Visualizing and Understanding Convolutional Networks

Clémence Réda Xiaoqi Xu
École Normale Supérieure Paris-Saclay
61, avenue du Président Wilson, 94230 Cachan
{creda, xiaoqi.xu}@ens-paris-saclay.fr

Abstract

In this project, we seek a better understanding of how Convolutional Neural Networks work. In order to achieve this, we suggest a careful observation of the feature maps of each internal layer of the network, along with a study of the influence of training on the weights learned in the network. We focus our study on two small CNNs we trained ourselves, and partly on VGG16 [15] and ResNet50 [7]. First, we visualize the feature maps of intermediate layers in order to understand their role in the network. Then, we reconstruct the input images using activation maximization, and perform a quantitative analysis on reconstructed inputs using classic image matching methods.

1. Introduction

Convolutional Neural Networks (CNNs) have shown impressive capability in fields like computer vision and natural language processing. They are also widely used in many domains to upgrade state-of-the-art results in artificial intelligence. However, despite the ubiquitous application of CNNs, they are poorly understood in theory. Besides, Nguyen *et al*. [11] have noted that even state-of-the-art CNNs are easily fooled: their predictions might completely change when some noise, which are completely imperceptible to human eyes, is added to the data. Thus, understanding how and why CNNs classify and get trained could be of great interest, in order to be able to design robust layers, and improve comprehensibly the training of the networks.

Recently, some researchers have attempted to decode CNNs by developing various visualizing techniques. Erhan *et al*. [3] have proposed the method of *activation maximization*, which finds input patterns that maximize the activation of a given unit, using gradient ascent onto the pixel space. Yet this only gives information about a single unit, but is not able to account for invariances. Inspired by ideas of Erhan *et al*., Simonyan *et al*. [14] described a method to generate a representative image of a certain class from trained CNNs, by maximizing the considered class score in the top layer. They have also noticed the similarities between this method and Deconvolutional Networks (DeconvNets) described by Zeiler and Fergus [19].

2. Method

2.1. DeconvNet

We use the method of Deconvolutional Networks (DeconvNet) suggested by Zeiler and Fergus [19]. This model can be thought as applying the inverse transformations down from a given selected intermediate layer of the network, with respect to those applied in the forward process. To each layer (convolutional, pooling, ...), a "reversing" layer is associated, in the reversed order of the layers in the forward network. To visualize a given feature map of one of the intermediate layers of the networks, one needs to set other feature maps in the same layer to zero, and then pass it though the DeconvNet, i.e. through unpooling, rectification by a non-linear function (for instance, ReLU), and convolution with the transpose of the same filter weight matrix learned in the initial convolutional layer.

2.2. Activation maximization

Erhan *et al*. [3] have first introduced the idea of looking at which patterns can be generated by a given layer after training. These patterns can be reconstructed by maximizing the mean activation of the hidden unit of interest. This problem is thus formulated as an optimization problem: at fixed network weights and biases, the goal is to build an optimal input image in a iterative manner, using gradient ascent. In spite of the non-convexity of this problem, we might still be interested in a local maximum. This method has also been used in Google DeepDream [10] for generating non-existent elements in images. In our experiments, we use the same idea but maximize with respect to the output of the last fully-connected layer associated with a given object class (without the *softmax* activation function, because, as noticed by [14], it may alter results). The corresponding algorithm is described in Alg. 1.

Algorithm 1 Gradient Ascent for Activation Maximization

- Start from a random noisy image.
- Define a loss function that seeks to maximize the mean activation of a certain class neuron output.
- Use gradient ascent to "twist" the image.

return the image which maximizes the activation of the last fully connected layer output associated with the considered class.

2.3. CNN models & Data

We have constructed two CNNs using Keras, writing our own pipelines for training and testing (see our code at [13]). The architecture of one of the CNNs, denoted by *Conv* (whose architecture is borrowed from [12]) is shown in Fig. 1. The other model is denoted by *Vonc*, and has a similar architecture. Both of their architectures are rather standard –each block comprises one convolutional layer with a rectified linear activation function, followed by max pooling. The top layer is fully connected, and the final layer is a *softmax* classifier of size equal to the number of classes. These networks have been trained using data augmentation, and contain respectively $3,410,184$ (*Conv*) and $1,791,144$ (*Vonc*) trainable parameters. Both of the CNNs described above are trained on the CIFAR-10 dataset [8]. For experiments requiring the reconstruction of inputs *via* gradient ascent, one picture of a cat, and a set of 22 pictures of Siamese cats found on Google Images have been gathered.

3. Results

3.1. About our trained networks

	Training	Validation
Accuracy	0.566	0.515
Loss	1.213	1.344

Table 1. Final training accuracy and loss on CIFAR-10 dataset for our *Conv* model, trained for 10 epochs, in batches of 128 images, using Adam optimizer with decay rate 10^{-6} and learning rate equal to 10^{-4}.

	Training	Validation
Accuracy	0.818	0.758
Loss	0.517	0.707

Table 2. Final training accuracy and loss on CIFAR-10 dataset for our *Vonc* model, trained for 250 epochs, with similar other parameters as in Tab. 1.

Compared to state-of-the-art results on CIFAR-10 [5], these are not good results, but this allows us to compare our basic CNNs with more powerful pretrained models, such as VGG16 [15] (denoted VGG later), and see their differences. In particular, we compare their feature maps to figure out why they perform better.

3.2. Visualization of feature maps and qualitative analysis

For this part of our experiments, we have built the DeconvNets associated with each of the networks we have studied, using the code for unpooling and deconvolution layers provided by Mihai Dusmanu [2]. The visualization of some feature maps –more accurately, of one feature map for some intermediate layers, ranked in order of increasing depth– of VGG is shown in Fig. 2. For each image displayed in the figure, we have run the forward model on the input image (top left image) up to the considered layer, and then applied the associated DeconvNet back to the pixel space. Edges and color patches seem to be detected in the first convolutional layer (top center image). As we go deeper in layers (from left to right, from top to bottom), the layer outputs become more and more abstract, and represent higher-level features, such as patterns and textures. In the feature map of the final convolutional layer (bottom right image), the cat head region is the most activated with respect to the remaining zones, from which we infer that it is the face of the cat that determines essentially the class for this image, thus influences most the classifier decision.

In contrast, we did not observe the same phenomena in visualized feature maps of *Conv*, which performs much worse than VGG (see Tab. 1). In Fig. 3, we do not observe the gradual abstraction of the information conveyed by the layers as they get deeper (from top to bottom, from left to right). Even in the feature map of the final convolutional layer (bottom right image), a merely rough segmentation of the image (cat/background) may be noticed, which does not seem to give key information for classification.

3.3. Reconstructed inputs *via* Gradient Ascent and quantitative analysis

In order to perform gradient ascent (Alg. 1), we have used the code from [1] and [17], as described in Sect. 2.2. We perform the following experiment: considering a pretrained model, we consider the dataset of pictures of Siamese cats (label equal to 284 in ImageNet [4]). Using weights from pretraining, we gather a few ($n = 5$) reconstructed inputs using the gradient ascent method. Then the model undergoes a training process using only images from the considered dataset, and consequently, its weights are updated. Then we gather one more time n reconstructed inputs from the model with updated weights, and we aim at comparing the two groups of reconstructed inputs (see Fig. 4). In order to obtain quantitative results from this comparison, we have used three classic image matching methods to try to match reconstructed inputs with the training images from our dataset. We have used SIFT [9] and

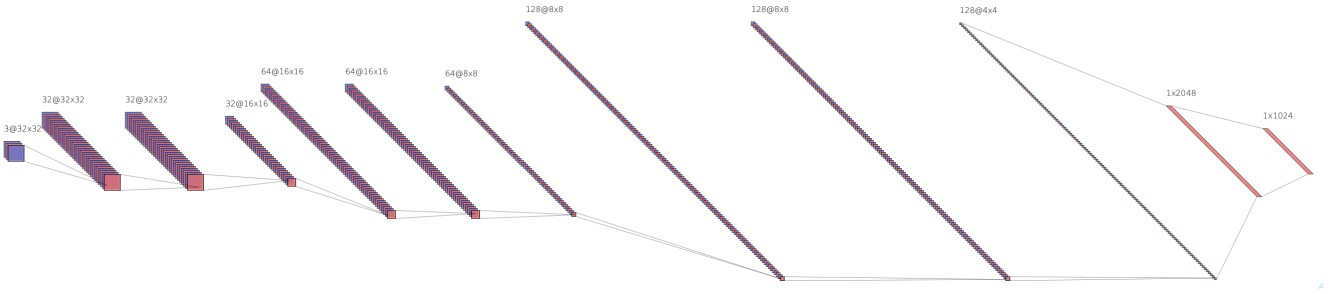

Figure 1. Architecture of *Conv*

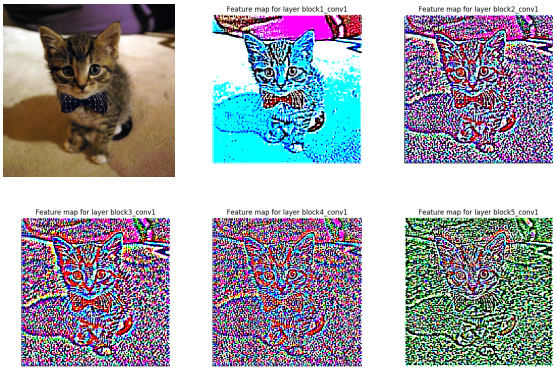

Figure 2. Input and feature map images of VGG for some convolutional layers, applied on an input cat image (top left image): output of each layer in the initial model is deconvoluted.

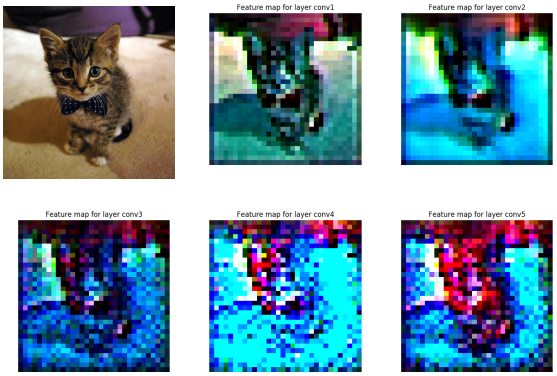

Figure 3. Input and feature map images of *Conv* for some convolutional layers, applied on an input cat image (top left image): output of each layer in the initial model is deconvoluted.

Harris corner [6] descriptors (implementation from scikit-image [18] –matching between descriptors is performed using Euclidean distance and RANSAC match filtering) to try to assess the contributions of each training image to a given reconstructed input after re-training (see Fig. 5), and a Bag-of-Words analysis [16] (that we have re-implemented by ourselves) to observe whether reconstructed inputs were closer (in terms of the cosine score computed on their respective histograms) to images from the dataset after re-training (see Tab. 3). Although these methods are usually meant for real, natural images, we have thought that resulting values and plots could give insights about the preservation of interesting features in computer vision, such as corners and visual words.

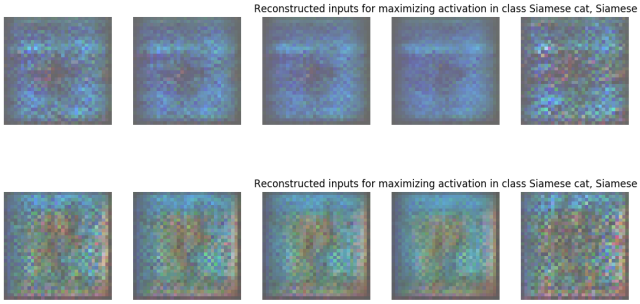

Figure 4. Two groups of reconstructed inputs: the top line displays reconstructed inputs (before re-training) from model *Conv*, whereas the bottom line comprises of reconstructed inputs after re-training of model *Conv*. Parameters for re-training: Adam optimizer, with decay rate equal to 10^{-6}, learning rate equal to 10^{-3}, for 10 epochs (there is a small number of images here in the dataset).

	Mean max score	Median max score
Before training	0.050	0.050
After training	0.095	0.090

Table 3. We display the values of the mean and median maximum score (that is, the maximum cosine score between the histograms of a reconstructed input and of an image from the dataset) overall $n = 10$ experiments. We assume that if the maximum score increases after re-training, this means that the reconstructed input after re-training share more patches ("visual words") with at least one image from the dataset than before the training process. Although the score values are still quite low (this might be normal, because the model *Conv* on which we have run the experiments has not been trained much), they have increased almost two-fold after re-training, which might account for a significant change.

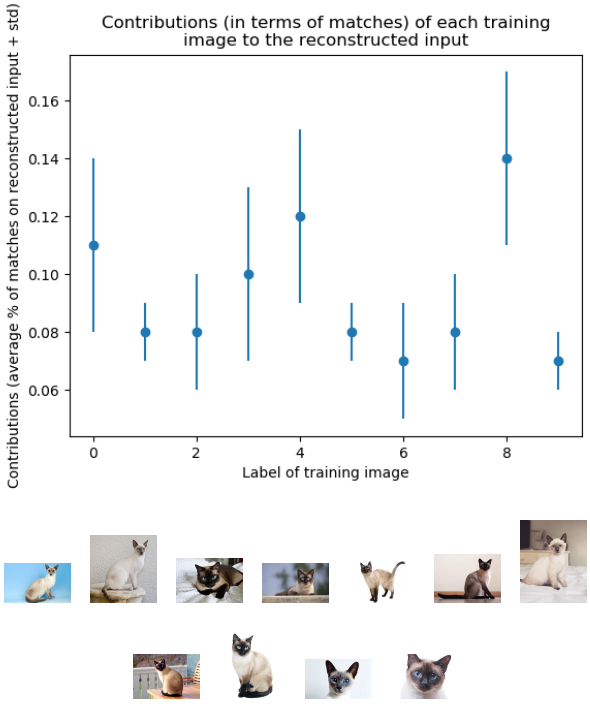

Figure 5. Plots of contributions of the first 11 images in the training dataset of Siamese cats (numbered 0 to 10 from top to bottom, from left to right, displayed below the plot). For each round of the experiment, one reconstructed input has been generated using gradient ascent on model *Conv*, then Harris descriptors of the reconstructed input along with all images from the Siamese cat dataset have been computed. Resulting contribution score between descriptors of a given reconstructed input and an image i of the dataset is equal to the ratio of the number of matches between descriptors of the reconstructed input and of image i over the total number of matches between descriptors of the reconstructed inputs and all descriptors from dataset images. This experiment has been iterated $n = 10$ times, yielding associated mean and standard deviation values with respect to every image from the training dataset.

4. Conclusion and perspective

In summary, we have trained two small CNNs on CIFAR-10 for our comparative analysis. We have visualized the feature maps of our CNNs and those of VGG and compared the amount of information displayed there between networks, and between layers of different depth. We have also reconstructed images using activation maximization, and used them to perform quantitative analyses. The results of the feature map visualization have shown the necessity of training through enough epochs, and the usefulness of a deep architecture in order to capture higher-order, more abstract, information in images. Our quantitative analysis seems to show that well-trained CNNs are able to extract the relevant information for image classification, as more characteristic images of a given class are more present in the network filters. For future work, one could apply our quantitative method on GANs, and study more precisely the weights of the discriminating network, in order to assess the type of information which is learned from its interactions with the generative network. It might also be more appropriate to use other types of descriptors than SIFT and Harris corner descriptors, in order to account for other interesting, and perhaps more robust to noise, image features.

References

[1] F. Chollet. How convolutional neural networks see the world. `https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html`, 2016.

[2] M. Dusmanu. Keras layers for pooling and unpooling (zeiler and fergus' paper). `https://gist.github.com/mihaidusmanu/5b4685ead7462c77aee923e75aeb689f`, 11/27/2018.

[3] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. Technical report, 2009.

[4] GitHub. imagenet 1000 class id to human readable labels. `https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a`, 12/20/2018.

[5] B. Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[6] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[9] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[10] A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: Going deeper into neural networks. `https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`, 11/27/2018.

[11] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.

[12] S. Plotka. Cifar-10 classification using keras (tutorial). https://blog.plon.io/tutorials/cifar-10-classification-using-keras-tutorial, 2017.

[13] C. Réda and X. Xu. Our code repository for the project. `https://github.com/kuredatan/nn-visu`.

[14] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualizing image classification models and saliency maps. arXiv, 2014.

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

[16] J. Sivic and A. Zisserman. Efficient visual search of videos cast as text retrieval. *IEEE transactions on pattern analysis and machine intelligence*, 31(4):591–606, 2009.

[17] Tensorflow GitHub contributors. Deepdreaming with tensorflow. `https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/deepdream/deepdream.ipynb`, 01/15/2018.

[18] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014.

[19] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.