# Visualizing and understanding convolutional networks

Nicolas Cloarec
ENS Paris-Saclay
91 avenue du President Wilson
94230 Cachan
nicolas.cloarec@ens-paris-saclay.fr

Tom Rousseau
ENS Paris-Saclay
91 avenue du President Wilson
94230 Cachan
troussea@ens-paris-saclay.fr

## Abstract

*Deep learning has proven successful at object recognition and classification tasks for the last few years. As an alternative to model-based techniques, convolutional neural networks offer promising performances on a wide range of image datasets. However, as a black box approach, the resulting learnt features lack interpretabiliy and often fail to process other datasets. This document review a feature visualization technique designed by [4] to perform a sensitivity analysis of the layers of a given CNN, thus providing a better understanding of its behaviour and relevance for a given dataset. We first detail the proposed method and the DeconvNet CNN it uses. We then present how we tested this technique on a custom CNN architecture trained on the Tiny ImageNet dataset before discussing our results.*

## 1. Introduction

Deep learning has become a major topic since the design of AlexNet. Convolutional Neural Networks (CNN) and deep learning have arisen as a convenient alternative to more traditional methods based on object models or hand-crafted features. Since its initial success on the ImageNet dataset, this technique has been successfully applied to other datasets and has quickly drawn the attention of the computer vision community. However, deep learning faces a number of challenges. Firstly, the design of CNNs mainly relies on some heuristic rules. Successful CNNs are radically different from an image dataset to another and there still exist no convincing and scalable alternative to a trial-and-error design. Moreover, as a black box approach, deep learning suffers from a lack of interpretability which is highly undesirable. Furthermore, CNN's have generally shown poor scores when applied to another dataset than that for which they were initially designed, phenomenon commonly reffered as a lack of generalizablity. [4] proposed a feature visualization approach to face these recent challenges. It consists in performing a sensitivity analysis on isolated features in the layers of a given CNN to provide insights on the learnt features and to identify possible improvements. The visualization is performed using another CNN, reffered as DeconvNet, which can be seen as the 'reverse' architecture of the CNN to be investigated. We first detail the proposed method and the DeconvNet architecture used for this purpose. We then review how we used this technique to explore a custom CNN architecture trained on the Tiny ImageNet dataset before presenting our results.

## 2. Feature visualization using DeconvNets

The method proposed by [4] aims at performing a sensitivity analysis on isolated features from a given layer of a given trained CNN. For this purpose, the feature maps considered are mapped from their layer back to the input image space. Because the core of CNNs is mainly composed of convolutional layers, max pooling steps and non-linear activations, the input space mapping relies on the definition of the reverse of the previous operations. Note that [4]'s transformation requires CNNs containing 2D convolution/max pooling and ReLu activations exclusively.

The reverse operations introduced are as follows. Firstly, the transposed kernel is used to revert a given filter, which is a common, albeit questionnable choice for deconvolution. Secondly, layers prior to a max pooling steps are recovered assigning the output values of the max pooling to the previously-stored locations of these maxima. This operation, reffered as 'unpooling', preserves the maxima but loses the rest of the information available before max pooling. Lastly, to enforce positivity of all outputs in the back-mapping process, ReLUs are inverted by ReLUs.

To perform the input space mapping, the DeconvNet should first be built by taking all the layers from the original CNN in reverse order (from the deepest to the shallowest until the input layer). The DeconNet should be provided accordingly with the weights/biaises of the CNN considered. The output of the layer of the CNN to be analysed should then be computed and provided as input 'image' to the DeconvNet. The DeconvNet eventually outputs an image with the same format as that of the CNN's input (sizes and number of color channels).

This procedure can be used to identify which pixel contribute most to the activation of a layer or a specific feature in the CNN and help improve its architecture. For instance, aliasing in this new image may indicate that excessive strides are being used. The absence of spectrally rich information (*i.e.* only low or high frequency components) may have been triggered by ill-adapted kernel sizes for convolutions. Vanishing activations after occluding the main object in a given image could mean that the CNN has actually learnt the background of the object instead of the object itself, which could also mean the architecture is not deep enough. Therefore, the sensitivity analysis proposed by [4] is expected to provided insights on what the CNN is actually learning and how to tweak its structure to improve its performances. The process can be performed while or after training, depending of the information sought.

## 3. Testing strategy

Our goal was to implement and test the DeconvNet approach. To achieve it, we first aimed at designing a CNN yielding acceptable scores on a dataset sufficiently difficult dataset. We then planned to perform a sensitivity analysis on our CNN, as described above, to identify both the information actually learnt and potential improvements.

### 3.1. Image dataset

The Tiny ImageNet dataset from Stanford was used for our tests. It contains 100,000 images equally splitted into 200 classes. The classes labels are noticeably varied and abstract, compared with ImageNet or other common datasets. As a consequence, achieving good error rates on this dataset is really challenging and [4]'s strategy potential benefits would be easily observed.

### 3.2. Training procedure

As Stanford did not provide the labels with the testing data folder, we splitted our training data into two subsets, one of which would be used for training ($90\%$ of the images) while the rest would be used for validation. The validation data folder was actually used for testing.

Batches of 32 images were chosen for the training. Categorical crossentropy was used as loss, and accuracy was chosen as metric. The optimization process was performed using SGD optimizer, with all weights and bias initialized to 0. 1000 epochs were chosen to let the biggest models be trained sufficiently. An additional patience of 20 epochs was opted for the training. Best models were automatically saved as `.hdf5` files during the training process.

### 3.3. CNN architecture

To facilitate the interpretation of the features mapped into the input space and to limit the computational efforts, a minimally-deep CNN was designed. Several architectures of our own were explored and benchmarked with a VGG-16 CNN.

The training was first performed with the VGG-16 CNN for further comparison. VGG-16 yielded around $46\%$ accuracy. However, its complex architecture (more than 40 millions parameters) would require expensive computations and would make the features more difficult to interprete. Thus, lighter structures were preffered.

Taking inspiration from common CNN, we then explored architectures similar to that represented in figure 1, with varying hidden layer sizes, activations, kernel sizes and strides. Following [2], the succession of the hidden layers were shaped as a pyramid, with hidden sizes given as descending powers of 2. As a first try, we tested a model containing only two convolutional layers (same as in figure 1 without the third conv. layer), without any dropout. The kernel sizes were chosen consequently to the images sizes, which are small ($64 \times 64$): $5 \times 5$ kernels with stride of 3 for the first layer was expected to be sufficient to harvest the majority of the information contained on the input image. This CNN showed poor performances after training ($\sim 17\%$ accuracy), regardless of the layer sizes (firstly 256 and 128, then 512 and 256). Therefore, the CNN was deepened with an extra convolutional layer as in figure 1. A data augmentation strategy was also used to prevent our network from overfitting and bring more robustness to the classification : horizontal/vertical flips, random rotation in range $[0, 360[$, random shifts ZCA whitening were preprocessed for both training and validation. First choosing 1024, 512 and 256 as hidden layer sizes for the conv. layers, we obtained reasonable scores around $32\%$, with excellent learning curves for the first 500 epochs. Nervertheless, despite our data augmentation strategy, signs of overfitting were detected afterwards. To reduce the complexity of the model, we reduced the sizes of the previous conv. to 512, 256 and 128 respectively. In addition, to provide more robustness, a dropout with probability $p = 0.5$ was included prior to the last convolutional layer, as suggested by [2]. The results obtained ($\sim 40\%$ accuracy) were sufficiently close to that of the VGG-16 to be used as a surrogate reference model for the sensitivity analysis. Our choice was also motivated by the significantly lower complexity of this custom model, expecting to improve the interpretability of the feature maps.

For unknown reasons, the confusion matrix we determined was not consistent with the behaviours we observed. Thus we chose to represent the scores more qualitatively with the learning curves. The CNN eventually used for the rest of our work is represented in figure 1 and its hyperparameters are listed in table 1.

### 3.4. Implementation

The DeconvNet and the sensitivity analysis were implemented using Keras (Tensorflow backend). We took inspiration from [1] to build our own code. The experiments were performed using Cuda 10.0 and CudNN 7.4.1 on a computer equipped with a i7 Intel Quad-Core Processor (4.2GHz), 64GB RAM and a 6GB GeForce GTX 1060 GPU.

## 4. Results

Deconvolution was successfully implemented and any whole feature map could be mapped back into input image space. The images obtained after the back-mapping procedure are closer to the original images.

Noticeable properties of our features can be observed. Firstly, as represented in figure 3 shallow layers clearly yielded images closer to the content of the original image. This phenomenon can be explained by the fact that given two layers, the deeper is computed after an increased number of max poolings/unpoolings and subsampling steps, yielding more precise but limited and spatially located information, whereas the shallower contains more diffuse and thourough information. Using convolution without padding (*i.e.* with valid computations) as we did eliminates the information available close to the borders, thus increasing even more the localisation of the activating pixels.

In addition, we observed that the feature maps of our CNN were all sensitive to large magnitude intensity gradients. As shown in figure 4, sharp intensity transitions are clearly kept in the last feature map of our CNN. This property holds for shallower layers.

Moreover, our features seems more sensitive to green and blue than to red. For instance, the saturated red patterns occuring in the top image in figure 5 lead to moderate activations in the last feature map of our CNN. In constrast, the edges of the turquoise car ($(R, G, B) = (154, 223, 222)$) and the yellow banana ($(R, G, B) = (212, 203, 172)$) both trigger higher activation levels. The strongest responses were obtained for contours separating black and white regions, as illustrated in figure 4, as they combine large shifts in intensity on the three color channels altogether.

Lastly, we observed that the presence of noise significantly degrades the image reconstructed from the feature maps. As an example, the blurred deer in figure 6, surrounded by hardly-fathomable textures yields a very noisy image from the third feature map, in which we hardly recognize any pattern from the original image. As a comparison, the visually clean picture of the chimney on a roof is properly recovered from the same layer. This phenomenon proves a lack of robustness of the features.

## 5. Conclusion

In this paper we proposed an implementation of DeconvNet and a preliminary sensitivity analysis inspired from [4] on our own CNN architecture. Several properties of its feature maps were identified, including hierarchichal information and sensitivity to gradients and blue/green objects. These provide a better comprehension of the learnt features and gives insights on how to improve the structure of the considered CNN and provide more robustness to the classification. Further possible work would be to explore the sensitivity of isolated features in the network, to build the DeconvNet with alternative inverse operations and to tune the hyperparameters accordingly.

Nevertheless, shortcomings of [4]'s DeconvNet approach can be mentionned. Firstly, the reverse operations used to build the DeconNet are questionnable and can be improved. For instance, even if using ReLU to invert ReLU is an acceptable choice, continuous non-decreasing activations such as sigmoid or $\tanh$ are unquestionnably invertible, with known inverses. Therefore, ReLU activations could easily be replaced. Secondly, the learnt features do not necessarily generalize well to other datasets. Eventhough we could not confirm this property on our own, [4] highlights it by bringing mitigated albeit promising results when tackling another dataset with their CNN unchanged. Thirdly, the method proposed by [4] do not provide clear insights about how deep a CNN should be. Lastly, even if the fine-tuning strategy deduced from the feature visualization seems intuitively well-grounded, [4] do not provide any figure concerning its global effect on the whole dataset. The classification may be improved on the samples observed but significantly degraded on already well-predicted images, even belonging to the same class. As only the overall performance matters [4]'s strategy can be argued.

## References

[1] M. Dusmanu. 2d deconvolution/unpooling in keras. `https://gist.github.com/mihaidusmanu/`, 2017. 3

[2] S. H. Hasanpour, M. Rouhani, M. Fayyaz, M. Sabokrou, and E. Adeli. Towards principled design of deep convolutional networks: Introducing simpnet. 2

[3] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps.

[4] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. 1, 2, 3

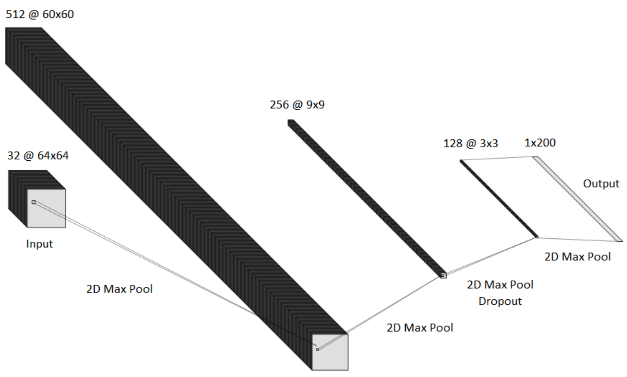
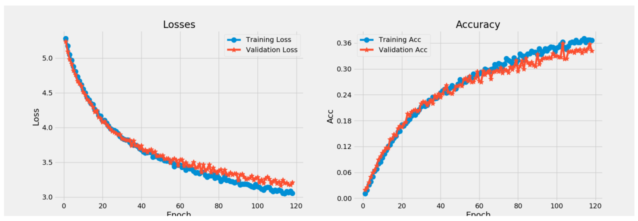


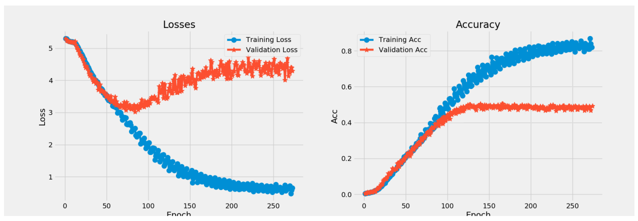
Figure 1: Architecture of the CNN used for the tests

| Layer | Kernel size | Stride | Layer size | Output size |
|---|---|---|---|---|
| Conv. 1 | $5 \times 5$ | 3 | 512 | $60 \times 60$ |
| Conv. 2 | $3 \times 3$ | 2 | 256 | $9 \times 9$ |
| Conv. 3 | $3 \times 3$ | 2 | 128 | $3 \times 3$ |
| Dense 4 | — | — | 200 | $1 \times 200$ |

Table 1: Hyperparameters chosen for our CNN

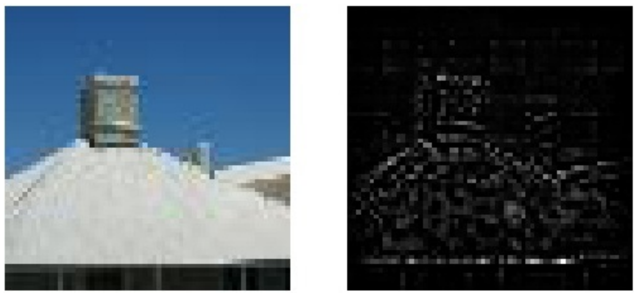


(a) Our model

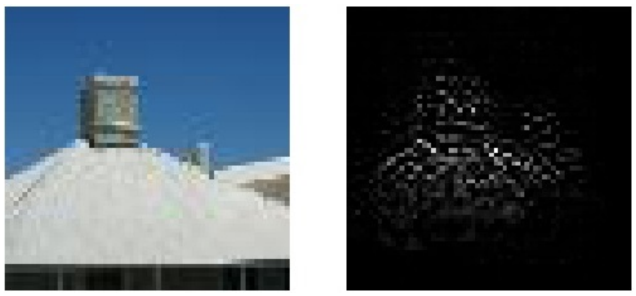


(b) VGG 16

Figure 2: Learning curves

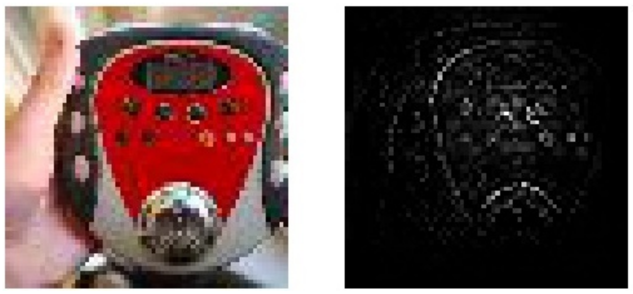


(a) Layer 2

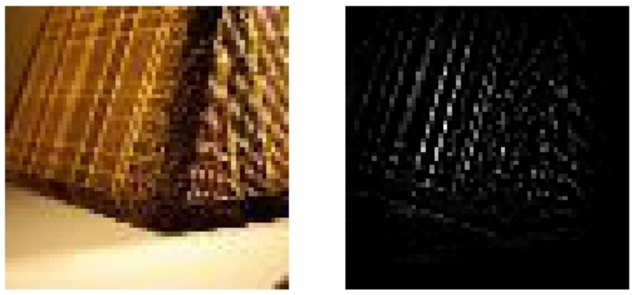


(b) Layer 3

Figure 3: Effect of the deepness of the feature maps

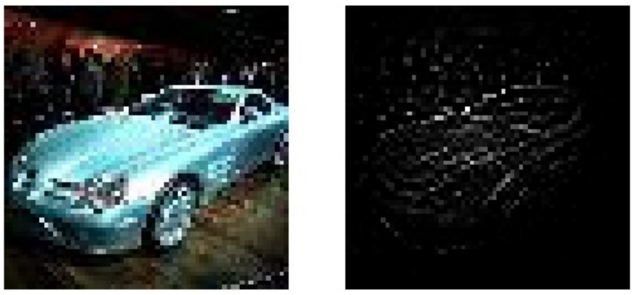


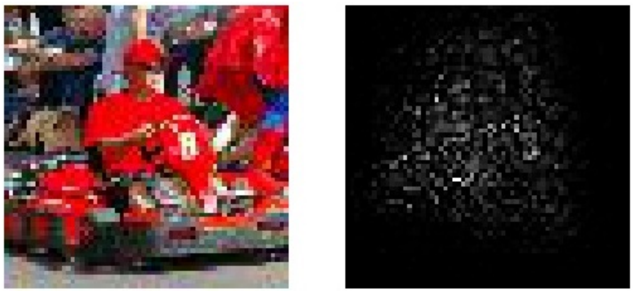
(a) Contours in complex object



(b) Texture-like content

Figure 4: Sensitivity of the third feature map to high magnitude intensity gradients


(a) Turquoise car (blue and green)


(b) Red

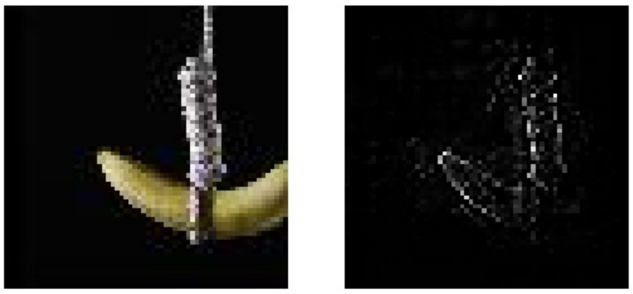
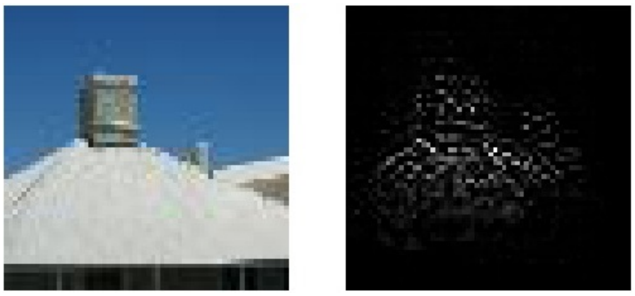

(c) Yellow banana (blue and green)

Figure 5: Sensitivity of the third feature map to colors


(a) Simple object, low level of noise

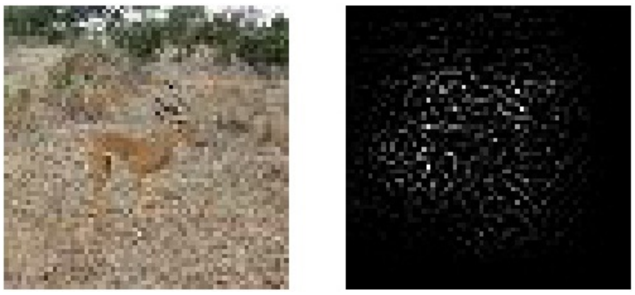

(b) Complex object, high level of noise

Figure 6: Effect of noise in the input image on the third feature map