice in order to replace the transpose in Eq.5.

Though an effective hypersurface acquired from AdjointBackMap is in the same space as the original input image, the scalar element values of the effective hypersurface might not lie on the same interval as its original image which has values in [0, 1]. In that case, we have to properly normalize the value so as to enable its visualization. We will explain more in our experiments.

Experiments

Pre-trained Model

We discuss below how we trained our CNN model. Dataset: We used CIFAR10 (Krizhevsky, Hinton et al. 2009) as our dataset. CIFAR10 contains 50,000 32 x 32 RGB images for training and 10,000 for testing and classes are 10. We separated the 50k samples to training and validation set with ratio 9 : 1, i.e. 45k, 5k, respectively. All analyses were conducted on the test set.

From Equation 2 we should make the norm of each sample small. However, we cannot make too small. The reason is, for any training sample \( x \) in a data space \( X \), there exists a kernel \( w_0 \in X_{r_1 \times r_2} \) and a constant \( c > 0 \) such that,

\[
\|\langle x_{r_1 \times r_2}, w_0 \rangle \| \geq c.
\]

Otherwise, the CNN layer will be silent. Then by the definition of \( \|w_0\|_X \),

\[
\|x\|_X \times \|w_0\|_X \geq \|\langle x_{r_1 \times r_2}, w_0 \rangle \| \geq c.
\]

This serves as an “uncertainty principle” inside the model. That is, if \( \|x\|_X \) was too small, then \( \|w\| \) would have to compensate to grow bigger and enlarge regularization penalty during training, which will finally cause the CNN to fail. So we set the range of pixel value from [0, 1] to [0, 0.25] by dividing each pixel value with 4 so as to make our CNN trainable in our experiments.

Data Augmentation: We use data augmentation for training. An input color image goes through random flipping of left to right, random adjustment of saturation within [0.0, 2.0], random adjustment of contrast within [0.4, 1.6], random adjustment in brightness with 0.5, resizing to 36 x 36 x 3 and then randomly cropping to 32 x 32 x 3.

Model: We used VGG (Simonyan and Zisserman 2014) with seven activation layers (VGG-7). Learnable parameters by layers are listed in Table 1 (Appendix). A ReLU activation is placed after each convolved feature map except for the pooling layers. After global pooling, it will fully connect to prediction layer for 10 classes with a ReLU. Our VGG-7 does not have any bias or batch normalization.

Cost and Accuracy: We regularized the kernels by \( L_1 \) (factor 10\(^{-4}\)). We used cross entropy with softmax as cost.

Algorithm 1 AdjointBackMap with modes from \( RM_k \) to \( RM_0 \)

Input: (a) \( x_d \): input \( (d = H \times W \times C) \); (b) \( z \): function for Eq.3 (c) \( T \): pre-trained model; (d) \( i \): layer index; (e) \( s \): stride during training; (f) \( L \): RM number.

Output: Effective hypersurfaces \( H^{adj}(z(x_d)) \)

function ADJOINTBackMap(x_d, z, T, i, s, L)
\( z_0 = z(x_d) \)
switch \( L \) do

\( L = 'RM_0' \):
load \( w_{cin \times clabel} \) from FC layer of \( T \)
load \( F_{cin} \) from \( T \)
\( J_F = \frac{\partial F_{cin}}{\partial z} \)
return matmul(\( J_F (z_0) \), \( w_{cin \times clabel} \), axis=cin)

\( L = 'RM_k' \) or \( 'RM_{3}' \):
load \( w_{r_1 \times r_2 \times cin \times cout} \) from \( T \) at \( i \)
load \( F_{hi \times wi \times cin} \) from \( T \) at \( i \)
\( J_F = \frac{\partial F_{hi \times wi \times cin}}{\partial w_{r_1 \times r_2 \times cin \times cout}} \)
\( w \) = unstack(\( w_{r_1 \times r_2 \times cin \times cout} \), axis=cin)
\( J_F = \) unstack(\( J_F \times \|w\|_c \), axis=cin)
Empty container \( R \), \( j = 0 \)
while \( j < cin \) do
\( J_F = \) expanddim(\( J_F\) at \( j \), axis=cin)
\( w = \) expanddim(\( w \) at \( j \), axis=cin)
\( R \) = append(\( J_F \times \|w\|_c \times \|w\|_c \times \|w\|_c \), \( w \), stride=s,
axis=(\( H_2 \times W_2 \times cin \times cout \)))
\( j = j + 1 \)
end while

\( L = 'RM_{3}' \) then
return \( \text{conv} \times \|w\|_c \times \|w\|_c \times \|w\|_c \)
end if

\( L = 'RM_{3}' \) then
return \( \text{conv} \times \|w\|_c \times \|w\|_c \times \|w\|_c \)
end if

end switch
end function
Accuracy is measured by a prediction index being exactly matched with its label (Top-1 accuracy).

Training, Validation, Test: We trained our model with GD (gradient descent) optimizer on a RTX2080Ti for a total of 301 epochs. We set the learning rate to 0.01 and batch size to 50. We trained our model on 15k samples every epoch. We validated the trained model on 5k samples every two epochs and if a higher validation accuracy is detected, the trained model will be saved. Test was conducted on 10k samples. VGG-7 reported test accuracy of 79.4% after training.

Five Experiments With Respect to Five RMs

We first verify the existence of a function $z$ in Equation 4 (Figure 4 and Figure 7). Then we move to five experiments related to five RMs ($RM_4$ to $RM_1$ are shown in the Appendix).

Visualization of $RM_0$: $RM_0$ works on weights of FC layer to generate a set \{ $H_{k}^{Adj}(z(x_0))$ \mid $k \in \{0, 1, ..., 9\}$ \} (Equation 10). Each $H_{k}^{Adj}(z(x_0))$ represents an effective hypersurface that decides the $k^{th}$ value of the FC output (before ReLU6) in VGG-7. That is, the VGG-7 taking input and passing forward through layers to make a prediction is equivalent to doing ten inner products between the input image and our \{ $H_{k}^{Adj}(z(x_0))$ \}. $H_{k}^{Adj}(z(x_0))$ has its shape of $32 \times 32 \times 3$. We apply the same techniques as $RM_4$ (Appendix). Just for visualization, we apply square root to values inside $H_{k}^{Adj}(z(x_0))$.

Results: We use those two images from $RM_4$ (Appendix). Also we add one image that has the same ship label. Results are illustrated at Figures 3 and Figure 16.

![Image 3](image3.png)

Figure 3: $H_{k}^{Adj}(\frac{z(x_0)}{\sqrt{3}})$ patterns mapped from the FC layer (Equation 10) by $RM_0$ (a and c), and their corresponding inputs (b and d, respectively). Number of subfigures in (a) and (c) is equal to the number of classes.

![Image 4](image4.png)

Figure 4: Histogram of relative errors between $F_i(x)$ and $\langle x \mid H_i^{Adj}(\frac{z}{\sqrt{3}}) \rangle$ for $i \in \{0, 1, ..., (c_{out} - 1)\}$, by Conv1~5 layers and FC layer (before ReLU6). 10k test samples. The x-axis is the error, and the y-axis is the frequency. In all cases, AdjointBackMap reproduces the neuron’s activity across Conv layers and the FC layer with nearly 0 error.

Analysis of Experiments

As the layer goes higher, effective receptive field (non-black pixels or non-zero area) enlarges. This is obvious from plots of both $RM_4$ and $RM_2$. The reason is if a kernel in Conv0 has a shape of $3 \times 3$, a same size kernel on Conv1 actually will have a maximal effective receptive field of $5 \times 5$ from the original input perspective because of stride offset $s = 1$. However, few kernel does fully utilize its effective receptive field. Even, some layer has a kernel that takes null from input image (its effective field blacks out) on its effective receptive fields and therefore makes $H_{k}^{Adj}(\frac{z(x_0)}{\sqrt{3}})$ (equation 17, Appendix) sparse.

Effective hypersurfaces from $RM_1$, $RM_2$ and $RM_0$ are not human recognizable patterns. For example, pattern of $H_{k=8}^{Adj}(z(x_0))$ in Figure 5 shows neither a clear ship nor a rough contour of ship. Patterns of Figure 5(a) are significantly different from Figure 5(c) although they are the same class (ship).

Even, a same kernel shows different AdjointBackMap patterns as stride moves. That means a kernel may make different decision from the perspective of input image when it moves to different receptive field though the kernel itself doesn’t physically change at all during stride move. However, both $RM_3$ and $RM_1$ have coherent colored shapes at low-level convolutional layers. But these shapes gradually turn to irregular pixels as layer goes higher. These imply CNN’s decision is sensitive to values in each pixel in the input image, which is different from humans who may ignore small variation in pixel values.

An interesting finding is that changing an input image forces the effective hypersurfaces $H_{k}^{Adj}(z(x))$ to vary at the same time. This has been verified in Figure 4. Formally, an input image $x$ goes through VGG-7’s $F$ to get an output value from the FC layer, $F_k(x)$, for the $k^{th}$ class, will have $F_k(x) = \langle x, h_k(x) \rangle$, where $h_k(x)$ is the effective decision boundary for the $k^{th}$ output from FC layer. So, the CNN decision boundary is very sensitive to the current given input.

Applications to Adversarial Inputs

We apply AdjointBackMap to the analysis of adversarial inputs to further probe the functional properties of CNNs. A well known adversarial attack comes from (Goodfellow, Shlens, and Szegedy 2014) that by adding an intended noise to “panda”, a CNN makes a wrong turn to “gibbon” from previous correct prediction. It seems not reasonable from
our human’s eye because adding that “noise” doesn’t visually change that “panda” much. Our AdjointBackMap recovers its decision process by reconstructing effective hypersurfaces \( \{ H_k^{\text{Adj}}(z(x_0)) \} \). And we are able to directly visualize those decision boundaries at this time.

**Experiment A:** We use the same method as [Goodfellow, Shlens, and Szegedy (2014)]. Let \( C \) denote a cost we use to train a CNN \( F \), then an adversarial noise is:

\[
Advr = 0.001 \times \text{sign}(\nabla C_F(x)).
\]

Our input pixel range is \([0, 0.25]\) which is different from usual range \([0, 1]\). So we choose a smaller factor than 0.007 in the original work. We add this pattern to input image and fool our VGG-7 from “automobile” to “dog”. We first have to verify equation (15) (Appendix). Results are listed in table 3 (Appendix). Thus, we confirm that \( \{ H_k^{\text{Adj}}(x_0 + \text{Advr}) \} | k \in \{0, 1, ..., 9\} \) is the set of effective hypersurfaces that decide the VGG-7 predictions on \( (x_0 + \text{Advr}) \) for 10 classes.

**Experiment B:** We use the same method as Kurakin, Goodfellow, and Bengio (2016) and make 9 adversarial noises for the ship image (Figure 3(b)), \( \{ Advr_i \} | i \in \{0, 1, ..., 9\} \) (name \( S_{B_i} \)) where \( S \) is the label index, to mislead our VGG-7 to 9 other classes. We project \( \{ H_{k=8}^{\text{Adj}}(x_0 + \text{Advr}) \} | i \in \{0, 1, ..., 9\} \) with t-SNE (Pedregosa et al. 2011) for analysis.

**Experiment B:** We generate a set, \( S_{B_2} = \{ \beta_j \times \text{advr} | j < 50, j \in \mathbb{N} \} \cup \{ g_{m} | m < 50, m \in \mathbb{N} \} \) where \( g_m \sim \mathcal{N}(\mu = 2.21 \times 10^{-5}, \sigma^2 = 4.16 \times 10^{-6}) \), with 100 “noisy” samples. First, we sequentially iterate \( \text{advr} \in S_{B_2} \) for 9 epochs (skip \( i = 8 \)); Each epoch, if two conditions, misclassified (prediction of VGG-7 on \( x + \beta \times \text{advr} \)) is not equal to the label of \( x \) and over threshold (predicted value from misclassified index is greater than the sum of threshold (0.5) and predicted value from label index on VGG-7), are satisfied in loop \( \beta \) from 1.0 to 0 with a step=-0.05, a qualified \( \beta \times \text{advr} \) will be stored; We randomly shuffle before filtering 51 qualified ones to 50. Then, we generate 50 Gaussian noise with same statistical mean and variance of pixels from elements in \( S_{B_1} \). And none of Gaussian, when applied to the original image, will fool the VGG-7 prediction. We united them together as \( S_{B_2} \) and project \( \{ H_{k=8}^{\text{Adj}}(x_0 + \text{Advr}) \} | n \in S_{B_2} \) with Factor Analysis (Pedregosa et al. 2011) for analysis.

**Results:** Results of Experiment A are illustrated in Figure 5. Results of Experiment B1, B2 are illustrated in figure 7 (Appendix), Figure 6, respectively.

**Analysis:** Experiment A reveals that values of \( H_k^{\text{Adj}}(x_0 + \text{Advr}) \) are significantly different from \( H_k^{\text{Adj}}(x_0) \) (Figure 5(d)). We learn from Figure 5(b) that this difference starts from Conv1, the second convolutional layer, through our AdjointBackMap with \( RM_3 \), though no kernel in that layer changes. It implies that VGG-7 takes different roads to make decision in response to two visually identical images.

As we have seen in Figure 3 in spite of the same class, having different effective decision boundaries. Further Experiment B1 reveals that the effective boundary is brittle to different adversarial noises despite being visually indistinguishable. Experiment B2 further discloses the effective decision boundary is still weak under a set of linearly scaled adversarial input. These imply that visually similar images are able to easily knock off a CNN by misleading its decision process, which takes advantage of the fact that CNN is essentially different from human vision.

**Conclusions & Future Work**

We introduced adjoint-operator-based AdjointBackMap, which maps a kernel or weight vector back to the dual of the input space as an effective decision boundary. Using Riesz representation, we were able to project them back to the input space to enable visualization. AdjointBackMap works as long as certain basic conditions are satisfied. Through five reconstruction modes of AdjointBackMap, we were able to visualize the decision boundaries at different layers in the CNN, which we found to be different with human vision. Also, we learned that the decisions boundaries are sensitive to small changes in the input. We expect our work to motivate principled approaches to explainable AI and adversarial attacks.
References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. Tensorflow: A system for large-scale machine learning. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI}) 16, 265–283.

Ancona, M.; Öztireli, C.; and Gross, M. 2019. Explaining deep neural networks with a polynomial time algorithm for shapley values approximation. arXiv preprint arXiv:1903.10992.

Banach, S. 1987. Theory of linear operations. Elsevier.

Berger, M. S. 1977. Nonlinearity and functional analysis: lectures on nonlinear problems in mathematical analysis, volume 74. Academic press.

Dabkowski, P.; and Gal, Y. 2017. Real time image saliency for black box classifiers. In Advances in Neural Information Processing Systems, 6967–6976.

Dosovitskiy, A.; and Brox, T. 2016. Inverting visual representations with convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, 4829–4837.

Folland, G. B. 1999. Real analysis: modern techniques and their applications, volume 40. John Wiley & Sons.

Fukushima, K. 1988. Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural networks 1(2): 119–130.

Gildenblat, J. June, 2020 (accessed). Grad-CAM implementation in Keras. https://github.com/jacobgil/keras-grad-cam

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

Hahnloser, R. H.; Sarpeshkar, R.; Mahowald, M. A.; Douglas, R. J.; and Seung, H. S. 2000. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature 405(6789): 947–951.

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

Ibrahim, M.; Louie, M.; Modarres, C.; and Paisley, J. 2019. Global explanations of neural networks: Mapping the landscape of predictions. In Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society, 279–287.

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

Kim, B.; Wattenberg, M.; Gilmer, J.; Cai, C.; Wexler, J.; Viegas, F.; et al. 2018. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In International conference on machine learning, 2668–2677.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images.

Kurakin, A.; Goodfellow, I.; and Bengio, S. 2016. Adversarial examples in the physical world. arXiv preprint arXiv:1607.02533.

LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; and Jackel, L. D. 1989. Backpropagation applied to handwritten zip code recognition. Neural computation 1(4): 541–551.

Lin, M.; Chen, Q.; and Yan, S. 2013. Network in network. arXiv preprint arXiv:1312.4400.

Luenberger, D. G. 1997. Optimization by vector space methods. John Wiley & Sons.

Matthew, D.; and Fergus, R. 2014. Visualizing and understanding convolutional neural networks. In Proceedings of the 13th European Conference Computer Vision and Pattern Recognition, Zurich, Switzerland, 6–12.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. 2011. Scikit-learn: Machine learning in Python. the Journal of machine Learning research 12: 2825–2830.

Petsiuk, V.; Das, A.; and Saenko, K. 2018. Rise: Randomized input sampling for explanation of black-box models. arXiv preprint arXiv:1806.07421.

Raina, R.; Madhavan, A.; and Ng, A. Y. 2009. Large-scale deep unsupervised learning using graphics processors. In Proceedings of the 26th annual international conference on machine learning, 873–880.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. “ Why should I trust you?” Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 1135–1144.

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In Proceedings of the IEEE international conference on computer vision, 618–626.

Shrikumar, A.; Greenside, P.; and Kundaje, A. 2017. Learning important features through propagating activation differences. arXiv preprint arXiv:1704.02685.

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

Springenberg, J. T.; Dosovitskiy, A.; Brox, T.; and Riedmiller, M. 2014. Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806.

Zhou, B.; Khosla, A.; Lapedriza, A.; Oliva, A.; and Torralba, A. 2016. Learning deep features for discriminative localization. In Proceedings of the IEEE conference on computer vision and pattern recognition, 2921–2929.
Appendix

Notations
We will use the following notations.
- "Equation" refers to an equation in the main paper;
- "algorithm" refers to an algorithm in the appendix;
- "table" refers to a table in the appendix;
- "figure" refers to a figure in the main paper;
- "Algorithm" refers to an algorithm in the main paper;
- "Figure" refers to a figure in the appendix;
- "Equation" refers to an equation in the main paper;
- "RM(s)" is an abbreviation of Reconstruction Mode(s) from the main paper.

Five Experiments With Respect to Five RMs
As mentioned in the main paper, we verify the existence of a function \( z \) in Equation 3. Then we move to the four remaining experiments related to the four RMs (RM1, RM2, RM3, RM4). Also, we supplement one more figure for Figure 3 to illustrate RM0.

Verify The Existence of \( z \) Our AdjointBackMap needs two necessary conditions: first one is satisfied (no bias condition) and we just verify the second one, i.e. \( \exists z, \, z(x) \in X \).

Basically, we use Equation 3 and 4 to verify the second condition. We verify two types of tasks:
- Verify the existence of \( z \) for the four RMs on every convolutional layer;
- Verify the existence of \( z \) for the RM0 on FC layer.

Verify the existence of \( z \) on every conv layer: A convolutional layer \( i \) (except the first layer) of our VGG-7 is equipped with kernels, \( w_{r_1 \times r_2 \times c_{in} \times c_{out}} \), which convolves the in-channel feature maps \( \langle F(x_0) \rangle_{H_i \times W_i \times c_{in}} \) for a fixed input image \( x_0 \) (shape: 32 \times 32 \times 3). The hypersurfaces returned from RM2 of Algorithm 1 are \( \langle H^s_{s,i}(z(x_0)) \rangle_{32 \times 32 \times 3} \), where \( s \in \{0, 1, \ldots,(H_i \times W_i - 1)\}, \, i \in \{0, 1, \ldots,(c_{out} - 1)\} \). Then we verify that there exists a \( z(x_0) \) for our VGG-7, such that,

\[
c_{s,i} = \langle (F(x_0))_{H_i \times W_i \times c_{in} \oplus w_{r_1 \times r_2 \times c_{in}}} \rangle_s = \langle x_0 | H^s_{s,i}(z(x_0)) \rangle,
\]

(14)

where \( \langle \rangle \) is Equation 2 and \( \langle \rangle_s \) denotes taking a real value from \( \langle \rangle \) at offset \( s \). In other words, we verify that a unit (offset by \( s \)) in the convolutional layer with input image \( x \) (left hand side of "\( \approx \" ") is approx-imate to the dot product between the input image \( x \) and the effective hypersurface (right hand side of "\( \approx \" ") \( H^s_{s,i}(z(x)) \) with a proper \( z \), reconstructed by our AdjointBackMap. Verifying \( RM2 \) is equivalent to verifying \( RM1, \, RM3, \, RM4 \).

Verify the existence of \( z \) on a FC layer: Suppose \( c_k \) denotes activational for the class index \( k \) of the FC layer, i.e. \( c_k = F_k(x_0) \). Also Algorithm 1 works on weights of FC layer with \( RM2 \) to generate a hypersurface, \( H^k_{k,i}(z(x_0)) \). Then according to Equation 10 and 11 we verify that,

\[
c_k \approx \langle x_0 | H^k_{k,i}(z(x_0)) \rangle.
\]

(15)

In other words, we verify that VGG-7’s activation value for the \( k \)th neuron through its FC layer with input image \( x \) is equivalent to the dot product between that input image and the effective hypersurface \( H^k_{k,i}(z(x)) \).

Experiment: We validate the above on a 10k test set of CIFAR10 with \( z(x) = \frac{x}{x} \). We calculate the relative errors between feature map neurons (or output neurons of the FC layer), \( f_i \), and the approximation \( p_i \) based on the dot product (equation 14 or 15), as

\[
e_i = \frac{p_i - f_i}{f_i} \neq 0,
\]

(16)

where \( i \in \{0, 1, \ldots,(c_{out} - 1)\} \) (or \( i \in \{0, 1, \ldots, 9\} \) denotes \( i \)th out-channel feature map (or \( i \)th unit value of FC layer before ReLU6); \( f_i \neq 0 \) substitutes all zeros inside \( f_i \), with the smallest positive number of the single-precision floating-point (32-bit float, FP32 for short) data type to avoid any divide by zero exception.

Results: We illustrate six statistical histograms of relative errors for Conv1 to Conv5 and FC layer of VGG-7 as figure 7 (elaboration of Figure 3). Each sample contributes six sets of relative errors and each set has quantity of entries (of \( e_i \)) listed in the 3rd column of table 1. We collect relative errors calculated from different test samples among these six sets and then count the statistics on each of them. Summarily, histograms verify that \( \exists z, \, z = \frac{x}{x} \), such that \( H_i^d(z(x)) \) achieves the approximation (Equation 5) with over 99.99% of the relative errors being less than 0.01 in both convolutional layer and FC layer.

Actually, these relative errors are negligible when considering the inaccuracy results from FP32 data type.

Thus we conclude our model satisfies the two necessary conditions with \( z(x) = \frac{x}{x} \).

Visualization of RM4 Visualizing \( H^d(z(x_0)) \) from RM4 is the most challenging task. The challenge mainly comes from resolution limitation because we decompose every kernel on every stride shift for AdjointBackMap reconstruction that generates large quantities of effective hypersurfaces for visualization. In detail, AdjointBackMap on the Conv1 layer could produce a series of 32 \times 32 \times 3 shaped \( H^d_{s,j,i}(z(x_0)) \) where \( s \in \{0, 1, \ldots, 1023\}, \, j \in \{0, 1, \ldots, 23\}, \, i \in \{0, 1, \ldots, 23\} \) (Equation 6). If we choose to illustrate them together, the resolution would reach 24,576 \times 24,576 \times 3, which pushes the plot DPI over 25k. So we figure all \( s \) together in one picture for an in and out channel pair \( (j,i) \), named \( H^d_{s,j,i}(z(x_0)) \). And we transpose it by \[3, 0, 4, 1, 2] \, then reshape it to 1024 \times 1024 \times 3 as preprocessing, i.e.,

\[
H^d_{s,j,i}(z(x_0)) = (\text{reshape})(H^d_{s,j,i}(z(x_0)) | s \in S)^T.
\]

(17)

where \( S \) denotes the range of stride offsets. We normalize \( H^d_{s,j,i}(z(x_0)) \) by max absolute values along input axes of 32 \times 32 \times 3. Then we take the absolute value on
$H_{s,j,i}^{Adj}(z(x_0))$ for visualization. We set the plot DPI with 800 to clearly depict each pixel, which keeps picture size small at the same time. Similar to these, Table 2 list dimension of $H_{s,j,i}^{Adj}(z(x_0))$ for different layers and different RMs.

Results: Two typical images are picked in this experiment. For every input image, we illustrate two typical effective hypersurfaces for Conv1 to Conv5 layer. Results are illustrated in figure 8 and 9.

Visualization of RM3 Dimension of $H_{s,j,i}^{Adj}(z(x_0))$ by $RM_3$ is reduced as merging happens along strides. Therefore, we figure $\{H_{s,j,i}^{Adj}(z(x_0)) \mid j \in \{0, 1, \ldots, (c_{\text{out}}-1)\}, i \in \{0, 1, \ldots, (c_{\text{in}}-1)\}\}$ (Equation 7) of a layer together, name as $H_{s,j,i}^{Adj}(z(x_0))$. That is,

$$H_{s,j,i}^{Adj}(z(x_0)) = (\text{reshape})(\{H_{s,j,i}^{Adj}(z(x_0))\})^T. \ (18)$$

We use the same techniques, transpose, reshape, normalization and absolute value, as $RM_4$ for visualization. Every layer has only one picture.

Results: We still use those two images from $RM_4$. Results are illustrated in figure 10 and 11.

Visualization of RM2 $RM_2$ merges in-channel-wise while keeping the stride separated to reconstruct hypersurfaces $\{H_{s,j,i}^{Adj}(z(x_0)) \mid s \in S, i \in \{0, 1, \ldots, (c_{\text{out}}-1)\}\}$ (Equation 6). As we mentioned, $H_{s,j,i}^{Adj}(z(x_0))$ from a layer represents an effective hypersurface responsible for a unit in out-channel feature map. Similar to $RM_3$, resolution restriction forces to figure $H_{s,j,i}^{Adj}(z(x_0))$ with all stride offset $s$ together in one picture for an out channel $i$, name $H_{s,j,i}^{Adj}(z(x_0))$. That is,

$$H_{s,j,i}^{Adj}(z(x_0)) = (\text{reshape})(\{H_{s,j,i}^{Adj}(z(x_0)) \mid s \in S\})^T. \ (19)$$

Therefore quantity of pictures for a convolutional layer is equal to quantity of the out channels. We apply the same visualization techniques as $RM_4$.

Results: We still use those two images from $RM_4$. Results are illustrated at figure 12 and 13.

Visualization of RM1 $RM_1$ merges both in-channel-wise and stride-wise to generate hypersurfaces $\{H_{s,j,i}^{Adj}(z(x_0)) \mid i \in \{0, 1, \ldots, (c_{\text{out}}-1)\}\}$ (Equation 9). A $H_{s,j,i}^{Adj}(z(x_0))$ on a layer represents an effective hypersurface for a summation of that out-channel feature map.

Results: We still use those two images from $RM_4$. Results are illustrated at figure 14 and 15.

Visualization of $RM_0$ Two images from the same class are illustrated in Figure 16. One more image (different class) is illustrated in figure 16.

Figures and Tables

![Visualization of RM3](image1)

Figure 7: Elaboration of Figure 4. Histogram of relative errors between $F_i(x)$ and $\langle x \mid H_{s,j,i}^{Adj}(\frac{z(x)}{x}) \rangle$ for $i \in \{0, 1, \ldots, (c_{\text{out}}-1)\}$, by Conv1~5 layers and FC layer before ReLU6). 10k test samples. The x-axis is the error, and the y-axis is the frequency. According to the 3rd column of table 1 a sample generates six sets of relative errors with respect to six layers. Here one figure represents statistics of one set collected from the 10k samples. Therefore (a) to (e) has quantities of relative errors: $24.576 \times 10^k = 245.76m$, $12.288 \times 10^k = 122.88m$, $4.096 \times 10^k = 40.96m$, $40.96m$, $100k$, respectively. These quantities have been numerically verified and printed to figure titles ("Total:"). (a) to (f) reports percentage of relative errors (equation 16 less than 1%): 99.9992%, 99.9978%, 99.9987%, 99.9952%, 99.9995%, 99.997%, respectively.

![Visualization of RM2](image2)

![Visualization of RM1](image3)

![Visualization of $RM_0$](image4)
Table 1: Parameters In Our VGG-7.

| Layer     | Parameters | Out-channel Feature Maps |
|-----------|------------|--------------------------|
| Conv0     | 3 x 3 x 3 x 24 | N/S                      |
| ReLU0     | N/A        | N/S                      |
| Conv1     | 3 x 3 x 24 x 24 | 32 x 32 x 24 (= 24,576) |
| ReLU1     | N/A        | N/S                      |
| Avg-pool-by-2 | N/A     | N/S                      |
| Conv2     | 3 x 3 x 24 x 48 | 16 x 16 x 48 (= 12,288) |
| ReLU2     | N/A        | N/S                      |
| Conv3     | 3 x 3 x 48 x 48 | 16 x 16 x 48 (= 12,288) |
| ReLU3     | N/A        | N/S                      |
| Avg-pool-by-2 | N/A     | N/S                      |
| Conv4     | 3 x 3 x 48 x 64 | 8 x 8 x 64 (= 4,096)     |
| ReLU4     | N/A        | N/S                      |
| Conv5     | 3 x 3 x 64 x 64 | 8 x 8 x 64 (= 4,096)     |
| ReLU5     | N/A        | N/S                      |
| Global-pool | N/A     | N/S                      |
| FC        | [64, 10]   | 10                       |
| ReLU6     | N/A        | N/S                      |

Avg-pool-by-2 denotes average pooling with both window and strides are 2; N/A denotes no learnable parameters; N/S denotes it is not necessary for our method.

Table 2: Dimensions of $H^{Adj\{z(x)\}}$ With Different RMs On Our VGG-7.

| Layer     | $RM_4$ | $RM_3$ | $RM_2$ | $RM_1$ | $RM_0$ |
|-----------|---------|---------|---------|---------|---------|
| Conv0     | N/A     | N/A     | N/A     | N/A     | N/A     |
| Conv1     | $d_{in} \times 32 \times 32 \times 24 \times 24$ | $d_{in} \times 24 \times 24$ | $d_{in} \times 32 \times 32 \times 24$ | $d_{in} \times 24$ | N/A     |
| Conv2     | $d_{in} \times 16 \times 16 \times 24 \times 48$ | $d_{in} \times 24 \times 48$ | $d_{in} \times 16 \times 16 \times 48$ | $d_{in} \times 48$ | N/A     |
| Conv3     | $d_{in} \times 16 \times 16 \times 48 \times 48$ | $d_{in} \times 48 \times 48$ | $d_{in} \times 16 \times 16 \times 48$ | $d_{in} \times 48$ | N/A     |
| Conv4     | $d_{in} \times 8 \times 8 \times 48 \times 64$ | $d_{in} \times 48 \times 64$ | $d_{in} \times 8 \times 8 \times 64$ | $d_{in} \times 64$ | N/A     |
| Conv5     | $d_{in} \times 8 \times 8 \times 64 \times 64$ | $d_{in} \times 64 \times 64$ | $d_{in} \times 8 \times 8 \times 64$ | $d_{in} \times 64$ | N/A     |
| FC        | N/A     | N/A     | N/A     | N/A     | $d_{in} \times 10$ |

$d_{in}$ denotes 32 x 32 x 3; N/A denotes a layer where our AdjointBackMap is not applicable.
Class index $k$ / Method | Input $(x_0 + Advr)$, FC output (Before ReLU6) | $(x_0 + Advr, H_{k}^{Adj}(z_0))$ | $(x_0 + Advr, H_{k}^{Adj}(z_1))$
---|---|---|---
0 (airplane) | 6.2982497 | 6.2982607 | 5.5662646
1 (automobile) | 7.7942696 | 7.7942758 | 6.3915267
2 (bird) | 7.2347097 | 7.2346945 | 6.7430825
3 (cat) | 7.806831 | 7.806829 | 7.859078
4 (deer) | 5.922823 | 5.922816 | 5.419629
5 (dog) | **7.829193** | **7.8291893** | 7.684642
6 (frog) | 7.8009434 | 7.80094 | 7.7138715
7 (horse) | 6.572267 | 6.572259 | 6.3699017
8 (ship) | 0.13836712 | 0.13837112 | −0.3877122
9 (truck) | 7.65885 | 7.658846 | 7.147852

Table 3: Experiment A. Verify The Hypersurface For Adding Adversarial Noise (equation 15, Equation 13).

$z_0 = \frac{x_0 + Advr}{k}$, $z_1 = \frac{x_0}{k}$.

The third column has significantly smaller errors than the fourth column. Also, the fourth column hits an inaccurate class index (largest value is "cat" instead of "dog"). Therefore it verifies that $H_{k}^{Adj}(z_0)$ is the effective decision boundary for the $k^{th}$ FC prediction. As we’ve verified that $H_{k}^{Adj}$ is an accurate model (figure 7) for a decision boundary reconstruction, this experiment further emphasizes that a pure accurate model is not enough. Only the accurate model working on an accurate input can the decision boundary be precisely reconstructed. This is the point that distinguishes our AdjointBackMap from others (Related Work section of the main paper).
Conv1: (a) (3, 2) (b) (10, 12) Conv2: (c) (1, 5) (d) (18, 10) Conv3: (e) (4, 6) (f) (41, 5) Conv4: (g) (15, 5) (h) (30, 3) Conv5: (i) (11, 20) (j) (37, 20) (x₀) Frog

Figure 9: Visualization of RM₄. Typical $H^{Adj}_{i,j} (\frac{x₀}{8})$ patterns (a to j) mapped from Conv1 to Conv5 layer by $RM_4$ with a frog input. Details are the same as figure 8.

Figure 10: Visualization of RM₃. $H^{Adj}_{i,j} (\frac{x₀}{8})$ patterns mapped from Conv1 to Conv5 layer (Equation 7) by $RM_3$ (equation 18) with a ship input. Each figure has its shape (quantity of stride moves): 1024 × 1024 × 3(32 × 32), 512 × 512 × 3(16 × 16), 256 × 256 × 3(8 × 8), 256 × 256 × 3(8 × 8), respectively. $H^{Adj}_{i,j} (\frac{x₀}{8})$ on a layer represents a decision hypersurface for the pixel of $j^{th}$ out-channel feature map on that layer. Similar to figure 8, a higher-layer kernel has a larger effective receptive field from the input space perspective; Local sparsity (black-out area) will increase when layer index decreases.

Figure 11: Visualization of RM₃. $H^{Adj}_{i,j} (\frac{x₀}{8})$ patterns mapped from Conv1 to Conv5 layer by $RM_3$ with a frog input. Details are the same as figure 10.

Figure 12: Visualization of RM₂. Typical $H^{Adj}_{i,j} (\frac{x₀}{8})$ patterns mapped from Conv1 to Conv5 layer (Equation 8) by $RM_2$ (equation 19) with a ship input. $i$ denotes the out-channel index where kernels are mapped. Each figure has its shape (quantity of stride moves): 1024 × 1024 × 3(32 × 32), 512 × 512 × 3(16 × 16), 256 × 256 × 3(8 × 8), 256 × 256 × 3(8 × 8), respectively. $H^{Adj}_{i,j} (\frac{x₀}{8})$ on a layer represents a decision hypersurface for the pixel of $j^{th}$ out-channel feature map on that layer. Similar to figure 8, a higher-layer kernel has a larger effective receptive field from the input space perspective; Local sparsity (black-out area) will increase when layer index decreases.

Figure 13: Visualization of RM₂. Typical $H^{Adj}_{i,j} (\frac{x₀}{8})$ patterns mapped from Conv1 to Conv5 layer by $RM_2$ with a frog input. Details are the same as figure 12.
Figure 14: Visualization of $\mathbf{R}M_1$. $H_{x_0}^{Adj}(\frac{x_{0}}{\alpha})$ patterns mapped from Conv1 to Conv5 layer (Equation 9) by $RM_1$ with a ship input. Quantity of subfigures in a plot is equal to the quantity of out channels.

Figure 15: Visualization of $\mathbf{R}M_1$. $H_{x_0}^{Adj}(\frac{x_{0}}{\alpha})$ patterns mapped from Conv1 to Conv5 layer by $RM_1$ with a frog input. Details are the same as figure 14.

Figure 16: Visualization by $\mathbf{R}M_0$ (One more example (frog) to supplement Figure 3). $H_{x_0}^{Adj}(\frac{x_{0}}{\alpha})$ patterns mapped from the FC layer (Equation 10) by $RM_0$ (e), and its corresponding input (f, different class from Figure 3(b) and (c)). Number of subfigures in (e) is equal to the number of classes.

Figure 17: Experiment $B_1$. We prepare 9 adversarial noises for the ship image (Figure 3(b) or figure 8($x_0$)). $S_{B_1} = \{Advr_i \mid i \in \{0, 1, ..., 9\}, Advr_8 = \theta\}$. An element $Advr_i \in S_{B_1}$ can fool our VGG-7 to the class index $i$. $Advr_8 = \theta$ denotes no (zero) adversarial noise prepared. Patterns of $H_{k=8}^{Adj}(\frac{x_0 + Advr_i}{\alpha})$ by $RM_0$ are sequentially subfigured in first two rows of (a) and rest two rows show its corresponding $Advr_i$ magnified by 200. These 9 adversarial noises are visually indistinguishable to humans and their euclidean distances are small, $\{||Advr_i|| \mid i \in \{0, 1, ..., 9\}\} = \{0.055, 0.055, 0.11, 0.13, 0.12, 0.13, 0.11, 0.15, 0.16\}$, compared with $||x_0|| = 9.448$. Different from the Experiment $B_2$ (Figure 6) which explores causality with factor analysis (misclassified with adversarial examples or correctly classified with Gaussian noises), we project the metric relationship among 10 effective hypersurfaces with tSNE for analysis (b). Visualization illustrates that differences exist in the effective boundaries ((a) and (c) in Figure 3) for two images from the same class. Now (a) and (b) show effective boundaries for the same image contaminated by indistinguishable noises are still far apart each other. Thus, it suggests CNN’s decision is brittle.