Visualizing and Understanding Convolutional Neural Networks RecVis18 Report

Eloïse Berthier
eloise.berthier@polytechnique.edu

Clément Mantoux
clement.mantoux@polytechnique.edu

Abstract

We follow the reasoning from M. D. Zeiler and R. Fergus [6] to visualize the role of individual feature maps in deep convolutional neural networks. We highlight the complexity of the information gathered in deep layers and take stock on the VGG16 network [5]. We use deconvolutional neural networks and mask occlusion to build and improve a new network trained on the CIFAR-10 dataset [3], and we investigate the structure of the network's layers.

1. Introduction

Understanding convolutional neural networks (convnets) is one of the most important current challenges in the area of computer vision. The authors in [6] proposed a novel approach to this problem. They use deconvolutional neural networks, introduced in [7], to visualize individual feature maps in a convolutional layer of a convnet. This approach differs from previous works like [4] or [1] because their method applies to all the convolutional layers.

We first show the relevance of the visualization technique on a VGG16 network (see [3]) and we study the network's information structure. We move on to the design of a new convnet trained on the CIFAR-10 dataset, introduced in [5]. We improve the network using the deconvnet feedback. We check that the network's inference is based on the object we want to identify, and not on its context, using the mask occlusion technique from [6].

2. Deconvolutional networks for convnet feature visualization

2.1. Building a deconvnet from a convnet

Deconvolutional neural networks (deconvnets) are introduced in [7]. They process an image in the reverse way of a convnet. In our context, a deconvnet is built along the convnet of interest. Just like when inverting an operator, each layer in the convnet yields a layer in the deconvnet (we do not consider dense layers). The reverse operators look similar to the original ones :

The convolution is replaced with a deconvolution which is performed by using a transposed convolutional operator. We keep the same weights and bias in the deconvnet related to the convnet.

The pooling operation is replaced with an up-sampling. When performing the regular maxpooling in the convnet, we record the position of the maxima in each averaged cell (thus keeping the argmax). The un-pooling is obtained by placing the maxima at the locations given by the switches.

The activation function used in the convnet is the Rectified Linear Unit (ReLU). The related deconvnet layer is also a ReLU function.

Unlike convnets, deconvnets increase progressively the image size. They take a collection of feature maps and project it back into the pixel space.

Implementation We implemented[1] a tool to build the deconvolutional network related to a given network. To that end, we used the code snippets provided by our teaching assistant for the ArgmaxPooling, Unpooling and Deconvolutional layer structures. We used it to visualize specific feature maps in various convnets, as explained in the next section.

2.2. Visualizing convnet features

In [6], the authors propose to visualize the role of individual convnet feature maps. To that end, they build a deconvnet along the convnet and begin at the selected filter's layer. They feed the output of that convnet layer into the deconvnet, except that all the feature maps but the one of interest are set to zero. Thus the output image represents the stimulus which activated the selected filter.

The authors show that deep filters produce very complex invariances, like text or face detection. Following their approach, we analyze the filters of the VGG16 network.

[1] Our implementation and experiments are available at https://github.com/eloiseberthier/ConvNet_Viz.git

2.3. Features structure in VGG16 network

The VGG16 network introduced in [5] is a deep convnet trained on Imagenet. It achieves a 7.4% top-5 error on the ImageNet test dataset which was a state-of-the-art performance when the article was published. This result proved that conventional convnets could achieve good results by increasing the network's depth.

The network is built with five blocks and three fully connected layers. Each block consists in two or three convolutional layers with 3×3 filters and a max pooling operation with stride 2.

We used sample images figuring a cat and a car to demonstrate the filtering process operating in the network's layers. The result is shown in figure 1. The first filters perform simple operations like edge and contour detection or low-band filtering. Higher layers become more specialized, and the feature maps activate only on more and more complex patterns like wheels and faces. The deconvolutional projections look kind of similar inside of a block, hence it seems that the max pooling operation also plays an important role in the information processing flow.

3. Training a network on CIFAR-10

In this section, we design and train a convolutional neural network on CIFAR-10, while monitoring learned weights with our visualizations.

3.1. Number of filters

We begin with a simple four-convolutional-layer neural network, with 3×3 filters and a total of 120k parameters. This yields a test accuracy of about 58%. However, the filters visualized by deconvolution are very blurred, very similar among the different layers, and no filter strongly dominates on any layer. Furthermore, the most activated areas on the images are frequently located in the background. Obviously, this explains the poor performances of the network.

Hence we increase the number of filters at each convolutional layer, and the number of neurons on the fully connected layer, keeping the same general architecture, and adding dropout for regularization. This model has 776k parameters and achieves an accuracy of 64.4% after 40 epochs.

3.2. Batch normalization

After visualization of the filters, we noticed that many filters appeared as blue. Indeed, even if the original image had no blue areas, the deconvolved image was always biased towards blue pixels (see left of figure 2).

Adding batch normalization [2] at layer 2 partly solved this issue. Batch normalization introduces an independent normalization at each layer. In our experiment, the colors of the filters were more balanced (see right of figure 2). Furthermore, it allowed a significant speed-up of the training process and also a great performance gap. This model achieved 80.6 % accuracy after 40 epochs.

3.3. Padding

Filters from the first convolutional layer sweep through the whole input image. The image is transformed by each filter into a new image, slightly smaller. It is not of crucial importance in general, but since the resolution of the images is very low (32×32), we still loose part of the information.

Adding zero padding (*same* padding in Keras, conserving the size of the image) at the first convolutional layer reduced the border effects. In figure 3, the sky at the top of the image was lost without padding, but kept with zero padding. Even though border information is seldom relevant, this allowed a slight accuracy increase to 80.9%.

3.4. Evolution of filters during training

Following the experiment led in [6], we display the evolution of one fixed chosen filter at each layer, for a fixed input image, during training (figure 4). Filters from deeper layers are modified later in the training process, while filters from the first two layers stay almost fixed after a few epochs. Unlike VGG16 on ImageNet, deep filters do not exhibit interpretable features from the images.

Hence deeper filters are more difficult to train. This motivates a careful search on the number of filters in the last two layers, that we achieved with a grid search (see figures 6, 7 and 8 at the end of the report). The model that performs best has 998k parameters and 82.9% test accuracy. Yet the model with 256 filters on the 3rd layer and 64 on the 4th layer achieves almost the same performance (81.9%) with only 588k parameters.

Unfortunately, due to the small size of the images, we do not observe any specific trend throughout the third and fourth layers size variations. This is partly caused by the checkerboard pattern mentioned hereafter, which hinders from clearly visualizing features in small convnets.

Remark We observe a checkerboard pattern on deep filter visualizations. This is a consequence of the combination of ReLU activations and MaxPooling. We observe a similar phenomenon for VGG16 on figure 9, but we don't get to see them in the final result because of the higher resolution of the input images.

3.5. Occlusion sensitivity

Once the final model was trained, we checked that it used meaningful parts of the images to classify them. This can be done by the sensitivity analysis also introduced by [6]. We implemented this method from scratch. It consists in sliding a gray square patch across the image, and making a prediction for each square location. If the prediction is

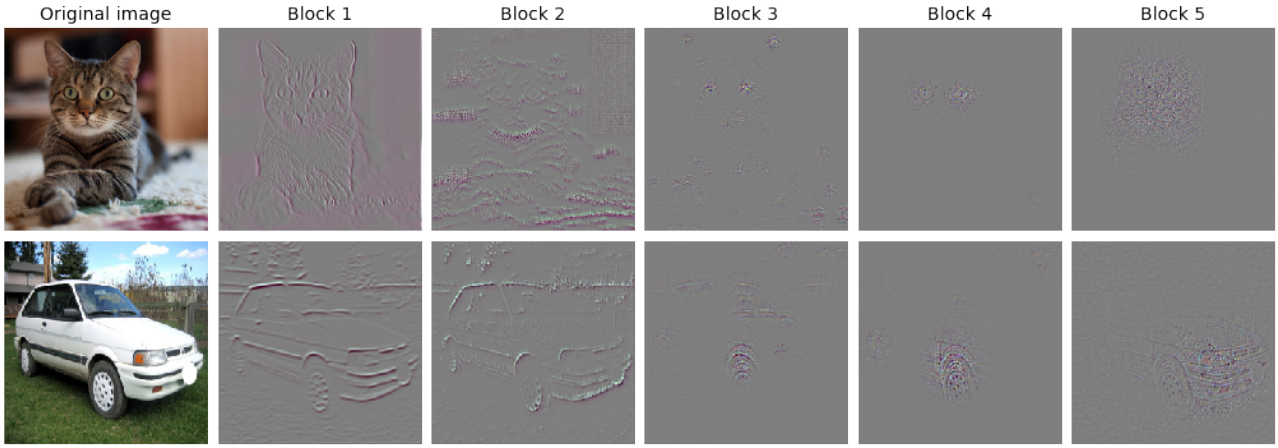

Figure 1. VGG16 deconvnet outputs. We took the output of the 5 convolutional blocks. For each block, we selected an interesting feature map. To that end, we computed the maximum value of each individual feature map. We used these values to rank the feature maps and displayed a relevant one among the 4 highest maps. The complexity of the information captured by the filter increases with the depth. For the cat, the highest activation is for the edges at the first block, then around the eyes, and for the entire face at the last block. For the car, the network progressively detects the wheels and the headlights (on other filters, not displayed here), and ends with the whole front part of the car.

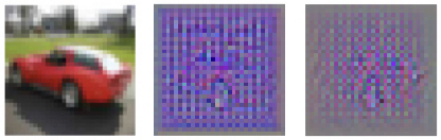

Figure 2. Most activated filter of layer 4.

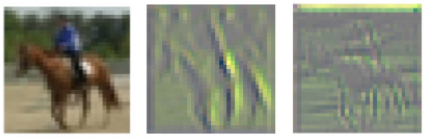

Figure 3. Most activated filter of layer 1.

affected at a given position, it means that it contains useful parts for the model. We used a patch of size 7×7, covering 5% of the image. In figure 5, we plotted the predicted class and the score of class *dog*. Both experiments confirm that useful areas for classification are meaningful, as they correspond to the true object and not to the background.

This occlusion experiment, although computationally expensive, can as well be applied to VGG16 on ImageNet. The highlighted areas are often part of the object, but are sometimes in the background. It also reveals the vulnerability of such models towards basic adversarial attacks.

4. Conclusion

Deconvolutional networks are powerful tools for convnet visualization. They proved helpful to design a convnet architecture from scratch, giving interpretable insights on the role of specific elements in convnets, like batch normalization or padding. Visualization and mask occlusion proved powerful when it comes to make sure that the network learned the desired representation. Deconvnet visualization thus contributes to the research towards making neural networks more interpretable.

References

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

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

[3] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[4] J. Ngiam, Z. Chen, D. Chia, P. W. Koh, Q. V. Le, and A. Y. Ng. Tiled convolutional neural networks. In *Advances in Neural Information Processing Systems 23*. 2010.

[5] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

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

[7] M. D. Zeiler, G. W. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. *2011 International Conference on Computer Vision*, pages 2018–2025, 2011.

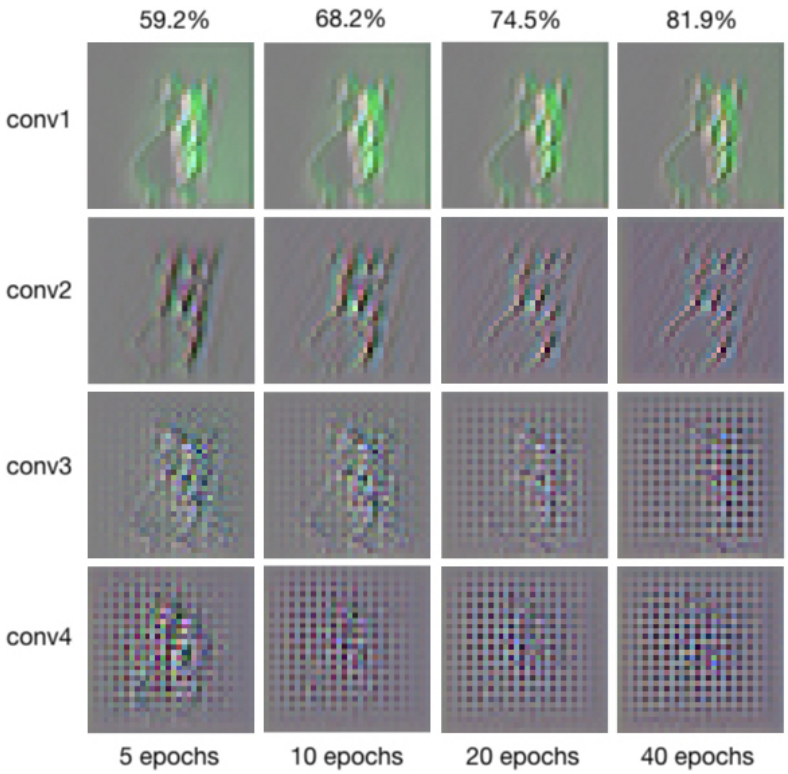

Figure 4. Tracking of one filter at each layer during training (the input image is a dog on a dark background, not displayed here). On top is the accuracy achieved after a given number of epochs. We see that filters from early layers stabilize very fast, whereas deeper filters still evolve between 20 and 40 epochs.

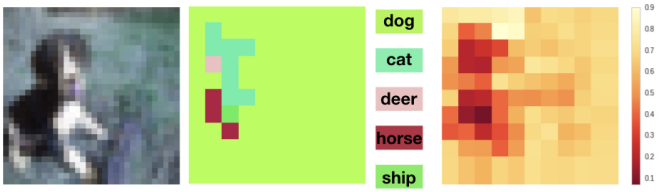

Figure 5. Occlusion sensitivity. The last two images show the predicted label and the score of class *dog* at each patch position. The network's performance clearly drops when the sliding patch covers the dog, which proves that it does not use other parts of the image to achieve its classification.

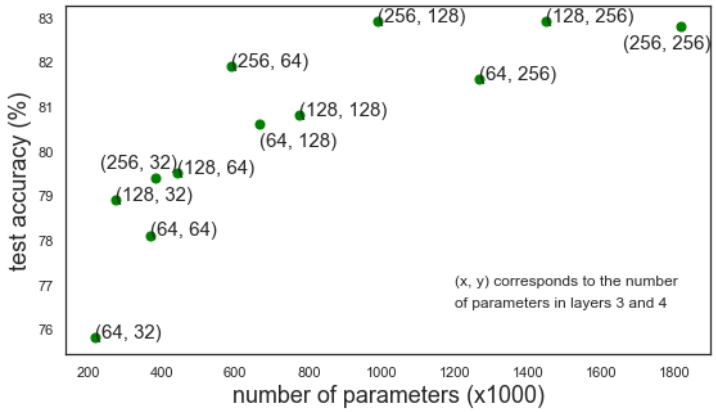

Figure 6. Grid search on the number of filters of the last two convolutional layers. All networks have 32 filters on layer 1 and 64 on layer 2. Interestingly, the network's performance is tightly correlated to the number of parameters. We also notice that the performance on the test set stops improving after a certain threshold, thus providing a good criterion for architecture selection.

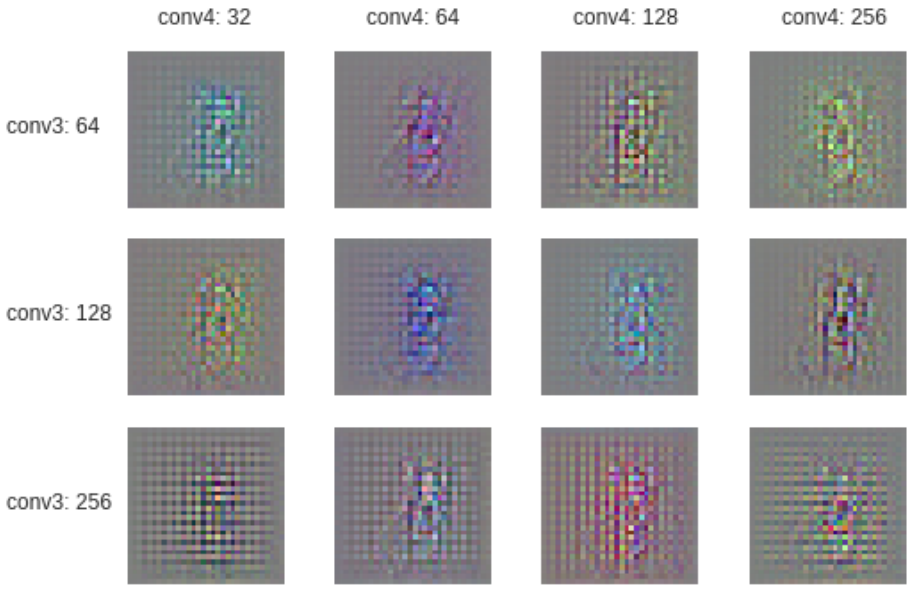

Figure 7. Most activated filter on layer 3 for each model of the grid search.

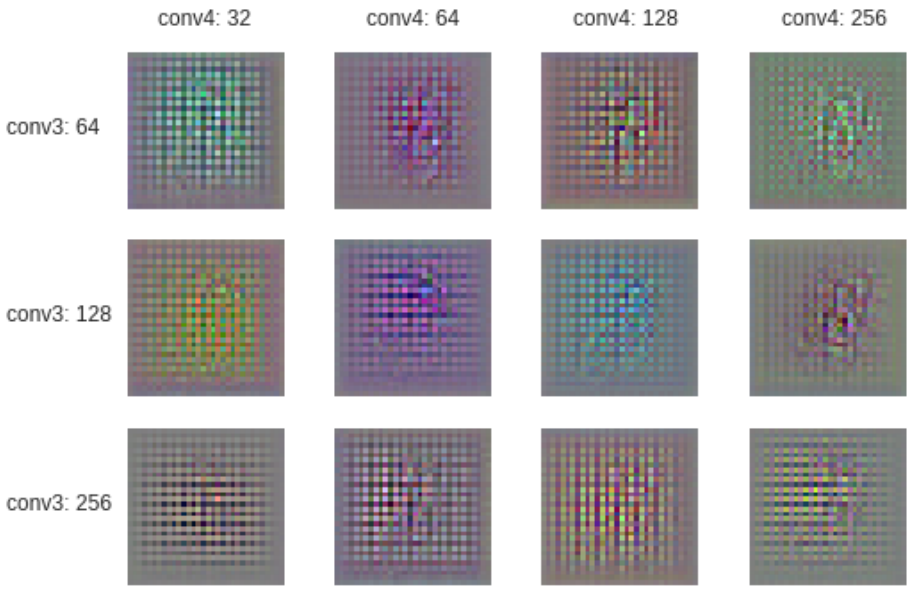

Figure 8. Most activated filter on layer 4 for each model of the grid search.

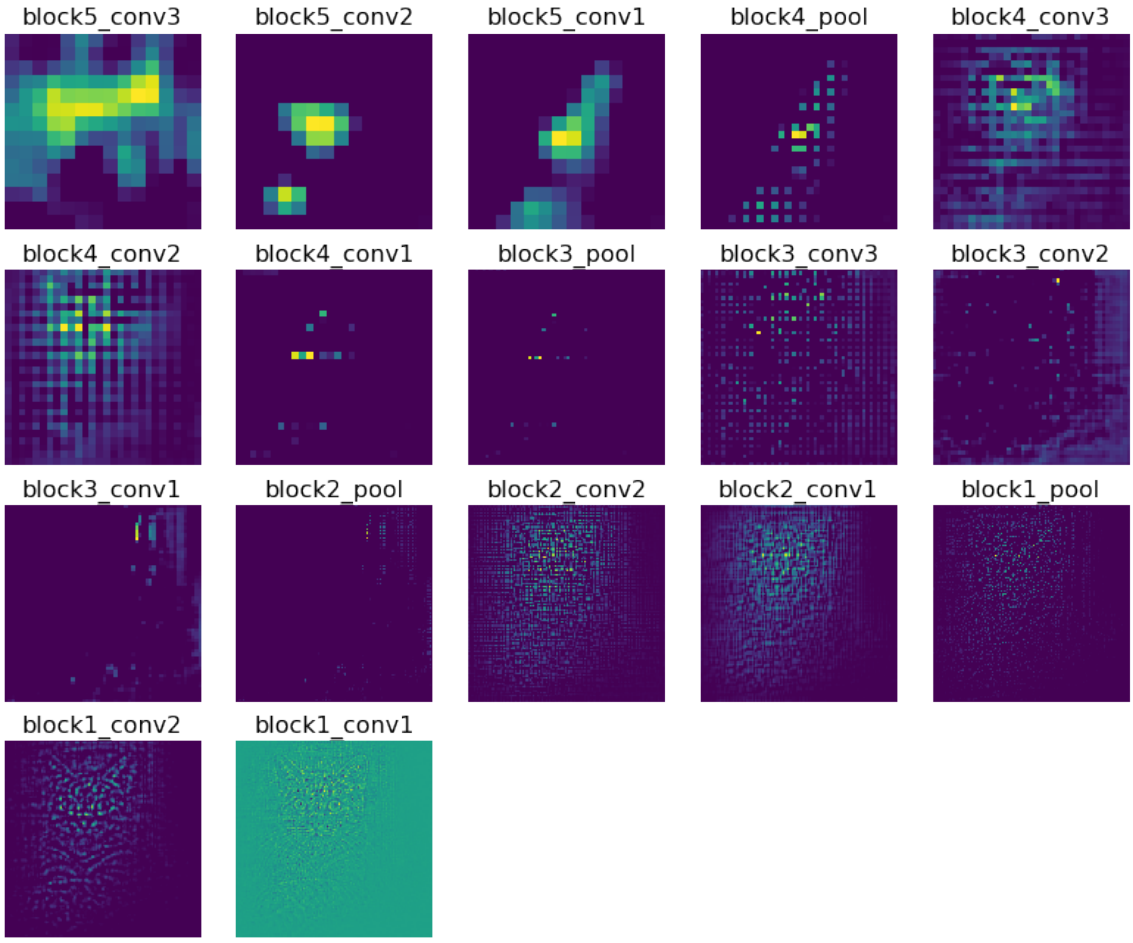

Figure 9. Visualization of random feature maps in VGG16. We feed our cat image into VGG16, and then plug the output back in the deconvnet. We plot a random filter at each layer of the deconvnet to see the progressive projection into the pixel space (the plots are ordered left to right, top to bottom). We clearly see that the checkerboard effect comes after the pooling layers. It is in fact still present in the final picture, but the image resolution is so high that it does not affect the structure of the image. This explains why deconvnets work better for ImageNet than for CIFAR-10.