Visualizing Deep Neural Network by Alternately Image Blurring and Deblurring

Feng Wang\textsuperscript{a}, Haijun Liu\textsuperscript{a}, Jian Cheng\textsuperscript{a,∗}

\textsuperscript{a}School of Electronic Engineering, University of Electronic Science and Technology of China

Abstract

Visualization from trained deep neural networks has drawn massive public attention in recent. One of the visualization approaches is to train images maximising the activation of specific neurons. However, directly maximising the activation would lead to unrecognizable images, which cannot provide any meaningful information. In this paper, we introduce a simple but effective technique to constrain the optimization route of the visualization. By adding two totally inverse transformations, image blurring and deblurring, to the optimization procedure, recognizable images can be created. Our algorithm is good at extracting the details in the images, which are usually filtered out by previous methods in the visualizations. Extensive experiments on AlexNet, VGGNet and GoogLeNet illustrate that we can better understand the neural networks utilizing the knowledge obtained by the visualization.

Keywords: Visualization, Deep Neural Network, Image Blurring and Deblurring

1. Introduction

Deep neural networks (DNNs) achieve state-of-the-art performance in various computer vision tasks, such as object recognition\cite{1,2,3}, detection\cite{4} and segmentation\cite{5}. However, with the high performance of DNNs, the explanation of how and why DNNs work is relatively rare. Visualization has been
Figure 1: The illustration of our proposed visualization algorithm. We take the visualization of a specific class, megalith, as an example. The elements of the class we want to visualize is in blue, while the red and green ones are of other classes. If we directly optimize Equation 1, a noisy image will be generated (top). The recognizable part is almost totally drowned in the high-frequency noise. In this paper, we propose a way to guide the optimization. By alternately blurring and deblurring, the generated images are much clear and rich in details. Best viewed in color, zoomed in.

seen as a powerful technique to understand how deep neural networks work [6][7][8]. People can understand what a specific neuron represents by the image created through visualization algorithms. For example, through visualization algorithms, we can see edges from the first layer, corners or shapes from the second layer, object parts from intermediate layers and objects from the last layer.

There are mainly two ways to visualize a neural network. The first one is data-driven, showing the strongest feature map activations [4], deconvolving neurons that is activated by input images [8] or reconstructing an image that matches a given input [6]. With the help of the input images, it is much easier to visualize the activated neurons with respect to the inputs.

The other visualization approach is model-based, which needs the trained neural network model only. Its main procedure is to train images to maximise
the activation of a specific neuron, with the parameters of the neural network remaining fixed\cite{9}. If we observe one neuron unit $u$ of the neural network, the neuron output can be seen as a function $f_u(I) \in \mathbb{R}$, w.r.t an image $I$. By optimising the problem,

$$\arg\max_I f_u(I),$$  \hspace{1cm} (1)

we can obtain image $I$ that maximizes the value of neuron unit $u$, i.e. we get the image patches that most strongly activate the neuron $n$. When the activation is chosen from the feature maps, the produced images will display what the neuron represents. When the activation is chosen to be one unit of the classifier, images of the corresponding class will be generated.

However, directly optimizing the Equation (1) will lead to unrecognizable noisy images \cite{10}. The preliminary reason is explained in \cite{7}. The problem has two folds. Firstly, filters of CNNs' first layer are mostly high-pass filters, with only a few color ones (Figure 6) being low-pass. Secondly, similar with the result of FFT, the responses of low-pass filters are much higher than high-pass filters in natural images. However, the second layer tends to balance all the responses from the first layer, so it would always give higher weights to the activation of high-pass filters. Due to the back propagation algorithm, the gradients passed from the second to the first layer are imbalanced, with high-pass filters getting much higher gradients through the weights. The quantity of high-pass filters is more than low-pass ones, and they also get more gradients. As a result, the gradients propagated to the image layer are almost full filled with high frequency noises. People can hardly find useful information from the noisy images (Figure 1 top), let alone learning more knowledge about neural network from the visualizations. Thus, the problem is transferred into how to make the training image $I$ more natural with respect to human eye perception.

1.1. Related works

To avoid the high frequency noise problem, researchers proposed to add some constraints on the input image $I$ to make it more likely to be a natural image. Various kinds of constraints were investigated by researchers, such as
weight decay [6][11][12][7], total-variance norm [6][12], gradient clip [7], image blurring [7] and patch prior [12] etc. The effects of the priors and constraints are listed below.

1. **Weight decay:** Weight decay is a widely used strategy to avoid large values in optimization problems. It can suppress large values produced by the high frequency gradients. Furthermore, weight decay has an effect of suppressing the background, whilst making the foreground more prominent when applied on image. With this feature, this prior can also be used to generate class saliency image [11].

2. **Total-variance norm:** Total-variance norm is widely utilized to make neighbour pixels numerically continuous. The total-variance norm of an image $I$ is defined by

$$
TV^\beta(I) = \sum_{i,j} ((I_{i,j+1} - I_{i,j})^2 + (I_{i+1,j} - I_{i,j})^2)^{\beta/2},
$$

where $\beta$ controls which norm should be used to penalise the numerical gradient of neighbour pixels. However, total-variance norm may cause “spike” problem, which leads to lower image quality [6].

3. **Gradient clip:** The generated images have both foreground and background. Usually the gradients corresponding to the foreground are much stronger than those from background. However, sometimes the foreground may be confounded with the background. Besides adding weight decay, we can also directly clipping the weakest gradients out. With gradient clipping, only the foreground is updated. However, its parameter is difficult to tune because we would not know how many pixels the foreground occupies for an arbitrary neuron.

4. **Image blurring:** Since the directly trained images are full-filled with high-frequency noises, it is intuitive to use some denoise algorithms, such as image blurring with gaussian low-pass filters. Note that training images is a dynamic procedure, we can insert the blurring step into the training cycle. With the most of the high-frequency noises being suppressed, the produced
images will be more interpretable. Note that blurring with a small kernel low-pass filter several times is equivalent to blurring with a big kernel. The total-variance norm constraint mentioned above can also be seen as a low-pass filter with kernel size of $3 \times 3$. So there is no need to use image blurring and total variance norm at the same time.

(5) **Patch prior**: Strictly speaking, this prior is data-dependent. Because the color information is not always conclusively useful for classification, a discriminative neural network tends not to record so much color information (Figure 3 in [13]). The images generated from a discriminative neural network are always inaccurate in color. Using the patches collected from natural images, the generated images’ color will be more close to real images.

(6) **Image Jittering**: This is a technique proposed in an unpublished blog[^1]. They randomly crop a sub-region from a bigger image and update the region for a few iterations. This technique is feasible because most of the modern CNNs have invariance against small image translation and scaling. Through this technique, each pixel is updated by multiple sub-region images at different location. Thus, it can also be seen as a kind of neighbour averaging. Its effect is similar with total-variance norm and image blurring, but it also has information sharing in different locations of the CNN’s data layer. This technique is only suitable for visualizing the deeper layers, because shallow neurons do not have the property of scale invariance. When visualizing the final layer, as we experimented, this technique is the most effective one. This is mainly because modern CNNs are always trained with data augmentation techniques such as image translation and small image scaling. Image translation and scaling have already encoded into the CNN models.

(7) **Center-biased regularization**: This regularization is based on an observation that the produced images always contain multiple overlapped in-

[^1]: [http://googleresearch.blogspot.no/2015/06/inceptionism-going-deeper-into-neural.html](http://googleresearch.blogspot.no/2015/06/inceptionism-going-deeper-into-neural.html)
stances and the target object is usually occupies only a small region of the whole image. Nguyen et. al\cite{14} proposed to update the gradients in the center region only after several iterations of global update. Since they also used the image blurring technique, the non-center areas would be blurred out. The model would still attempt to create enough information in the small region. Thus, they claimed that they could create more structurally complete objects.

(8) **Multifaceted initialization** One neuron in deep layers usually corresponds to multiple instances. Using gaussian random initialization, we would obtain only a small subset of all instances due to the highly non-convex structure of neural networks. Multifaceted initialization\cite{14} is a data-dependent technique which can generate multiple initial images using clustering algorithms applied on image patches that highly activate the target neuron. Through this technique, we may obtain many potential visualizations hidden in the complex structure of the neural network function $f_u(I)$ in Equation 1.

(9) **Generative Adversarial Networks** Recently, Generative Adversarial Networks (GAN) model\cite{15} is applied as a special form of image prior for creating neural network visualizations\cite{16}. In \cite{16}, a pre-trained generator network is spliced to the target network. The activation maximizing loss function propagates the gradients through both the target network and the generator network to optimize the bottom feature vector of the generator. In another word, the GAN-based model optimize the bottom feature vector instead of the image $I$. After converge, the bottom feature is forwarded through the generator network to create natural images as visualization with regard to the target neuron.

In summary, the priors and constraints can be divided into four categories by their purposes. Total-variance norm, image blurring and image jittering aim at filtering the high-frequency noise out. Weight decay, gradient clip and center-biased regularization are utilized to suppress the background. The patch prior is
introduced to make the produced image’s color more close to the color of natural images. Multifaceted initialization is a technique that can generate more kinds of instances for one target neuron. The GAN based model is a perfect image prior and it can directly create realistic images.

These techniques can also be divided into two groups by whether they need extra data or not. The patch prior and multifaceted initialization are data-dependent, they require additional image data in the preprocessing or training procedure. GAN based model requires additional data in its training procedure, but it also shows excellent generalization performance to other neural networks that haven’t been trained on. This method needs no additional data in generating visualization and can work even better if fine-tuned on the target neural network models.

1.2. Motivation

As we explained previously, the gradients back-propagated to the image layer are full-filled with high frequency signals. One intuitive idea to solve this problem is to blur the high frequency signals out\[7\]. However, the foreground usually has more gradients compared with the background. This property has been utilized to generate saliency maps\[11\]. The blurring kernel used for foreground recovery may not be suitable for the background and the details in foreground. Moreover, directly using small kernels usually leads to disordered object structures as shown in Figure 2. Thus, we propose to add one more step, deblurring the image using a smaller kernel, to recover the background and details in the foreground when generating visualization.

1.3. Contribution

In this work, we propose a method to guide the production of the visualization images. It includes two basic image operators, image blurring and image deblurring. The two operators can be implemented by convolving and deconvolving with a gaussian low-pass filter. The blurring operation is used for filtering the high frequency noises out and the deblurring operation is to
counteract the blurriness caused by image blurring. Our algorithm is good at extracting the background and details in foreground, which is usually ignored by previous methods.

With the visualization of neurons, we can observe some interesting phenomena and knowledge to better understand the neural networks:

(1) **Filter group in Alexnet** [1]: The second convolution layer of Alexnet is trained with two groups of filters. This causes a problem that the two groups of filters have totally different behaviours observed by the visualization. The first group contains more lines and edges, while the second one is colorful.

(2) **Effect of bottleneck structure in GoogLeNet**: GoogLeNet’s Inception block has a thin $1 \times 1$ convolution layer before a $3 \times 3$ or $5 \times 5$ convolution layer to compress the parameters to be trained. From the visualizations, we could see that a single neuron in the $5 \times 5$ layers usually contains multiple instances in it, while the $3 \times 3$ layers usually only have one. This means that the $5 \times 5$ branches work differently from other branches. They can be activated with more kinds of input images.

(3) **Tendency of different models**: Ensemble is a powerful tool to improve the performance of many pattern recognition tasks. However, we usually take the neural network models as black boxes, and select the models for ensemble peremptorily. Now with the help of visualization, we can observe the tendency of different models, some are good in local details, and some remember more global shape information et al. Utilizing the visualizations, we may select models that have high diversity of tendencies.

### 2. Method

In this section, we will introduce the blurring and deblurring strategy to constrain the optimization route of Equation [1] and techniques to tune the hyper-parameters of our algorithm.
2.1. Visualization by Activation Maximization

Weight visualization of the first layer is a widely used technique to examine the effectiveness of a trained neural network model. The algorithm of visualizing the first layer’s weight is quite simple: just normalize the $(3 \times s \times s \times n)$ weights separately and $n$ filters with size of $(3 \times s \times s)$ will be obtained, where 3 regards to RGB channels, $s$ is the side length of the filters and $n$ is the number of output channels. However, this method cannot be generalized to the second or deeper layers. For example, the second layer can be seen as the linear combination of the first layer’s activations. By just showing the weights of the second layer, no meaningful information will be obtained because of the complex factors such as the first layer’s activation function and the spatial correlation of the weights with size larger than $1 \times 1$.

However, when we reformulate the question, visualizing the weight of the first layer can be written as,

$$\arg\max_x w x + b,$$

s.t. $\|x\| = 1$.  \hspace{1cm} (3)

The constraint, $\|x\| = 1$ is added to limit the magnitude of the visualization images. This is a linear programming problem and the solution is

$$x = \frac{w}{\|w\|}.  \hspace{1cm} (4)$$

In this perspective, directly showing the normalized weights is equivalent to activation maximization for the first layer. More importantly, this activation maximization method can be generalized to the deeper layers of a neural network,

$$\arg\max_I f_u(I),$$

s.t. $\|I\| = 1$. \hspace{1cm} (5)

where the $f_u(I) \in \mathbb{R}$ is the forward function of a specified neuron $u$ w.r.t an input image $I$. This function is highly non-convex. Even though we can find a local minima by gradient ascent algorithm, directly optimizing the formulation
on deep neural networks would always creates images full of noises\cite{10}. The noises can also produce high neuron activation, but have no practical meanings. Thus, the next task is to find a method that can produce meaningful and visually pleasurable images. In order to achieve this goal, some constraints need to be added to guide the optimization route towards the natural image subspace.

In this work, we propose to use two fundamental operators in computer vision, image blurring and image deblurring. By alternately applying these two operations, the training image will finally converge to a subspace in which images are all have a property that they are visually stable under the two operations.

2.2. Image Blurring and Deblurring

As mentioned in Section 1.1, image blurring is an intuitive method when people see the highly noisy image. This step is simply implemented by convolving the training image with a gaussian point spread function (PSF) $G$ of size $(s, s)$ and a specific $\sigma$. The $s$ and $\sigma$ are determined by the receptive field size of the neuron to be visualized. With a general size of the input image of AlexNet\cite{1}, VGG\cite{2} and GoogLeNet\cite{3}, a good parameter set is $s = 7$ and $\sigma = 0.8$ (Figure 2). Written in math formula, the blurred image $\hat{I}$ is,

$$\hat{I} = I \ast G.$$ (6)

In general, purely blurring the image will cause lack of details, especially the details in the background, where the pixels can not receive as many gradients as
the foreground. However, the context information is also important for classifying an image. Visualization algorithm should not ignore it. Thus, we propose to add a deblurring step after blurring. It can recover some of the high-frequency information discarded by the blurring step.

Deconvolution with known PSF, also named as non-blind deconvolution, is a fundamental topic in signal and image processing. It is the inverse process of convolution, given \( \hat{I} \) and \( G \), attempting to recover \( I \). Many effective algorithms have been proposed by researchers. In this paper, we choose Richardson–Lucy deconvolution algorithm\([17][18]\) because it is more stable with repeated iterations.

Deblurring is applied on the training image alternately with blurring by several iterations interval. The size of the deblurring PSF should be the same with the one used for blurring, but the standard deviation is set to be slightly smaller than the one of blurring, because the gradients tend to have more high frequency information as we explained in the introduction section.

2.3. Optimization for Visualization

Gradient ascent with fixed learning rate and momentum of 0.9 is used to optimize Equation 1. If we want to see the visualization of a neuron, we just set the gradient of the neuron to 1 and other neurons’ gradients to 0. Then back propagation algorithm is applied to get the gradient \( f'(I) \) w.r.t. the training image \( I \). While applying the gradient ascent, we alternately blur or deblur \( I \) every \( n \) iterations, where \( n \) is set manually. Here we call every \( 2n \) iterations as an epoch, with one blurring and one deblurring applied to the image. When the historically maximal output value \( f(I) \) remains unchanged for 10 epochs of blurring and deblurring, the optimization procedure is terminated.

As shown in Figure 2, higher \( \sigma \) would make the produced image blurry, but has better structural integrity, while image created with lower \( \sigma \) is disordered but rich in details. Thus, we introduce a fine-tune strategy. After using a higher \( \sigma \) to train the image by some iterations, we halve the \( \sigma \) and the learning rate, and continue to train the image by a few more iterations (Figure 3). In the fine-
Figure 3: **left:** The whole loss wave of training an image. The red line is the loss with no constraints. **right:** Loss in one epoch. The loss decrease steeply when blurring or deblurring applied on the training image. Then the gradient ascent algorithm starts to accumulate gradient to eliminate the influence. During each interval, pixels must accumulate enough gradients to build an image with a recognizable dynamic range. Note that the standard deviation of deblurring kernel is slightly smaller than that used by blurring, so the decrease caused by deblurring is also smaller. After 1000 iterations, fine-tuning is applied by halving the $\sigma$ and learning rate. We can also observe from the left figure that by fine-tuning, we can get higher activation value, revealing that our constraint can overcome some of the local minima.

In this section, we will firstly describe how to tune the hyper-parameters, such as the learning rate and the kernel standard deviation $\sigma$ in image blurring and deblurring (Section 3.1.1 and 3.1.3). Then we will show the visualizations of some popular networks and discuss what can we learn from the visualizations (Section 3.2). Section 3.3 shows the effectiveness of another two techniques, randomly cropping and scaling. Finally, taking these two techniques and our alternately blurring and deblurring algorithm together, we indicate that information sharing is the key factor that make these methods work.
In this paper, we will not compare our method with the GAN model because our algorithm is an incremental work based on traditional image prior methods and our goal is to overcome the shortage of the previous constraints.

### 3.1. Parameter Tuning

In order to control the dynamic range of the produced image in an acceptable range, we should manually tune the learning rate and the interval of blurring and deblurring. Due to the high non-linearity of deep neural networks, we cannot provide a set of general parameters for every model. However, we will introduce some tricks to give some guidance of tuning the learning rate and the interval of the blurring and deblurring.

#### 3.1.1. Learning rate

The trick for tuning the learning rate is based on an observation that under the circumstance that no regularization nor constraint is added, the learning rate producing the best image dynamic range will approximately lead to the biggest activation when it stops increasing, i.e. too low or too high learning rate will both cause lower activation. In practice, the stop criteria is set to when the least square slope of the loss is below 1e-3. When we need to determine the learning rate automatically, we may turn off the image blurring and deblurring and use a grid search to find the learning rate that lead to the highest activation. However, in practice, we usually manually try several times and find a learning rate that creates a visually acceptable image.

#### 3.1.2. Interval of Image Blurring and Deblurring

For tuning the interval of the image blurring and deblurring, we need the assistance of the curve of the loss value of Equation 1. The blurring and de-blurring’s influence on the loss is plotted in Figure 3. Both the blurring and the deblurring algorithm can decrease the loss, however, in different ways. When the decrease caused by blurring and deblurring is precisely remedied by gradient ascent, the optimization is seen to be converged. Empirically, for RGB values in natural images, with size of $7 \times 7$ and standard deviation of 0.8, the pixel
\[ \sigma = 0.2 \quad \sigma = 0.3 \quad \sigma = 0.4 \quad \sigma = 0.5 \quad \sigma = 0.6 \quad \sigma = 0.7 \quad \sigma = 0.8 \]

Figure 4: Visualization of part of the filters in the conv2 layer of GoogLeNet [3] with three strategy of blurring and deblurring: (a) Pure Blurring. (b) Alternately blurring and deblurring using the same \( \sigma \). (c) Alternately blurring and deblurring with different \( \sigma \)s. Here we shrink the \( \sigma \) of deblurring by multiplying a factor of 0.8.

values changed by blurring and deblurring is approximately 5 – 10 in average. The interval should be tuned to ensure that the training image has accumulated enough gradients to eliminate the decrease on the loss function caused by blurring and deblurring. If the training image do not accumulate enough gradients during the intervals, the optimization will still converge, but with a lower image magnitude, and vice versa. Similar with tuning the learning rate, in practice, we may also manually try several times to search for a fine interval.

3.1.3. \( \sigma \) in Image Blurring and Deblurring

In this section, we will examine the benefits of our proposed algorithm compared with the previously widely used constraint, image blurring, as a baseline. There are mainly three hyper-parameters to be tuned, such as the learning rate, \( \sigma_{\text{blur}} \) and \( \sigma_{\text{deblur}} \). As we introduced in the last section, the learning rate can be determined by several attempts until the produced images’ dynamic range to be acceptable, the next task is to examine the effect of the \( \sigma \) in the blurring and deblurring procedure. Since the produced images will be different if the initial image is randomly created, in the following experiments in this section,
we will train a base image by gradient ascent without any constraints for 200 iterations, and use it as the initial image to do each experiment. In this way, all images in the experimental and control group will have similar instances but various styles caused by different constrain strategy.

To make our experiments more convictive, we illustrate the visualizations of two different layers from GoogLeNet[3]. One is from $conv2$ layer, which represents the low-level features(Figure 4). Another is from the inner-product layer before softmax classifier, which demonstrates the highest level neurons(Figure 5). Due to the limitation of the paper’s content size, we only display the starfish class neuron as an example.

As we can see from Figure 4 and 5 purely using image blurring may cause the produced image too blurry to observe. Using the same kernel both for image blurring and deblurring leads to rather good but roughened images. However, from the roughened images we can get better understanding about how alternately image blurring and deblurring works. Through image blurring and
deblurring, our algorithm creates “particles” on the image with their color collected from a specified local region. The size of the “particles” depends on the \( \sigma \) of the blur kernel. Empirically, the final \( \sigma \) in the two-step strategy described in Section 2.3 is set as 0.6 ∼ 0.8 to let the particle’s radius being within 2 pixels (> 0.9 energy in the center 3 × 3 region), which leads to the most natural image for people to cognize.

Finally, we shrink the \( \sigma_{\text{deblur}} \) by a factor of 0.8 to smooth the particles. We claim that using a smaller \( \sigma \) for deblurring is different from directly blurring the image with a small kernel, e.g. blurring the image with \( \sigma = 0.2 \times \sigma_{\text{blur}} \). Blurring with small kernel cannot erase the high frequency part of the training image as illustrated in Figure 4 and Figure 5.

3.2. Visualization of popular networks

In this section, we provide visualizations of various neurons in different models, such as AlexNet[1], VGGNet[2] and GoogLeNet[3]. Through the visualizations, we can make some interesting conclusions to help us better understand how neural networks work, especially for convolutional neural networks. All the models are downloaded from Caffe[19] Model Zoo. All the codes for the following experiments can be found in [https://github.com/happynear/DeepVisualization](https://github.com/happynear/DeepVisualization).

3.2.1. Visualization of Filters

Our algorithm can be utilized to show every neuron of a given neural network model. Due to the space limitation of this paper, we only display the filters of AlexNet’s convolution layers (Figure 6) and some convolution layers in VGGNet (Figure 7). For GoogLeNet[3], we illustrate the neurons from Inception layers (Figure 8), which is a special CNN structure proposed in [3]. For more visualization results, please see the supplement materials.

**AlexNet[1]**: We can see from Figure 6 that the AlexNet’s visualization is the most unrecognizable. We can hardly find any meaningful object parts in the visualization. Its network capacity is too small to contain the whole...
Figure 6: Filters from the convolution layers of AlexNet [1]. Due to the space limitation, we only show a part of filters in conv2-conv4. Note that there is a clear boundary between the two groups of conv2 layer. The first 128 filters (32 filters in the figure) are mostly lines and edges, while the other 128 filters are more colorful. Best viewed in color, zoomed in.
ImageNet dataset, so it is not strange that the convolutional features are all fundamental bases. Due to the limitation of computation resources in the year of 2012, the second convolution layer of Alexnet is trained with two group of filters separately(Figure 6). This causes a phenomenon that the two groups of filters have totally different behaviours observed by the visualization, which is named as “pattern decoupling” by us. The first group contains more lines and edges, while the second one is full of colorful blobs. This phenomenon may not be a good characteristic of feature representation, while we would like the representation to contain more diversity, such as more colored lines and edges, and combinations of lines and blobs, etc. We cannot find these patterns in the conv2 layer of AlexNet in Figure 6. Thus, dividing filters in groups may not be a good methodology when the GPU memory is sufficient and the final purpose is an ultimate accuracy. However, pattern decoupling doesn’t always mean a bad thing. The loss in the representation capacity can be remedied by enlarging the network’s width or depth, and the group parameter can decrease the model’s volume by a large margin. In some recent works, the group parameter is picked up again by researchers to increase the width of the convolutional layers with limited GPU memory [20] [21] or decrease the model size without too much hurt on accuracy [22].

VGGNet [2]: VGGNet’s visualizations are much clearer compared with ones from AlexNet. We can see how the object is built from bottom to top. The convolution layers of VGGNet are very suitable for representing texture patterns [23] [24] [25]. The reason lies in three aspects. Firstly, the convolutional kernel of VGGnet is so small that the edge patterns can still be observed in the fourth parametric layer, and the object level patterns appear as late as conv5_1 layer. Patterns in layers before conv5_1 are similar with the properties of “textures”, which are larger than pixels and smaller than objects. Here we argue that the fifth stage of convolutional features is no longer suitable for representing texture, because layers in this stage contains more about object level patterns, which are too complex for describing textures. Secondly, the feature channel number of each layer is rich, or even redundancy. Every level of textures can
Figure 7: Filters from the convolution layers of VGGNet\cite{simonyan2014very}. We only illustrate some filters from the first convolution layers in each stage of VGGNet. Best viewed in color, zoomed in.
be represented with such rich bases. For example, we can find some repeated patterns in \textit{conv3,1} layer, such as a red sphere on green background. Finally, the background of the visualization map is much cleaner than GoogleNet. It is more suitable for describing the repeated textures, whose appearance is usually all foreground.

\textbf{GoogLeNet}\cite{3}: In the visualization of a special structure of GoogLeNet, which is named as Inception, different branches behave much differently (Figure 8). We can find recognizable objects in $1 \times 1$ \textit{convolution} branch, $3 \times 3$ \textit{convolution} branch and the latter layers of \textit{MaxPool} branch. The $5 \times 5$ \textit{convolution} branches usually contain piles of patterns, such as combination of lines in the earlier layers and mussy mixtures of objects in the latter layers. This is mainly because of the high compression rate of the $1 \times 1$ compression layer before the $5 \times 5$ convolution layer. Patterns in $5 \times 5$ branches are highly mixed to get more compact representations, and then expanded to a larger spatial area through the $5 \times 5$ convolutions. With this structure, various patterns may get positive responses on a same neuron, i.e. they share the same neuron. In this way, the model would be smaller while still has high ability of pattern expression.

\textbf{Comparison and Discussion:} We have also implemented some other constraints, such as weight decay \cite{6,7,11,12}, total-variance norm \cite{6,12} and gradient clip \cite{7}, to compare the performance with ours. Previous works usually take a subset of the constraints \cite{6,7,11}. In Figure 9 and 10, we provide the results of several model-based algorithms, such as weight decay only\cite{11}, weight decay and total-variance norm\cite{6} and image blurring\cite{7}.

For low-level features, adding no constraint or adding weight decay only lead to noisy, but still recognizable and stable images, except for AlexNet’s \textit{conv2} layer. Using weight decay may suppress some of the noise, but also makes the optimization procedure harder. For VGGNet’s \textit{conv2-1} layer and GoogLeNet’s \textit{conv2} layer, image blurring may cause the produced images too blurry to recognize.

For mid-level features, the problem gets more difficult. Weight decay and total-variance norm can only work in VGGNet. This is because VGGNet’s
Figure 8: Filters from the Inception layers of GoogLeNet [3]. It can be inferred from the illustration that different branches have different behaviors. 1×1 convolution branch has the smallest receptive field, and we can always find some recognizable objects in its visualization. 3×3 convolution branch has two ReLU layers, so it represents more complex features. There are many meaningful objects in it. MaxPool branch has similar receptive field with 3×3 convolution branch, but with less objects in earlier layers. Behavior of 5×5 convolution branch is somewhat weird, compared with 1×1 convolution branch and 3×3 convolution branch, little obviously meaningful object can be observed from its feature map. Best viewed in color, zoomed in.
convolution layers are all with $3 \times 3$ kernels, which suffers less gradient imbalance problem between low and high pass filters. From the middle layers, much more high frequency signals are propagated to the image layer. Purely applying image blurring starts to work well.

3.2.2. Visualization of Classes

Visualizing the class neurons is similar with visualizing the neurons in the front layers. Since each class neuron corresponds to multiple images, the produced images are not the same every time. So we firstly trained a base image on each classes with no prior and then used it as the initial image to generate images with other constraints. The results are shown in Figure 11. We can notice from the visualization that the images generated by our algorithm has more and clearer details.

3.3. Translation and Scaling

When visualizing the final layer, image translation and scaling are usually applied on the training images to utilizing the translation and scale invariance property of modern CNN models. Specifically, we modified the input image size to $[300, 300]$ and at every epoch we selected one $[224, 224]$ sub-region to update, then embedded the sub-region back to the big image. To make a comparison,
Figure 10: Visualization of mid-level filters, created with constraints: (a) weight decay only [11], (b) weight decay and total-variance norm [12], (c) image blurring [7], (d) our alternately blurring and deblurring. Filters are selected from AlexNet’s conv4 layer, VGGNet’s conv5 1 layer and GoogLeNet’s inception4b layer, conv3 3 branch. Best viewed in color, zoomed in.

Figure 11: [224, 224] images (original size) created from class neurons of GoogLeNet [3], with constraint (a) weight decay, (b) image blurring, (c) alternately image blurring and deblurring. Best viewed in color, zoomed in.
visualizations of class neurons with five different strategies in four different deep models were generated\(^2\) [Figure 12].

With multi-region updating, the produced visualization is generally much clearer and more structurally complete than updating in a single [224, 224] region only. It can also be observed from the illustration that applying translation transform only leads to noisy, but still recognizable images. The second and third rows’ images are relatively a little disordered compared with the images created by the last two methods, but they are much more natural compared with the translation only augmentation. One reason for explaining why scaling works better than translation is that the bilinear or cubic interpolation is also a low-pass filter, with every pixel being computed by the weighted summation of some neighbour pixels in the original image, which takes a similar role with image blurring.

The fourth strategy, translation and scaling with image blurring, creates images with good structure, but a little blurry and lack of background details. Our method, the fifth one, creates vivid and clear images, both in the foreground and background. However, this evaluation is subjective because we can hardly find an objective evaluation criterion for judging the visualization’s quality. Though subjective, we claim that our algorithm is at least no worse than the baseline technique, image blurring.

As a conclusion, translation, scaling, image blurring and deblurring can all promote the visual effect of the produced images. The effect of scaling and blurring is remarkably superior to translation, and the deblurring is a complementary technique for image blurring. Deblurring can counteract the blurriness caused by blurring, especially in the background. It is noteworthy that the

\(^2\)For GoogLeNet, we used two models released by two different institutes, one from BVLC group, website: \texttt{https://github.com/BVLC/caffe/wiki/Model-Zoo}, the other from Princeton University, website: \texttt{http://vision.princeton.edu/pvt/GoogLeNet}.

It is noteworthy that VGGNet’s gradient always has extreme large values in some locations, and the large values can pollute the neighbour pixels via image resizing or blurring, so we add weight decay to supress them. For more details, please see the complementary material.
translation and scaling can only be applied on the high level features since low level neurons do not have the property of invariance to image translation and scaling.

3.3.1. Tendency of Different Networks

From Figure[12] it can be observed that visualizations of class neurons in different networks look very differently. Even for the same architecture GoogLeNet, models trained by various institutes tend to remember different information. Images generated by BVLC’s GoogLeNet show more complete objects, especially for birds and fishes, while ones from Princeton’s model has more details on the surface of the objects, such as the particles on the starfishes and megaliths. These visualizations provide a strong evidence for the effectiveness of model ensemble algorithms. When choosing models for ensemble, it is a good manner to observe whether the models have different tendencies through a visualization toolkit.

4. Conclusion and Future Work

In this paper, we have proposed a new constraint strategy, alternately blurring and deblurring, to guide the optimization route of visualizing trained neural network model. We detailedly describe the effect and property of alternately image blurring and deblurring in feature visualization. Through the mid-level feature visualizations, we have explained why VGGNet[2] is so popular to describe texture features compared with GoogLeNet[3]. We have also illustrated how the effect of “bottleneck” layers by their visualizations. By showing the visualizations from various models, we show the tendencies of the models can be very different. Even the models with the same architecture would have different tendencies observed from the visualizations, which usually reveal what the models are good at. Besides alternately blurring and deblurring, we have also explained why another two operations, image translation and scaling, have great ability of making the produced image natural.
Figure 12: [300, 300] images created form class neurons of AlexNet [1], GoogLeNet [3] with two different implementation and VGG-16 Net [2]. These images are generated with strategies: (a) translation only and no constraint, (b) scaling only and no constraint, (c) translation and scaling, no constraint, (d) translation and scaling, image blurring, (e) translation and scaling, alternately blurring and deblurring. Please pay attention to the background and foreground details, especially for the eyes of the animals in the visualization of GoogLeNet. Best viewed in color, zoomed in.
We hope that the deep learning and computer vision community could take visualization as a fundamental tool to analyse the trained neural network models. We believe that we can obtain more knowledge through the visualization to guide us to better understand and use the neural networks.

Acknowledgment

This research was supported by the National Natural Science Foundation of China (61671125, 61201271, 61301269), and the State Key Laboratory of Synthetical Automation for Process Industries (NO. PAL-N201401).

References


