Exploration of ImageNet pre-trained VGG16

Alvise Dei Rossi (# 2004250)
January 15, 2022
alvise.deirossi@studenti.unipd.it

1. Introduction

Deep learning models are often regarded as being black-box models, whose output is the product of too complex calculations to be comprehended easily. While this is partially true in some cases, plenty of methods exist to gain insights about the representations learned by deep convolutional neural networks and hence shed some light on the inner workings of these complex systems. The objective of this project is exactly this. The models used to do so are variants of VGG16[1], deep convolutional neural networks, pre-trained on the large dataset ImageNet [2]. ResNet50 [3] and Places365 [4] will also be briefly used as a comparison to VGG16 and ImageNet.

VGG16 is a famous convnet architecture which was used to win the ILSVR competition in 2014. While being an excellent model for image classification, its implementation is rather straight-forward: convolutional layers of 3x3 filters and stride 1 are followed by 2x2 max pooling layers of stride 2 throughout the whole architecture, followed by two large fully connected layers of 4096 units and a softmax layer for the 1000 classes of ImageNet . The complete architecture is shown in Figure 1. It's a pretty large network, with 138 milions parameters, most of them being in the last part of the model.

2. Dataset

Due to the large memory and computational requirements that using ImageNet would have imposed, subsets of this were used to perform exploration of the representations learned by the model. In particular, subsets of Imagenette and Imagewoof [5] were used: the first one is a dataset composed by 2000 pictures of 10 very different categories, while the second one is composed by 2000 images of 10 different dog breeds, so we can expect a large inter-class similarity. Details about the classes of the two datasets are reported in Table 1. Note that all these classes are part of the ImageNet dataset. All images were re-sized to 224x224x3 and pre-processed so to be in a format adequate to what VGG-16 pre-trained on ImageNet expect as input.

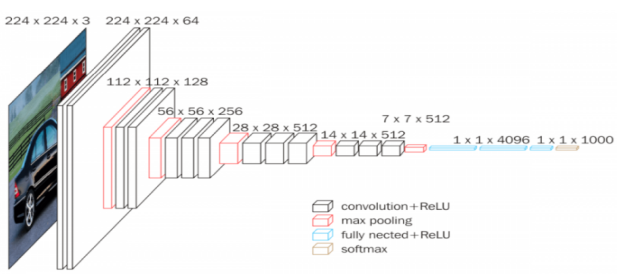

Figure 1. VGG16 architecture.

Label	Imagenette	Imagewoof
0	Tench	Australian terrier
1	English springer	Border terrier
2	Cassette player	Samoyed
3	Chainsaw	Beagle
4	Church	Shih-tzu
5	French horn	English foxhound
6	Garbage truck	Rhodesian ridgeback
7	Gas pump	Dingo
8	Golf ball	Golden retriever
9	Parachute	Old english sheepdog

Table 1. Class labels in Imagenette and Imagewoof.

3. Discussion

3.1. Read-outs

The first method used to explore VGG16 consisted in performing read-outs at different levels of the hierarchy of the model to understand if the representations learned by the model were increasingly separable for different classes going deeper into the network, hence obtaining better accuracy in a classification task.

Due to the structure of VGG16, it's easy to determine 5 different points where these read-outs can be applied (see layers in red in Figure 1), corresponding to the max-pooling layers were dimensionality reduction occurs along the first two axis. The complete pre-trained model (on ImageNet) was hence imported and cut at these points, freezing all

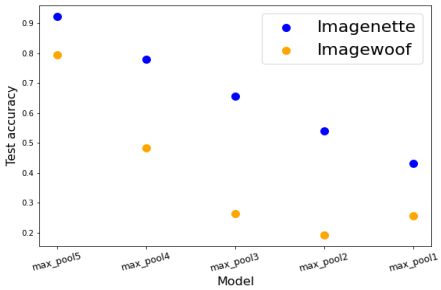

Figure 3. Test accuracies for read-outs at different levels of the network for Imagenette and Imagewoof datasets.

weights for the remaining layers. For all the 5 sub-models obtained, a simple classifier layer was added on top and trained for 10 epochs with a training set of 1500 instances, using early stopping to avoid overfitting, and restoring the weights of the model to the best performing version of the model, as measured by validation loss. Note that the weights of the model relative to the convolutional layers are unchanged in this procedure, since we want to determine what sort of representations are obtained by the pre-trained model; only the parameters of the added classifier change during training. The accuracy results over the test set for this procedure are reported in Figure 3 for both the datasets tested.

As we can see, in both cases the network is able to classify better for deeper representations (about 92% for Imagenette and 80% for Imagewoof). As expected, the Imagewoof dataset is intrinsically harder than Imagenette due to the fact that discriminating between breeds of the same general category (dog) is harder than separating totally different concepts, from a visual point of view. Applying read-outs at lower levels results in worse performances in both cases. This is expected, as deeper representation encode higher-level concepts which are more specific for each class. Due to inter-class similarity, this aspect has a larger impact for the Imagewoof dataset. In fact, while lower level representation are still able to achieve decent results for Imagenette, this is not the case for Imagewoof. Performance rapidly degrades to the point that, for the shallower models, which are only able to represent low-level concepts, we reach accuracies of about 20%, barely above chance-level.

A brief study of confusion matrices is shown in Figure 2 for the different levels of the read-outs for both datasets. For deeper representations (on the left), errors are rare and occur more for the Imagewoof datasets. As mentioned, this happens most likely due to inter-class similarity: note how many errors occur, for example, between classes "beagle" and "english foxhound", which indeed are rather similar breeds. As discussed, performance degradation is evident as we use lower level representations, particularly so for Imagewoof where it's evident that deeper, more complex representations are absolutely necessary in order to appropriately discriminate between classes.

3.2. Visualization of intermediate activations and maximum responses to filters

We can get a deeper understanding of what kind of representations are learned by VGG16 by visualizing the intermediate activations of the network. This gives an idea of how an input is decomposed by the different filters learned by the model. In order to so, every channel of a feature map is plotted as a 2D image, under the assumption that each channel essentially encodes a relatively independent feature. From an implementation point of view, a VGG16

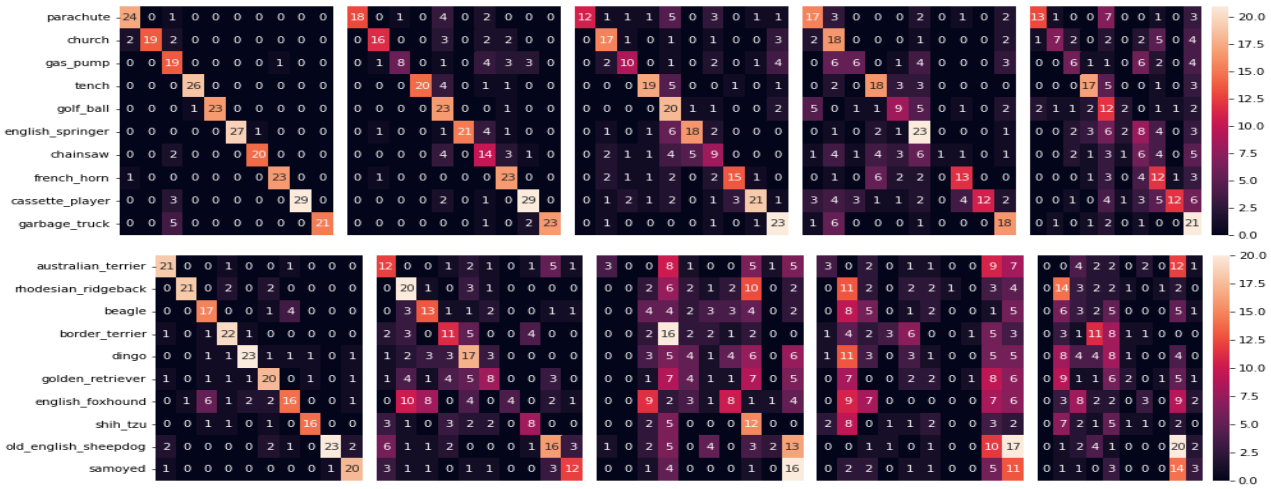

Figure 2. Confusion matrices of the results obtained for the read-outs at different levels (Imagenette top, Imagewoof bottom). Going from left to right the representation used for the read-out is taken from lower levels progressively.

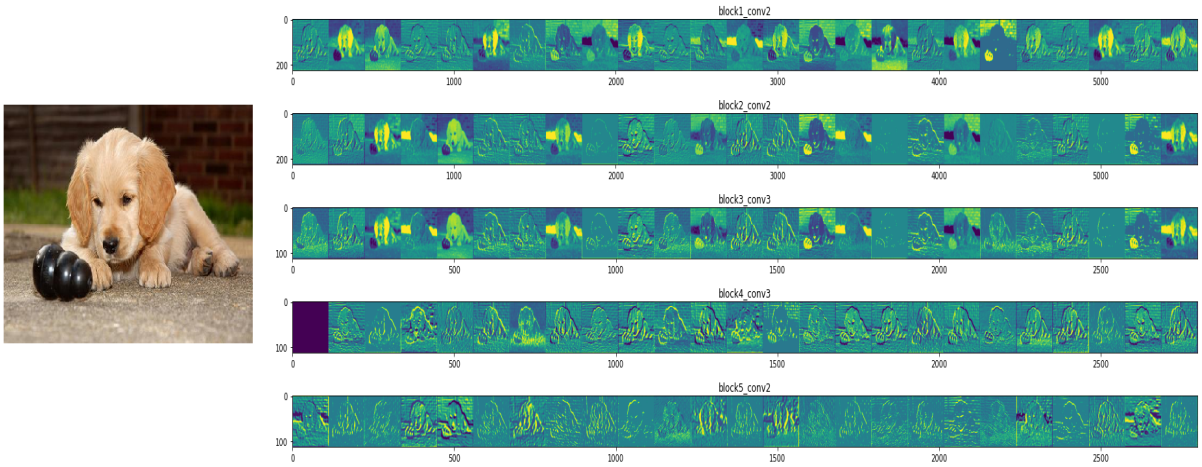

Figure 4. Visualization of activations for different lavels of depth into the network.

model can be instantiated with the ordinary input images as input and the activations of different levels of the network as multi-outputs. An example of the results obtained by this procedure is shown in Figure 4 for a puppy picture from Imagewoof. Note that not all filters are activated by the image. Looking at the first two rows we can gain an intuition over the fact that the first layers essentially act as detectors of features common to most images (e.g. color, edges), while going deeper the activations become more and more abstract, as they encode more class-specific concepts.

Similarly we can also check the filters learned by understanding which kind of input is going to maximize their responses, rather than feeding the network a specific input. This can be done by gradient ascent: a score function is provided so to maximize the response to a specific filter. An optimizer is then applied to an initially blank noisy image in order to generate the input desired. We can do so at different levels of the hierarchy of the model and for multiple filters. The results obtained from this procedure on VGG16 are shown in Figure 5. Once again the deeper we go into the network, the more complex are the images generated: for the first rows of the picture we can see that lower level filters encode edges and colors, while in the last rows deeper level filters encode increasingly complex shapes.

3.3. Deep Dream

Fairly similarly to the procedure just discussed, we can also use gradient ascent to maximize the activation of entire layers rather than a specific filter, by considering a score based on the weighted sum of the L2 norm of the corresponding activations, hence mixing together many features at once, but at the same level of the hierarchy. If we do so starting from gaussian noise we have a complete generative process (Figure 7). The output generated depends by the

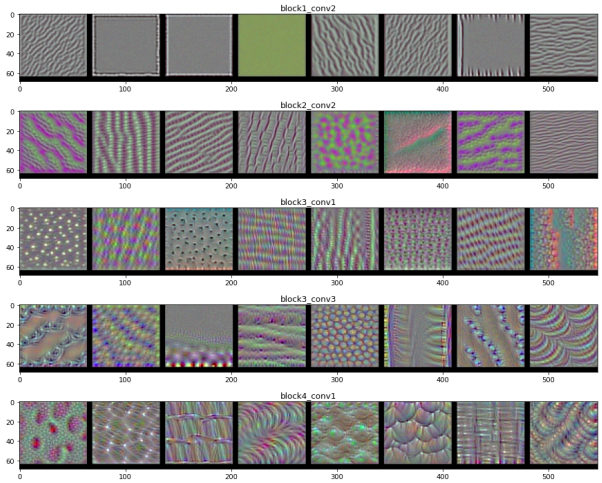

Figure 5. Patterns generated to maximize responses from filters learned, for different levels in the hierarchy.

layer chosen for the generation. Choosing layers low in the hierarchy results into geometric, colorful outputs, without any observable semantical meaning, while choosing layers deep into the network generate high-level features (e.g. eyes). If instead we start from a real image, we're amplifying the features detected by the network for that level. This is the idea of the DeepDream algorithm [6,7]. Once again, VGG16 pretrained on ImageNet was used in the implementation.

DeepDream produces psychedelic outputs, which, while certainly peculiar, also help to understand what the network sees, or is trying to see, in the pictures fed. The intensity of the generation process depends on several parameters of the implementation, one of which is the number of iterations of gradient ascent performed. In fact, initially the generation occurs due to the actual features of the image fed, ampli-

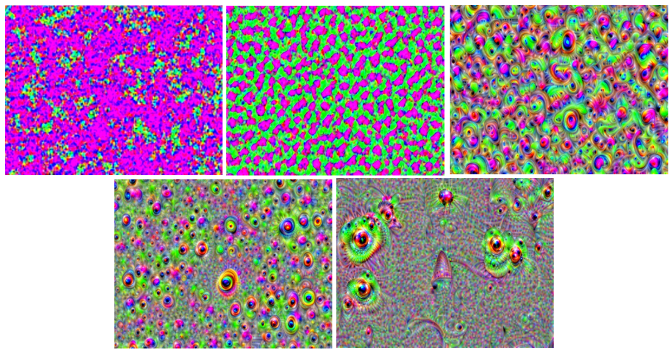

Figure 7. Patterns generated from noise by maximizing response at different levels of the model (from low-level to high-level layers).

fying the responses in characteristic areas, for example a dog's face, but if the process goes on for too long it feeds onto itself and resembles more and more a generation process from noise (Figure 6, first row, increasing number of iterations).

The result doesn't only depend on the input image or the implementation of the procedure: we can obtain different outputs depending on the depth of the layer chosen, the dataset used to train the network and the architecture of the network itself. As observed for previous visualization methods, for shallower layers we get low-level geometric and colorful patterns, while for deeper layers increasingly complex features emerge (Figure 6, second row). Regarding the dependence on the training dataset chosen, if a model is trained on ImageNet, like the VGG16 model used so far, we get a lot of animal-like features like feathers, eyes, ears and beaks: this is due to the fact that ImageNet has a lot of animal classes, in particular dogs and birds.

For sake of completeness, and due to the lack of readily available pre-trained weights other than Imagenet for a VGG16 model, an example of what can happen when we use another dataset is shown using a ResNet50 implementation and its pre-trained weights for ImageNet and Places365, a large dataset used for scene recognition. Differently from ImageNet, the categories of Places365 are composed entirely by natural and (mostly) artificial scenarios. This results in totally different learned filters, at least for high level layers. When the DeepDream algorithm is applied (Figure 6, third row) we observe a rather complex, geometric and structured output (on the right), resembling architectural features, street, lights and maybe railways. This is rather different when compared to the ImageNet case (on the left). Note also that, while similar, since we still observe eyes-like generation, the patterns we observe for the image generated with ResNet50 pre-trained on ImageNet are distinguishable from those generated by VGG16, underlying that representations learned by the network also depend on the architecture of the network itself.

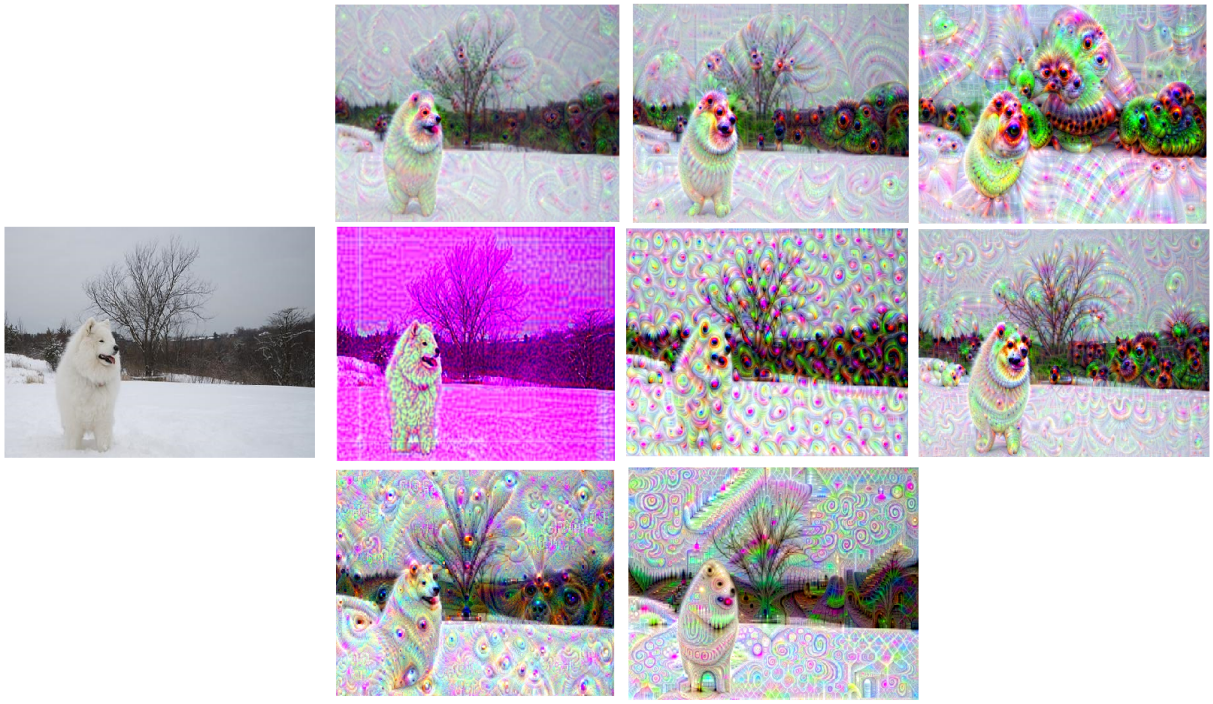

Figure 6. Pictures generated by the DeepDream algorithm for an Imagewoof picture. First row: increasing number of iterations of gradient ascent. Second row: increasing depth of layer chosen. Third row: ImageNet and Places365 pre-trained weights difference.

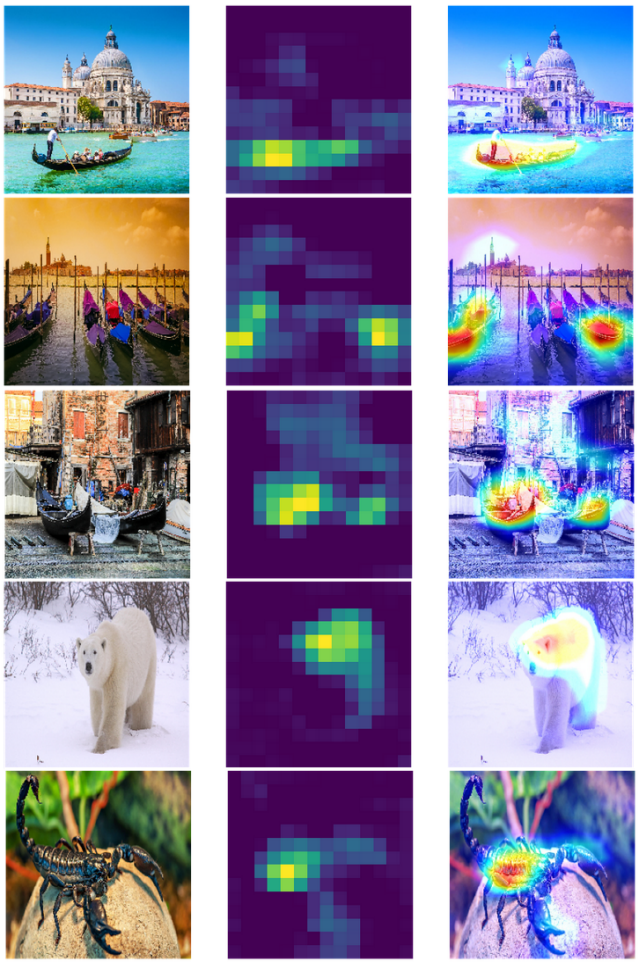

Figure 8. Examples of Grad-CAM procedure.

3.4. Importance score heatmaps over pixels

While the visualization procedures discussed so far give an idea of the type of features to which the different parts of the network respond, the classification reasoning by which the network operates appears to be still hard to grasp. Recently, the widespread diffusion of AI in a multitude of aspects of life and jobs has made clear that one of the main issues we face is the explainability of the decisions of AI systems, especially deep learning models. This is why, in the case of convnets at least, using visualizations which allow to directly understand the reason an output was given, is of great importance and may help, for example, to fight algorithmic bias.

One way to do so is to overlap heatmaps of the importance score of the pixels of an image onto the image itself. Essentially, in the case of classification, a 2D grid of scores associated with a specific output class is computed for every pixel, indicating its relevance for the decision. The procedure used to implement this visualization technique, called Grad-CAM, is described in Ramprasaath R. Selvaraju et al. [8]. Given an input and a chosen layer, we get the feature map of that layer (i.e. the intensity of the activations, given the input) and weight every channel of the feature map by the gradient of the target class w.r.t. that channel, which is a measure of its importance w.r.t. the class.

By applying this procedure, we can spot which part of the picture leads the network to understand that the output desired is, for example, "Gondola". As mentioned, this is useful when we want to check for biases in our system: are we sure the network is not recognizing a class due to its surroundings rather than its own features? For a gondola, we might think that the network is recognizing the boat due to the presence of the gondolier or to the fact there's water around. But as shown in the first three rows of Figure 8, this is not the case for pretrained VGG16. The model gives the right output in every case. Hence we can state that, at least for the scenarios tested, the network is robust. We can also try to understand which are the features most representative of a class. What characterizes most a polar bear? Apparently its ears and its overall posture. What about a scorpion? The claws and the tail? They're important, but so is its back (last two rows of Figure 8).

We can apply this procedure also for Imagenette and Imagewoof. It's particularly interesting to do so for layers at different levels of the hierarchy to understand which characteristics are considered most important for the classification as we change the representation used. The five VGG16 submodels discussed in the linear read-outs section are hence used for both datasets. The results obtained are shown in Figure9: a tench picture is chosen for Imagenette while a golden retriever image is chosen for Imagewoof. The pictures are fed to the respective models, then the Grad-CAM method is applied, outputting a heatmap, which is finally imposed onto the images.

In the case of the tench picture chosen, the models are able to classify it correctly for all the representations chosen. For the deepest representation the network is focusing entirely on the fish, but it's clear that as we choose lower level representations (from left to right), the model is focusing on the context of the picture, as the man holding it and the grass around (which are both very common features of the tench images in the dataset), rather than the tench itself. This may suggest that when we consider representations obtained by lower levels we might misclassify pictures with classes that are commonly occurring in a similar context or that share similar low-level features.

Similar considerations can be made for the golden retriever picture chosen. High level representations allow to focus on the relevant part of the image, which lead to correct classification. As we use shallower representations, the focus is lost and, in this case, for the two last models, the network isn't even able to give the correct answer.

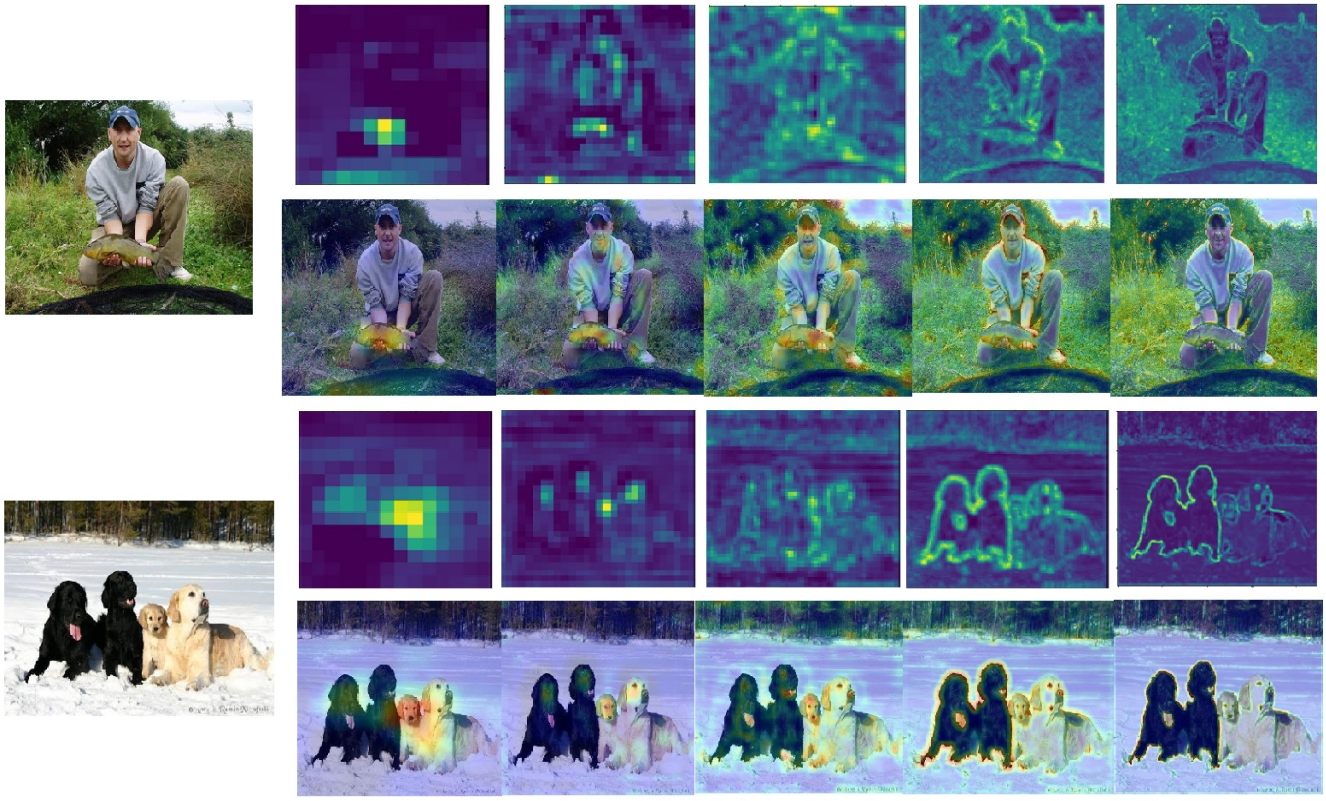

Figure 9. Grad-CAM procedure applied for an image of Imagenette (top) and of Imagewoof (bottom), for different levels of depth.

4. Conclusions

When we study the inner mechanisms of deep convolutional neural networks, we can apply many different procedures to gain insights.

Linear read-outs and confusion matrices can help to understand quantitatively the performance we can expect and the importance of the hierarchical structure of these networks, giving intuition about the separation pipeline the images go through to get classified with the appropriate label.

Visualization techniques, on the other side, give a qualitative sense of what the networks are learning and what they respond to at different levels. This was discussed by looking at the activations of a puppy image going through the network and implementing gradient ascent methods, such as the DeepDream algorithm.

Finally, visualization methods as Grad-CAM are also useful to provide explainability to the whole system: importance scores plotted as heatmaps over the images highlight which pixels are most responsible for the decisions provided by the network. This last part is especially important when debugging a complex system and to scrutinize if its usage is appropriate for certain delicate tasks.

5. References

[1] K. Simonyan, A. Zisserman, "Very Deep convolutional networks for large-scale image recognition", 2015

[2] Deng J. et Al., "Imagenet: A large-scale hierarchical image database", 2009, IEEE conference on computer vision and pattern recognition, pp. 248-255

[3] K. He et Al., "Deep residual learning for image recognition", 2016, Conference on Computer Vision and Pattern Recognition, pp. 770-778

[4] B. Zhou et Al., "Places: A 10 Milion Image Database for Scene Recognition", 2017, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.40-6, pp. 1452-1464

[5] Howard et al., https://github.com/fastai/imagenette

[6] A. Mordvintsev ,"Inceptionism: Going Deeper into Neural Networks", 2015, https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

[7] A. Mordvintsev et Al. "Deep Dream: A Code Example for Visualizing Neural Networks", Google Research Blog, 2015, https://ai.googleblog.com/2015/07/deepdream-code-example-for-visualizing.html

[8] Ramprasaath R. Selvaraju et al., "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization", 2017, https://arxiv.org/abs/1610.02391