Salient deconvolutional networks

Aravindh Mahendran and Andrea Vedaldi

Department of Engineering Science,
University of Oxford
{aravindh,andrea}@robots.ox.ac.uk

Abstract. Deconvolution is a popular method for visualizing deep convolutional neural networks; however, due to their heuristic nature, the meaning of deconvolutional visualizations is not entirely clear. In this paper, we introduce a family of reversed networks that generalizes and relates deconvolution, backpropagation and network saliency. We use this construction to thoroughly investigate and compare these methods in terms of quality and meaning of the produced images, and of what architectural choices are important in determining these properties. We also show an application of these generalized deconvolutional networks to weakly-supervised foreground object segmentation.

Keywords: DeConvNets, Deep Convolutional Neural Networks, Saliency, Segmentation

1 Introduction

Despite the success of modern Convolutional Neural Networks (CNNs), there is a limited understanding of *how* these complex black-box models achieve their performance. Methods such as *deconvolutional networks* (DeConvNets) have been proposed to *visualize* image patterns that strongly activate any given neuron in a CNN [25] and therefore shed some light on the CNN structure. However, the DeConvNet construction is partially heuristic and so are the corresponding visualizations. Simonyan *et al.* [16] noted similarities with their *network saliency* method which partially explains DeConvNets, but this interpretation remains incomplete.

This paper carries a novel and systematic analysis of DeConvNets and closely related visualization methods such as network saliency. Our first contribution is to extend DeConvNet to a general method for architecture reversal and visualization. In this construction, the reversed layers use selected information extracted by the forward network, which we call *bottleneck information* (Sect. 2). We show that backpropagation is a special case of this construction which yields a reversed architecture, SaliNet, equivalent to the network saliency method of Simonyan *et al.* (Sect. 2.1). We also show that the *only* difference between DeConvNet and SaliNet is a seemingly innocuous change in the reversal of Rectified Linear Units (ReLU; Sect. 2.2). However, this change has a *very significant* effect on the results: the SaliNet response is well localized but lacks structure, whereas the DeConvNet response accurately reproduces the image boundaries and object shapes, but is less localized (Fig. 1). We also show that the two methods can be combined in order to simultaneously obtain structure and localization (DeSaliNet). De-

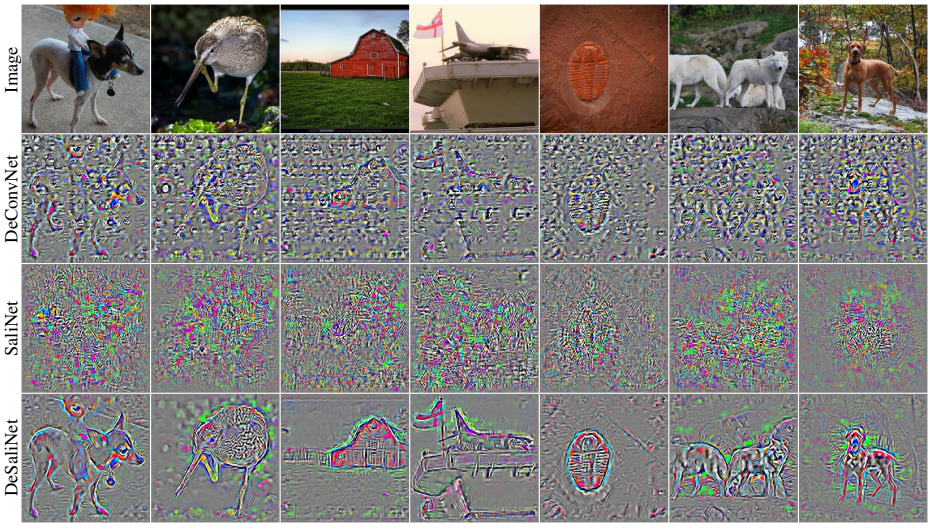

Fig. 1: From top row to bottom: Original image, DeConvNet, SaliNet and our DeSaliNet visualizations from the fc8 layer in AlexNet (just before the softmax operation). The maximally active neuron is visualized in each case. DeSaliNet results in crisper visualizations. They suppress the background while preserving edge information. Best viewed on screen.

SaliNet is also similar to results recently obtained by [17].

We then move to the important question of whether deconvolutional architectures are useful for visualizing neurons. Our answer is partially negative, as we find that the output of reversed architectures is mainly determined by the bottleneck information rather than by which neuron is selected for visualization (Sect. 3.3). In the case of SaliNet and DeSaliNet, we confirm that the output is selective of any recognizable foreground object in the image, but the class of the selected object cannot be specified by manipulating class-specific neurons.

Having established the dominance of bottleneck information, we draw an analogy between that and phase information in the Fourier transform (Sect. 3.4) and show the importance of polarity information in reversed architectures.

Finally, we quantitatively test the ability of SaliNet and DeSaliNet to identify generic foreground objects in images (Sect. 3.5). Combined with GrabCut, we achieve near state-of-the-art segmentation results on the ImageNet segmentation task of [4], while using off-the-shelf CNNs pretrained from a *largely disjoint subset* of ImageNet and with only image-level supervision.

Related work. DeConvNets were originally proposed as a method for unsupervised feature learning [26,27] and later applied to visualization [25]. There are several CNN visualizations alternative to DeCovnNets. Some recent ones such as [24] build on the idea of *natural* (regularized) pre-images introduced in [13], which in turn are based on prior contributions that applied pre-images to representations such as HOG [21],

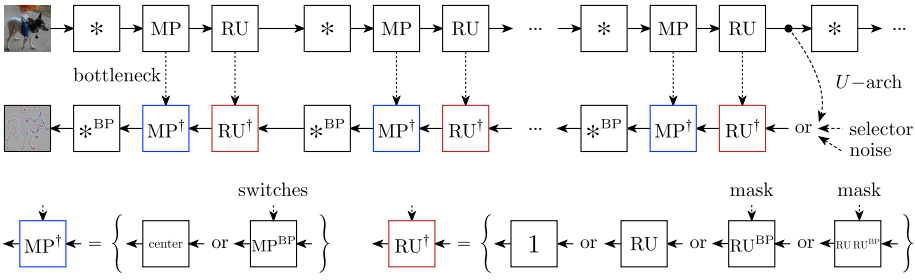

Fig. 2: The top row shows a typical CNN obtained by repeating short chains of convolution ($*$), max pooling (MP) and ReLU (RU) operators. The middle row shows a generic "deconvolution" architecture, in which information flows backward to the image using the convolution transpose $*^{\mathrm{BP}}$ operator. Different variants are obtained by: (i) choosing a different input (U-architecture, feature selector, or noise), (ii) choosing a variant of backward ReLU $\mathrm{RU}^{\dagger}$ (1, ReLU, ReLU backpropagation, or hybrid), and (iii) choosing a variant of backward max pooling $\mathrm{MP}^{\dagger}$ (unpool to centre or MP backpropagation). This schema generalizes DeConvNets [25].

SIFT [22], BoVW [2,7], as well as early neural networks [23,10,9,12,6,20]. A related line of work [1] is to learn a second neural network to act as the inverse of the original one. Several authors characterize properties of CNNs and other models by generating images that confuse them [19,18,14]. Our DeSaliNet architecture is also similar to the work of [17].

Recently, DeConvNets have also been proposed as a tool for semantic image segmentation; for example,[15,5] interpolate and refine the output of a fully-convolutional network [11] using a deconvolutional architecture. In this paper, inspired by [16], we apply reversed architectures for foreground object segmentation, although as a by-product of visualization and in a weakly-supervised transfer-learning setting rather than as a specialized segmentation method.

2 A family of deconvolutional architectures

Given an image $\mathbf{x} \in \mathcal{X}$, a deep CNN extracts a feature vector or representation

$$\phi : \mathbf{x} \mapsto \phi_L \circ \cdots \circ \phi_1(\mathbf{x}) \tag{1}$$

using a sequence of L linear and non-linear layers ϕ_i (Fig. 2.top). Typical layers include convolution, ReLU, max pooling, and local contrast normalization.

The goal is to associate to ϕ a corresponding architecture $\phi^{\dagger}$ that reverses in some sense the computations and produces an image as output. While such reversed architectures have several uses, here we focus on the problem of *visualizing* deep networks: by looking at the images produced by $\phi^{\dagger}$, we hope to gain some insights about the forward network ϕ. This method was popularized by the work of Zeiler and Fergus in [25], where a particular construction called DeConvNet was shown to produce surprisingly crisp renderings of neural activations. However, given the heuristic nature of some choices in DeConvNets, it is difficult to precisely characterize the meaning of

these results.

In order to explore this point, we consider here a generalization of the DeConvNet construction. To this end, each layer ϕ_i is associated with a corresponding layer $\phi_i^\dagger$ that reverses input $\mathbf{x}$ and output $\mathbf{y}$ (Fig. 2.middle row). We also allow the reverse layer to be influenced by auxiliary information $\mathbf{r}$ computed by the forward layer. For instance, in DeConvNet, the reverse max pooling layer requires the "setting of the pooling switches" computed in the forward pass. Thus a layer ϕ_i and its reverse $\phi_i^\dagger$ are maps:

$$\text{forward } \phi_i : \mathbf{x} \mapsto (\mathbf{y}, \mathbf{r}), \qquad \text{reversed } \phi_i^\dagger : (\hat{\mathbf{y}}, \mathbf{r}) \mapsto \hat{\mathbf{x}}. \tag{2}$$

The $\hat{\cdot}$ symbol emphasizes that, in the backward direction, the tensors $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ have the same shape as $\mathbf{x}$ and $\mathbf{y}$ in the forward pass, but different values.

Since the auxiliary information $\mathbf{r}$ is a function of the input $\mathbf{r} = \pi(\mathbf{x})$, one can always let $\mathbf{r} = \mathbf{x}$ without loss of generality; however, the interesting case is when the auxiliary information is limited and $\mathbf{r}$ is an *information bottleneck*. For example, the pooling switches $\mathbf{r}$ in DeConvNet contain much less information than the input data $\mathbf{x}$. In Fig. 2 these bottlenecks are denoted by dotted arrows.

The question then is how can we build reverse layers $\phi^\dagger$? Next, we show that back-propagation provides a general construction for reverse layers, which only in some cases corresponds to the choice in DeConvNet.

2.1 SaliNet: network saliency as a deconvolutional architecture

The *network saliency* method of Simonyan *et al.* [16] characterizes which pixels of an image are most responsible for the output of a CNN. Given an image $\mathbf{x}_0$ and a network $\phi(\mathbf{x}_0)$, saliency is obtained as the derivative of the (projected) CNN output $S(\phi, \mathbf{x}_0, \mathbf{p})$ with respect to the image:

$$S(\phi, \mathbf{x}_0, \mathbf{p}) = \frac{\partial}{\partial \mathbf{x}} \langle \mathbf{p}, \phi(\mathbf{x}) \rangle \Big|_{\mathbf{x}=\mathbf{x}_0}. \tag{3}$$

Since the CNN output $\phi(\mathbf{x})$ is in general a vector or tensor, the latter is transformed into a scalar by linear projection onto a constant tensor $\mathbf{p}$ before the derivative is computed. In practice, $\mathbf{p}$ is usually a one-hot tensor that selects an individual neuron in the output. In this case, the value of a pixel in the saliency map $S(\phi, \mathbf{x}_0, \mathbf{p})$ answers the question: "how much would the neuron response $\langle \mathbf{p}, \phi(\mathbf{x}_0) \rangle$ change by slightly perturbing the value of that pixel in the image $\mathbf{x}_0$?"

Saliency is computed from (1) and (3) using the chain rule:

$$\operatorname{vec} S(\phi, \mathbf{x}_0, \mathbf{p}) = \operatorname{vec} \mathbf{p}^\top \times \frac{\partial \operatorname{vec} \phi_L}{\partial \operatorname{vec} \mathbf{x}_L^\top} \times \cdots \times \frac{\partial \operatorname{vec} \phi_1}{\partial \operatorname{vec} \mathbf{x}_0^\top}. \tag{4}$$

Here the vec operator stacks tensors into vectors and allows us to use a simple matrix

notation for the derivatives.

The *Back Propagation* (BP) algorithm is the same as computing the products (4) from left to right; this reduces to a chain of derivatives in the form of (3), one for each layer, where $\mathbf{p}$ is replaced with the derivative $\hat{\mathbf{y}}$ obtained from the layer above. In this manner, BP provides a general way to define a reverse of any layer ϕ_i:

$$\phi_i : \mathbf{x} \mapsto \mathbf{y} \quad \text{BP-reversed becomes} \quad \phi_i^{\text{BP}} : (\mathbf{x}, \hat{\mathbf{y}}) \mapsto \frac{\partial}{\partial \mathbf{x}} \langle \hat{\mathbf{y}}, \phi_i(\mathbf{x}) \rangle. \tag{5}$$

We denote the BP-reversed of a layer with the symbol ϕ_i^{BP}. Any CNN toolbox can compute BP-reversed for any layer as it contains code for back-propagation. Note also that the BP-reversed layer is *a linear map* in the argument $\hat{\mathbf{y}}$, even if the forward layer is not linear in $\mathbf{x}$. In this manner, one can compute backpropagation, and therefore the saliency map $S(\phi, \mathbf{x}_0, \mathbf{p})$ of [16], by using a "deconvolutional" architecture of the type of Fig. 2, where layers are reversed using the BP equation (5). We call this architecture *SaliNet*.

The BP-reversed layer ϕ_i^{BP} takes as input both $\mathbf{x}$ and $\hat{\mathbf{y}}$, whereas from our discussion above we would like to replace $\mathbf{x}$ with a bottleneck $\mathbf{r}$. Formally, using the definition (2), we rewrite the BP-reversed layer $\phi_i^{\text{BP}}(\mathbf{x}, \hat{\mathbf{y}})$ as $\phi_i^{\dagger}(\hat{\mathbf{y}}, \mathbf{r})$ where $\mathbf{r} = \pi(\mathbf{x})$ projects the data $\mathbf{x}$ onto the smallest possible bottleneck. Note that this does not change the meaning of a BP-reversed layer, but it does characterizes how much auxiliary information it requires. The latter is easy to find in an abstract sense,[1] but it is much more instructive to derive it for concrete layer types, which we do below for common layers.

Affine layers. A *fully connected layer* ϕ_{fc} simply multiplies the data $\mathbf{x}$ by a matrix A and adds a bias b. Given that the data $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ is a 3D tensor of height H and width W and C feature channels, we use the vec operator to write this in terms of matrices[2] as $\phi_{\text{fc}} : \operatorname{vec} \mathbf{y} = A \operatorname{vec} \mathbf{x} + b$. *Linear convolution* ϕ_* can conveniently be defined in the same way, by replacing matrix A with a matrix $\rho(F)$ constructed by "sliding" a bank of filters F, giving $\phi_* : \operatorname{vec} \mathbf{y} = \rho(F) \operatorname{vec} \mathbf{x} + b$. Using (5), the BP-reversed layers are obtained by transposing the respective matrices:

$$\phi_{\text{fc}}^{\text{BP}} : \operatorname{vec} \hat{\mathbf{x}} = A^\top \operatorname{vec} \hat{\mathbf{y}}, \qquad \phi_*^{\text{BP}} : \operatorname{vec} \hat{\mathbf{x}} = \rho(F)^\top \operatorname{vec} \hat{\mathbf{y}}. \tag{6}$$

The layer ϕ_*^{BP} is often called *deconvolution* and gives the name to DeConvNets.

Note that the computation of these layers does not require any information from the forward pass, so the bottleneck $\mathbf{r}$ is empty. This is due to linearity and explains why in Fig. 2 there are no dashed arrows connecting convolutional layers.

Rectified linear Unit (ReLU or RU). ReLU and its BP-reversed layer are given by

$$\phi_{\text{RU}}(\mathbf{x}) = \max\{\mathbf{x}, 0\}, \qquad \phi_{\text{RU}}^{\text{BP}}(\mathbf{x}, \hat{\mathbf{y}}) = \phi_{\text{RU}}^{\dagger}(\hat{\mathbf{y}}, \mathbf{r}) = \hat{\mathbf{y}} \odot \mathbf{r}, \quad \mathbf{r} = [\mathbf{x} > 0], \tag{7}$$

where max is computed element-wise, $\odot$ is the element-wise product, and $[\mathbf{x} > 0]$ is a

[1] Let $\mathbf{x}' \sim \mathbf{x}''$ be equivalent whenever functions $\phi_i^{\text{BP}}(\mathbf{x}', \cdot) = \phi_i^{\text{BP}}(\mathbf{x}'', \cdot)$ are the same. It is easy to check that this defines an equivalence relation. Then the smallest possible bottleneck $\pi : \mathbf{x} \mapsto \mathbf{r} \in \mathcal{X}/\sim$ projects $\mathbf{x}$ into its equivalence class.

[2] This is slightly more general than usual as it specifies a different bias for each output dimension instead for each output feature channel.

mask (binary tensor) with a 1 for every positive element of $\mathbf{x}$ and 0 otherwise. Hence the *bottleneck information for ReLU is the mask*. Note that $\phi^{\text{BP}}_{\text{RU}}(\mathbf{x}, \hat{\mathbf{y}})$ is not the reversal used by DeConvNets [25,16] and this choice changes the output significantly.

Max Pooling (MP). Let x_{uc} be the element of tensor $\mathbf{x}$ at spatial location $u \in \Omega$ and feature channel c. MP is obtained by computing the maximum of x_{vc} over a small spatial neighbourhood $v \in N(u) \subset \Omega$ corresponding to u:

$$[\phi_{\text{MP}}(\mathbf{x})]_{uc} = \max_{v \in N(u)} x_{vc} = x_{s(u|c,\mathbf{x}),c} \quad \text{where} \quad s(u|c,\mathbf{x}) = \operatorname*{argmax}_{v \in N(u)} x_{vc}. \tag{8}$$

Here $v = s(u|\mathbf{x}, c)$ tells which element x_{vc} of the input is associated by max to each element y_{uc} of the output and is informally called a *setting of the pooling switches*. A short derivation from (5) shows that the BP-reversed is given by

$$[\phi^{\text{BP}}_{\text{MP}}(\mathbf{x}, \hat{\mathbf{y}})]_{vc} = [\phi^{\dagger}_{\text{MP}}(\hat{\mathbf{y}}, \mathbf{r})]_{vc} = \sum_{u \in s^{-1}(v|c,\mathbf{x})} \hat{y}_{uc}, \qquad \mathbf{r} = s(\cdot|\cdot, \mathbf{x}). \tag{9}$$

Hence the *bottleneck information for MP is the setting of the pooling switches*.

2.2 Deconvolutional architectures

BP-reversal is only one way of defining reversed layers in a deconvolutional architecture. Here, we consider three variations. The **first variation** is whether the reversed max-pooling layers are the BP-reversed ones $\phi^{\text{BP}}_{\text{MP}}$, with pooling switches as bottleneck, or whether they simply unpool to the center of each neighborhood, with empty bottleneck. The **second variation** is whether the reversed ReLU units are the BP-reversed ones $\phi^{\text{BP}}_{\text{RU}}$, with the ReLU mask as bottleneck, or whether they are simply replaced with the identity function 1, with empty bottleneck. The **third variation** is whether the reversed ReLU units are, as in DeConvNets, also composed with a second ReLU. We will see that, while this choice seems arbitrary, it has a very strong impact on the results as it preserves the polarity of neural activations. Overall, we obtain eight combinations, summarized in Fig. 3, including three notable architectures: DeConvNet, SaliNet, and the hybrid DeSaliNet. Note that only SaliNet has an immediate interpretation, which is computing the derivative of the forward network.

Affine layers, max pooling, and ReLU cover all the layer types needed to reverse architectures such as VGG-VD, GoogLeNet, Inception and ResNet.[3] AlexNet includes *local response normalization* (LRN) layers, which in DeConvNet are reversed as the identity. As discussed in the supplementary material, this has little effect on the results.

3 Experiments

Experiments thoroughly investigate the family of deconvolutional architectures identified in Sect. 2. Sect. 3.1 tests eight possible network architectures and identifies DeConvNet, SaliNet, and DeSaliNet as interesting cases for further exploration. Sect. 3.2 compares the architectures in terms of clarity of the generated images. Sect. 3.3 investigates whether visualizations provide useful information about neurons, and Sect. 3.4 looks at the effect of the bottleneck information. Finally, Sect. 3.5 evaluates these tech-

[3] In all cases we deal with the network only till the layer before the softmax

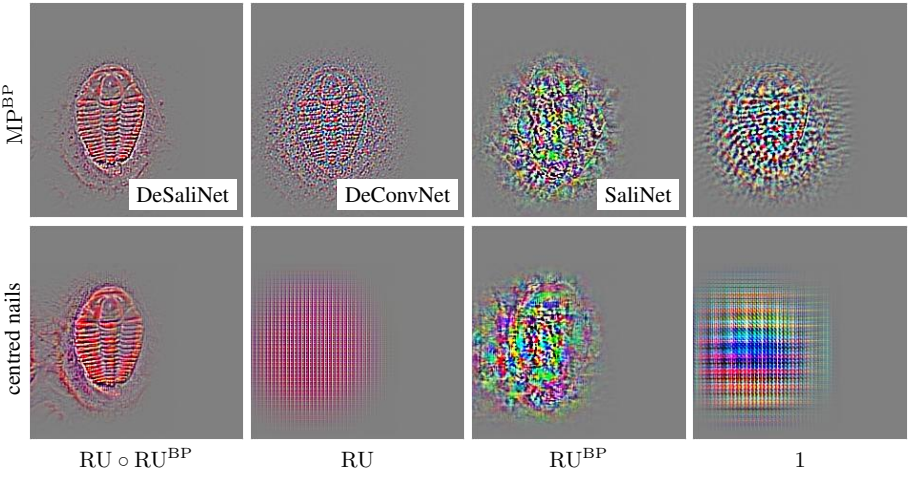

Fig. 3: Visualizations of VGG-16 using the *trilobite* image of Fig. 1 using eight deconvolutional architectures. The architectures are used to visualize the maximally-firing neuron in the *pool5_3* layer and the full output image is shown (localization is mostly due to the finite support of the neuron). From top to bottom we change the $\mathrm{MP}^\dagger$ reverse and from left to right the $\mathrm{RU}^\dagger$ reverse. Here all methods use the identity as the reverse $\mathrm{LRN}^\dagger$.

niques on a practical application: segmentation of foreground objects.

Several experiments are shown here for a few representative images, but many more examples are provided in the supplementary material.[4]

3.1 Overview of deconvolutional architectures

The first experiment compares the eight deconvolutional architectures of Fig. 3. This is done by "reversing" the computations obtained when a network ϕ is evaluated on an image $\mathbf{x}_0$ (in the example, the "trilobite" image of Fig. 1). Here the forward network $\mathbf{y} = \phi(\mathbf{x}_0)$ is AlexNet [8] truncated at the last max-pooling layer (pool5). The input $\mathbf{p}$ to the reversed network $\phi^\dagger(\mathbf{p})$ is the one-hot tensor selecting the pool5 neuron y_{uc} that is maximally excited by $\mathbf{x}_0$.

We can make the following observations. First, as in [16], SaliNet computes a fuzzy saliency map. Likewise, matching the results of [25], the result of DeConvNet has structure, in the sense that object edges are recovered.

Second, we compare the left four deconvolutional architectures to the right ones, which differ by the use of the ReLU units in the backward direction. We note that adding these units is *necessary* in order to recover the image edges. In particular, by modifying SaliNet in this manner, DeSaliNet produces an image with structure.

Third, using pooling switches (top row) slightly improves the clarity of the results

[4] To improve the clarity of visualization images $\mathbf{x} = \phi^\dagger(\mathbf{p})$ in print, their real valued ranges are remapped using the expression $\sigma(\mathbf{x}/(-\log(99)a))$ where a is the 0.5% quantile in $\operatorname{vec} I$.

compared to unpooling to center (bottom row). Even so, we note that the image structure can still be clearly recognized in the bottom-left image, using unpooling to center complemented by the hybrid $\mathrm{RU} \circ \mathrm{RU}^{\mathrm{BP}}$ as reverse ReLU. In fact, this image is arguably crisper than the DeConvNet result. This suggests that, perhaps unexpectedly, the ReLU polarity (captured by RU in the backward direction) is more important that the MP switches. It also shows that the ReLU masks (captured by $\mathrm{RU}^{\mathrm{BP}}$) significantly improve the sharpness of the results.

So far the LRN layers in AlexNet have been reversed using the identity, as in DeConvNet; however, the original saliency method by [16] uses the BP-reversed $\mathrm{LRN}^{\mathrm{BP}}$. In the supplementary material we show that this has a minor impact on the result, with slightly sharper results for the DeConvNet solution. Therefore, in the rest of the manuscript, DeConvNet and DeSaliNet use identity, while SaliNet, in keeping with the original saliency method by [16], uses $\mathrm{LRN}^{\mathrm{BP}}$.

3.2 Generated image quality

A first striking property of DeSaliNet is the clarity of resulting visualizations compared to the other architectures (e.g. Fig. 1, 3, 4 6). While sharper visualizations than SaliNet are expected given the results in [16], the gap with DeConvNet is somewhat unexpected and particularly strong for deep layers (e.g. Fig 1) and deeper architectures (e.g. Fig. 6). DeConvNet results appear to be less sharp than the ones shown in [25], which could be due to the fact that they used a custom version of AlexNet, whereas we visualize off-the-shelf versions of AlexNet and VGG-VD. Unfortunately, it was not possible obtain a copy of their custom AlexNet to verify this hypothesis.

3.3 Meaning and selectivity of the deconvolutional response

Visualizations obtained using reversed architectures such as DeConvNets are meant to characterize the *selectivity of neurons* by finding which visual patterns cause a neuron to fire strongly. However, we will see here that this interpretation is fragile.

Consider the i-th neuron $[\phi(\mathbf{x})]_i = \langle \mathbf{e}_i, \phi(\mathbf{x}) \rangle$ in the output layer of a (truncated) CNN architecture, where $\mathbf{e}_i$ is an indicator vector. In order to characterize this neuron, Zeiler and Fergus [25] search a large collection of images to find an image $\mathbf{x}^*$ that causes $\phi_i(\mathbf{x}^*)$ to respond strongly. Thus, even before the deconvolutional network $\phi^\dagger(\mathbf{e}_i, \mathbf{r})$ is applied, the image $\mathbf{x}^*$ is already representative of the neuron. The application of $\phi^\dagger$ then *refines* this information by highlighting which regions in $\mathbf{x}^*$ are most responsible for this activation.

While this sounds simple, there is a subtle complication. Note in fact that the deconvolutional architecture $\phi^\dagger(\mathbf{e}_i, \mathbf{r})$ is a function both of the neuron indicator $\mathbf{e}_i$ as well as the bottleneck information $\mathbf{r}$ extracted from the forward pass of $\mathbf{x}^*$ through $\phi(\mathbf{x})$. In the deconvolution process, $\mathbf{e}_i$ is a direct specification of the neuron to be visualized. The other parameter, $\mathbf{r}$, can also be considered a specification of the same neuron, although a fairly indirect one, because it is extracted from an image $\mathbf{x}^*$ that happens to excite the neuron strongly. Then the question is whether the deconvolutional response can be interpreted as a *direct* characterization of a neuron or not. This is answered next.

Lack of neuron selectivity. If the output of $\phi^\dagger(\mathbf{e}_i, \mathbf{r})$ is a direct characterization of the

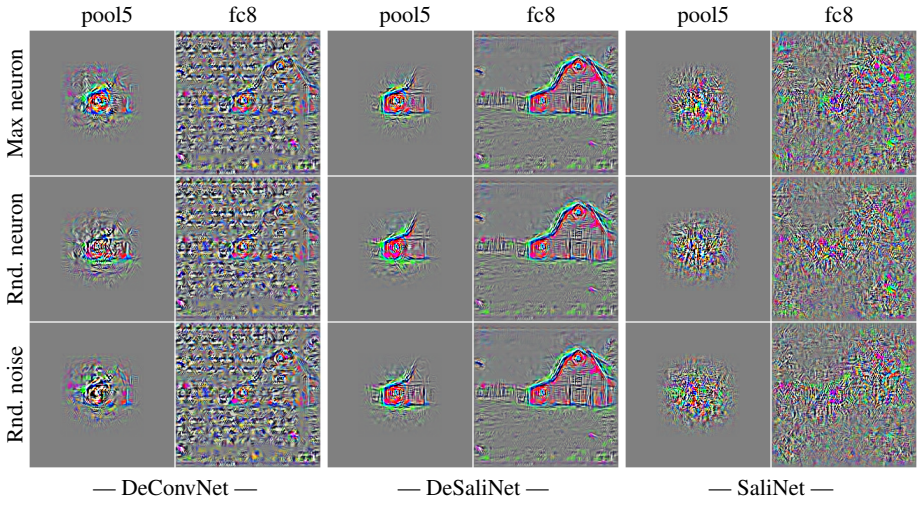

Fig. 4: *Lack of neuron selectivity.* The bottleneck information $\mathbf{r}$ is fixed to the one computed during the forward pass $\phi(\mathbf{x})$ through AlexNet and the output of $\phi^\dagger(\mathbf{e}, \mathbf{r})$ is computed by choosing $\mathbf{e}$ as: the most active neuron (top row), a second neuron at random (middle), or as a positive random mixture of all neurons (bottom row). Results barely differ, particularly for the deeper layers. See figure 1 for the original house input image $\mathbf{x}$. Best viewed on screen.

i-th neuron, we would expect the generated image to *meaningfully change* as the input $\mathbf{e}_i$ to the deconvolutional network changes.

In Fig. 4, DeConvNet, DeSaliNet, and SaliNet are used to visualize the responses of different neurons at the center of the image. The reversed function $\phi^\dagger(\mathbf{e}, \mathbf{r})$ is evaluated by keeping $\mathbf{r}$ fixed (as obtained from the forward pass $\phi(\mathbf{x}_0)$) and by replacing $\mathbf{e}$ with either: the indicator vector $\mathbf{e}^*$ of the neuron that has the maximal response, a second random neuron $\mathbf{e}'$ that still generates a non-zero image, and a random non-negative vector $\mathbf{e}$. It can be noted that, particularly in deeper layers, the response changes very little with different choices of $\mathbf{e}$.

A clear difference between images from different depths (e.g. pool5 vs fc8 in Fig. 4 and 6) is the extent of the response, which however corresponds to the neuron support and depends on the architecture and not on the learned network weights or data. This is further confirmed in Fig. 5 by considering a network with random weights. There, it is also shown that renormalizing the image intensities reveals the full neuron support, which is only partially suppressed in the visualization, and in a manner which is architecture-dependent rather than weight or data dependent.

We conclude that the reversed architectures $\phi^\dagger(\mathbf{e}, \mathbf{r})$ are mainly dependent on the bottleneck information $\mathbf{r}$ rather than the neuron selector $\mathbf{e}$. Hence, they provide poor direct characterizations of neurons, particularly of deep ones.

Note that methods such as [16,13,24], which visualize individual neurons by activation maximization, are not affected by this problem. There are two reasons: first, they start from random noise, such that the bottleneck information $\mathbf{r}$ is *not* primed by a

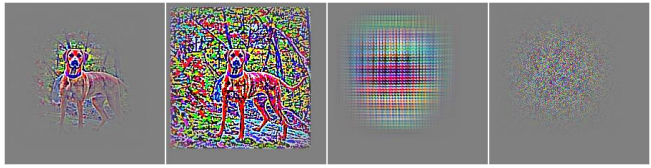

Fig. 5: *Effect of finite neuron support.* From left to right: Visualization from VGG-16 pool5_3 using DeSaliNet; the same result after $\mathbf{x}^{0.1}$ renormalization; visualization without any bottleneck information as in Fig. 3-bottom right; the same visualization without bottleneck information but with randomized filter weights for the $*^{\mathrm{BP}}$ operators. The re-normalization reveals that the true receptive field of pool5 is much larger and that the sides are not discarded but simply weakened in the deconvolution process.

carefully-selected reference image $\mathbf{x}_0$; secondly, they iteratively update the bottleneck information, drifting away from the initial value.

Foreground object selectivity. SaliNet is an equivalent implementation of the network saliency technique of Simonyan et al. [16], which showed that the deepest class-specific neurons (in fc8) in an architecture such as AlexNet are strongly selective for the foreground object in an image. However, in the previous section we have shown the apparently contradictory result that this response depends very weakly on the choosen class-specific neuron.

To clarify this point, in Fig. 1 and Fig. 6 we observe that SaliNet and DeSaliNet *are indeed* selective for the foreground object in the image; however, the information *comes mainly from the bottleneck* $\mathbf{r}$ and not from which specific neuron is selected. In other words, SaliNet and DeSaliNet emphasize whatever foreground object is detected by the network in the forward pass, regardless of which neuron is specified as input to the reversed architecture. The main difference between SaliNet and DeSaliNet, as observed before, is that the latter produces a much more localized and crisp response

Compared to SaliNet and DeSaliNet, DeConvNet fails to produce a clearly selective signal from these very deep neurons, generating a rather uniform response. We conclude that saliency, in the sense of foreground object selectivity, requires not only the max pooling switches (available in all three architectures), but also the ReLU masks (used only by SaliNet and DeSaliNet).

Informativeness of bottleneck. In order to characterize the amount of information contained in the bottleneck, we used the method of [3] to train a network that acts as the inverse of another. However, while the inverse network of [3] operates only from the output of the direct model, here we modified it by using different amounts of bottleneck information as well. The reconstruction error of these "informed" inverse networks illustrates importance of the bottleneck information. We found that inverting with the knowledge of the ReLU rectification masks and the MP pooling switches has 15% lower L^2 reconstruction error (on validation images) compared than using pooling switches alone, and 46% lower than using the rectification masks alone. Finally, pooling switches alone have 36% lower L^2 error than using only rectification masks.

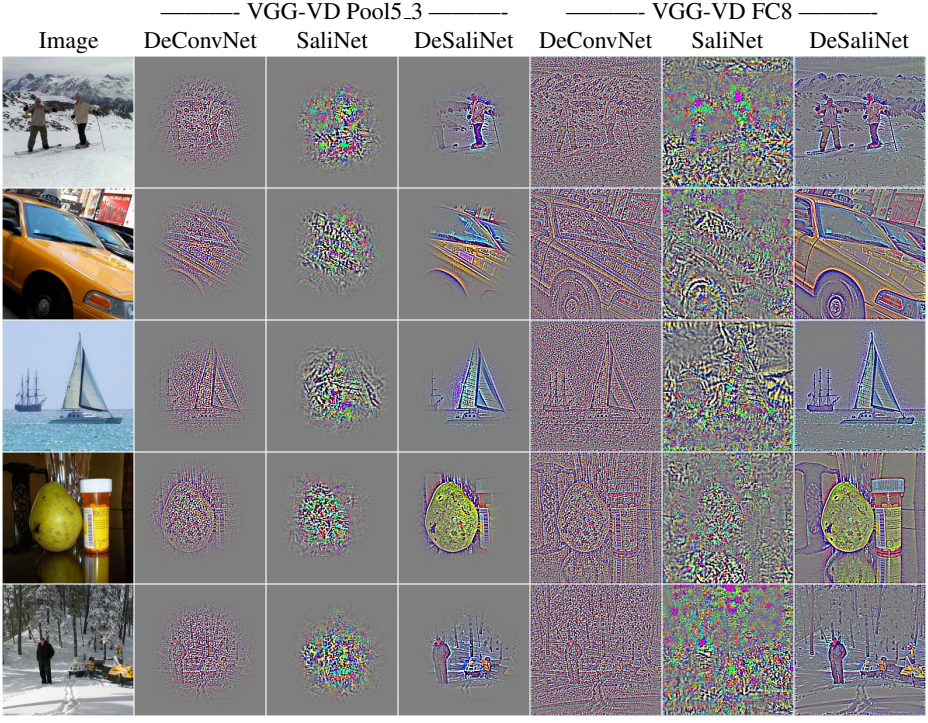

Fig. 6: *Foreground object selectivity.* This figure compares the response of DeConvNet, SaliNet, and DeSaliNet by visualizing the most active neuron in Pool5_3 and FC8 of VGG-VD. SaliNet and DeSaliNet tend to emphasize more foreground objects (see e.g. the faces of people), whereas DeConvNet's response is nearly uniform. Note that the apparent spatial selectivity of Pool5_3 is due to the finite support of the neuron and is content independent. Best viewed on screen.

3.4 Dominance of "phase" information

If a message emerges from the previous sections, it is that the response of all reversed architectures $\phi^\dagger(\mathbf{e}, \mathbf{r})$ is *dominated by the bottleneck information* $\mathbf{r}$. As seen in Sect. 2, the bottleneck information comprises 1) the setting of the pooling switches in the Max Pool units and 2) the setting of the masks in the ReLU units.

Interestingly, this information does not code for the intensity of the neural activations, but rather for their spatial location and polarity. We argue that this information is somewhat similar to phase information in the Fourier transform and related representations. To explain this analogy, consider the Fourier transform $X(\omega_x, \omega_y) = \mathcal{F}[\mathbf{x}](\omega_x, \omega_y) \in \mathbb{C}$ of image $\mathbf{x}$; a well known result is that if one replaces the modulus of the Fourier transform with a random signal but preserves the phase, then the reconstructed image $\hat{\mathbf{x}} = \mathcal{F}^{-1}[|Y(\omega_x, \omega_y)|e^{i\angle X(\omega_x, \omega_y)}]$ still contains the structure (edges) of $\mathbf{x}$ and very little of $\mathbf{y}$ is recognizable. In fact the resulting image, an example of which is shown in fig. 7, is not dissimilar from the output of DeConvNet and DeSaliNet.

In the Fourier transform, changing the phase of a spectral component $Ae^{j(\omega x + \theta)}$ by

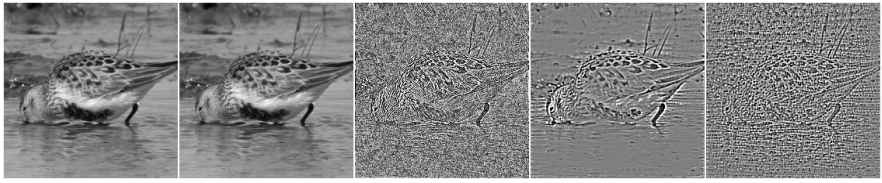

Fig. 7: *Analogy with phase information in Fourier Transform.* From left to right: The original image, the inverse Fourier transform of the Fourier transform of the original image, the inverse Fourier transform but after randomizing the amplitude of the spectrum, DeSaliNet $\phi^{\dagger}(\mathbf{p}, \mathbf{r})$ with positive random input ($\mathbf{p}$) and DeConvNet with positive random input ($\mathbf{p}$). Best viewed on screen.

Table 1: Mean Intersection over Union (IoU) and Mean Per-Pixel (PP) accuracy for different segmentation methods on the dataset of [4].

Method	CNN	PP	IoU	CNN	PP	IoU	Method	PP	IoU
SaliNet	AlexNet	82.82	57.07	VGG-16	82.45	56.33	Baseline	78.97	46.27
DeSaliNet	AlexNet	82.31	55.57	VGG-16	83.29	56.25	Guillaumin *et al.* [4]	84.4	57.3
DeConvNet	AlexNet	75.85	48.26	VGG-16	76.52	48.16	Baseline of [4]	73.4	24.0

$\Delta\theta$ amounts to shifting it by $-\Delta\theta/\omega$. Furthermore, negating the signal is equivalent to a phase shift of π. In the deconvolutional architectures, the max pooling switches record the location of filter activations, whereas the ReLUs applied in the backward direction contribute to reconstructing the polarity. More precisely, in the forward pass the ReLU block computes $y = \max\{0, x\}$. In the backward direction, the signal $\hat{y}$ is propagated towards the input as follows:

$$\hat{x} = \max\{\hat{y}, 0\} \text{ (in DeConvNet)}, \quad \hat{x} = \max\{\hat{y}, 0\} \odot [x > 0] \text{ (in DeSaliNet)}. \quad (10)$$

We see that both constructions guarantee that the polarity of the backward signal $\hat{x}$ is the same as the polarity of the forward signal y, which is non-negative. In fact, DeConvNet guarantees that $y\hat{x} \geqslant 0$, and DeSaliNet adds the guarantee that $y = 0 \Rightarrow \hat{x} = 0$. The first condition is stronger in term of preserving the polarity, and as seen in fig. 3 it is necessary to obtain a clear reconstruction of the image edges.

3.5 Objectness for free: weakly-supervised salient object segmentation

In this section we demonstrate that pre-trained CNNs reversed using SaliNet and DeSaliNet can accurately segment generic foreground objects. To this end, we consider the benchmark dataset of [4] consisting of 4276 ImageNet images annotated with the binary mask of the foreground objects. Notably, the object categories in this benchmarks are partially disjoint from the ones in the ImageNet ILSVRC data used to pre-train the CNNs: of the 445 synsets present in the segmentation benchmark data only 215 of them overlap with the 1000 ILSVRC classes.

In order to perform segmentation, we improve the setup of [16]. Given an image

$\mathbf{x}$, the CNN $\phi(\mathbf{x})$ is evaluated until the last layer before softmax (FC8 in AlexNet[5] and VGG-VD), recording the bottleneck information $\mathbf{r}$. Rather than resizing the image to the standard network input size, the CNN is applied in a fully convolutional manner [11]. The tensor $\mathbf{e}^*$ is set to the indicator of the channel that contains the maximally-activated neuron in FC8, the L^∞ norm of each RGB triplet in the output $\hat{\mathbf{x}} = \phi^\dagger(\mathbf{e}^*, \mathbf{r})$ of the reversed architecture is computed, and the resulting saliency map is used in GrabCut to segment the object as in [16]. Besides the ILSVRC data used to pre-train the CNN and 98 segmented images for validating the design choices above, there is no further training. For segmentation, this is a weakly-supervised setting as no object bounding boxes or segmentations are used for training.

Table 1 and Fig. 8 compare the three reversed architectures and the method of [4], which uses a combination of segment transfer from VOC2010 data, label propagation, bounding box annotations for 60k training images, and class label annotations for all images. It also compares a simple baseline obtained by assuming as a saliency map a fixed Gaussian blob (fig. 8), similar but much better than the analogous baseline in [4].

DeSaliNet and SaliNet performed about as well, much better than the baseline, and nearly as well as the method of [4], despite using weak supervision and a training set that, for the most part, contains different classes from the test set. This suggests that CNN learn the appearance of generic objects, which SaliNet and DeSaliNet can extract efficiently. DeConvNet did not perform better than the Gaussian baseline confirming its lack of foreground selectivity (Sect. 3.3).

Somewhat surprisingly, VGG-VD did not perform better than AlexNet, nor DeSaliNet better than SaliNet, despite achieving in general much sharper saliency maps. Qualitatively, it appears that GrabCut prefers a more diffuse saliency map as opposed to a sharper one that focuses on the object boundaries, which may create "holes" in the segmentation. In fact, GrabCut improves dramatically even the weak Gaussian baseline.

4 Discussion

In this paper we have derived a general construction for reversed "deconvolutional" architectures, showed that BP is an instance of such a construction, and used this to precisely contrast DeConvNet and network saliency. DeSaliNet produces convincingly sharper images that network saliency while being more selective to foreground objects than DeConvNet.

We showed that the sharpness of generated images depends mainly on the polarization enforced by reversed ReLU units, followed by the ReLU unit masks, and with a secondary contribution from the max pooling switches. Some of these ideas may be transferable to other applications of deconvolution such as the U-architecture of [15] for semantic segmentation. We also showed that bottleneck information (pooling switches and ReLU masks) dominates the output of deconvolutional architectures which questions their utility in characterizing individual neurons.

Acknowledgements. We gratefully acknowledge the support of ERC StG IDIU for Andrea Vedaldi and of BP for Aravindh Mahendran.

[5] For DeSaliNet, the LRN layers in AlexNet are reversed using BP-reversal $\mathrm{LRN}^{\mathrm{BP}}$ instead of the identity, which was found to be slightly superior in terms of IoU performance.

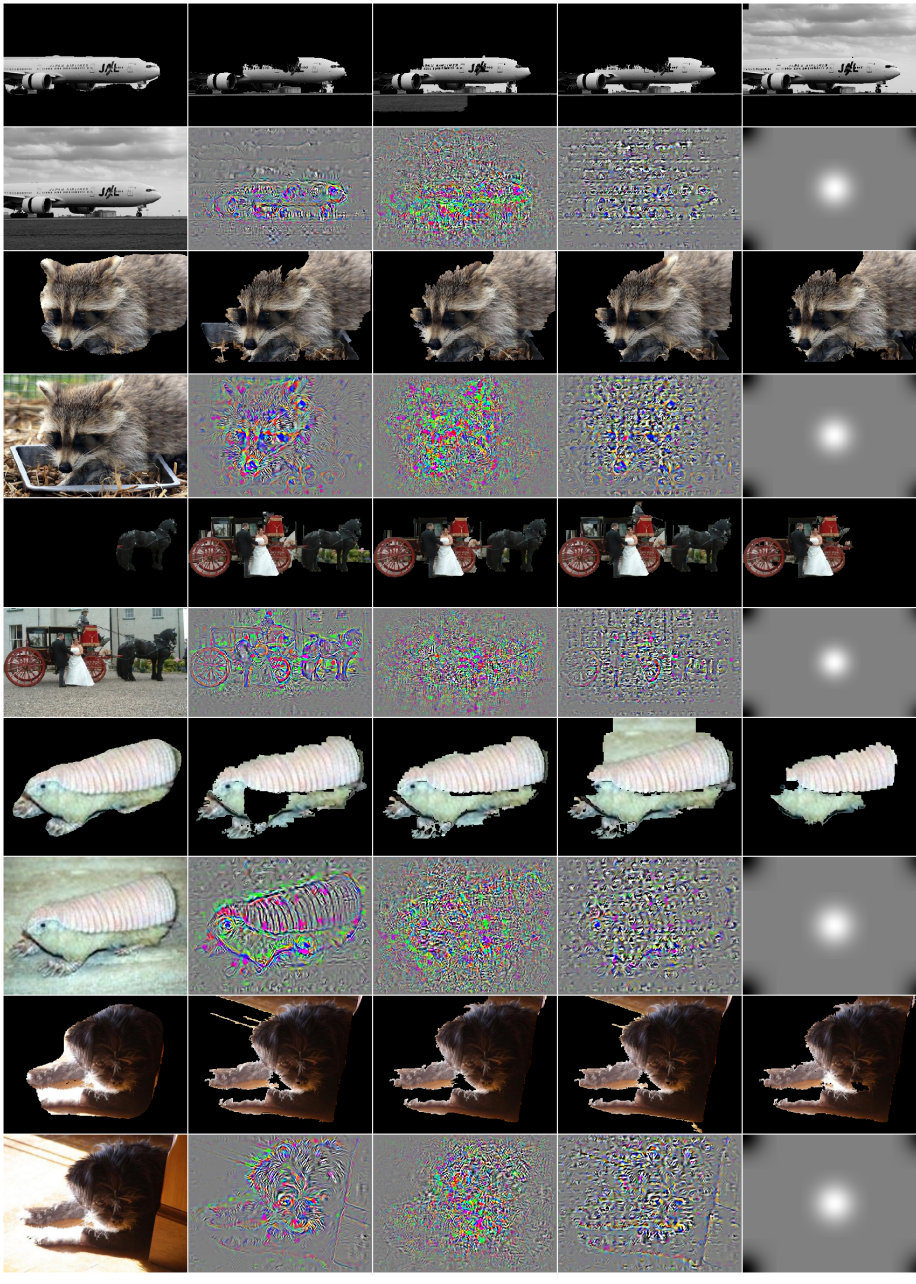

Fig. 8: Segmentation results (random selection). For each image, the top row shows the GrabCut segmentation and the bottom row shows the output of the corresponding deconvolutional network derived from AlexNet

References

1. Bishop, C.M.: Neural Networks for Pattern Recognition. Clarendon Press, Oxford (1995) 3
2. d'Angelo, E., Alahi, A., Vandergheynst, P.: Beyond bits: Reconstructing images from local binary descriptors. In: Proc. ICPR. pp. 935–938 (2012) 3
3. Dosovitskiy, A., T.Brox: Inverting visual representations with convolutional networks. In: Proc. CVPR (2016) 10
4. Guillaumin, M., Küttel, D., Ferrari, V.: Imagenet auto-annotation with segmentation propagation. IJCV (2014) 2, 12, 13
5. Hong, S., Noh, H., Han, B.: Decoupled deep neural network for semi-supervised semantic segmentation. In: Proc. NIPS. pp. 1495–1503 (2015) 3
6. Jensen, C.A., Reed, R.D., Marks, R.J., El-Sharkawi, M., Jung, J.B., Miyamoto, R., Anderson, G., Eggen, C.: Inversion of feedforward neural networks: algorithms and applications. Proc. of the IEEE 87(9) (1999) 3
7. Kato, H., Harada, T.: Image reconstruction from bag-of-visual-words. In: Proc. CVPR (2014) 3
8. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Proc. NIPS (2012) 7
9. Lee, S., Kil, R.M.: Inverse mapping of continuous functions using local and global information. IEEE Trans. on Neural Networks 5(3) (1994) 3
10. Linden, A., Kindermann, J.: Inversion of multilayer nets. In: Proc. Int. Conf. on Neural Networks (1989) 3
11. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proc. CVPR. pp. 3431–3440 (2015) 3, 13
12. Lu, B.L., Kita, H., Nishikawa, Y.: Inverting feedforward neural networks using linear and nonlinear programming. IEEE Trans. on Neural Networks 10(6) (1999) 3
13. Mahendran, A., Vedaldi, A.: Understanding deep image representations by inverting them. In: Proc. CVPR (2015) 2, 9
14. Nguyen, A., Yosinski, J., Clune, J.: Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In: Proc. CVPR (2015) 3
15. Noh, H., Hong, S., Han, B.: Learning deconvolution network for semantic segmentation. In: Proc. ICCV. pp. 1520–1528 (2015) 3, 13
16. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: Visualising image classification models and saliency maps. In: Proc. ICLR (2014) 1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13
17. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. In: Proc. ICLR Workshop (2015) 2, 3
18. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: Proc. ICLR (2014) 3
19. Tatu, A., Lauze, F., Nielsen, M., Kimia, B.: Exploring the representation capabilities of the HOG descriptor. In: ICCV Workshop (2011) 3
20. Várkonyi-Kóczy, A.R., Rövid, A.: Observer based iterative neural network model inversion. In: IEEE Int. Conf. on Fuzzy Systems (2005) 3
21. Vondrick, C., Khosla, A., Malisiewicz, T., Torralba, A.: HOGgles: Visualizing object detection features. In: Proc. ICCV (2013) 2
22. Weinzaepfel, P., Jégou, H., Pérez, P.: Reconstructing an image from its local descriptors. In: Proc. CVPR (2011) 3
23. Williams, R.J.: Inverting a connectionist network mapping by back-propagation of error. In: Proc. CogSci (1986) 3

24. Yosinksi, J., Clune, J., Nguyen, A., Fuchs, T., Lipson, H.: Understanding neural networks through deep visualization. In: ICML Deep Learning Workshop (2015) 2, 9
25. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Proc. ECCV (2014) 1, 2, 3, 6, 7, 8
26. Zeiler, M.D., Krishnan, D., Taylor, G.W., Fergus, R.: Deconvolutional Networks. In: Proc. CVPR (2010) 2
27. Zeiler, M.D., Taylor, G.W., Fergus, R.: Adaptive Deconvolutional Networks for Mid and High Level Feature Learning. In: Proc. ICCV (2011) 2