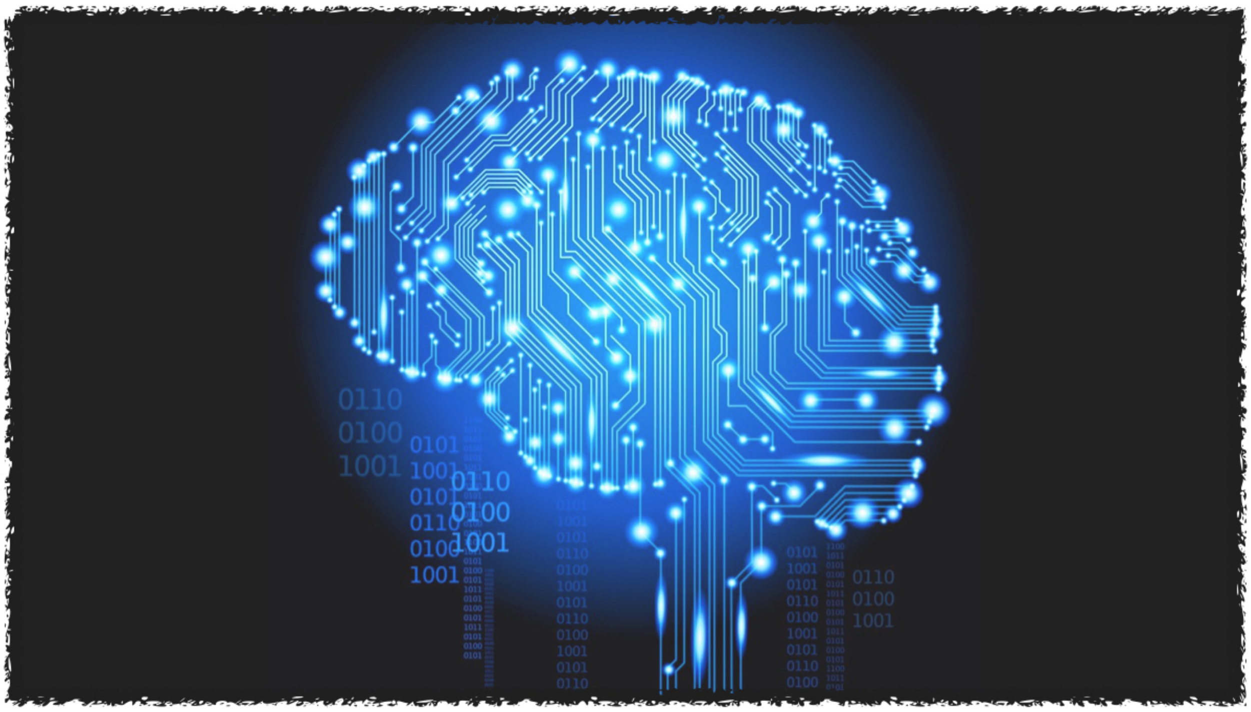
CHALMERS
UNIVERSITY OF TECHNOLOGY



Probabilistic Population Coding in Convolutional Neural Networks

Testing Efficient Coding hypothesis and noise effects in Artificial Neural Networks performing visual tasks

MSc thesis in Complex Adaptive Systems

LAURA MASARACCHIA

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2017

Master's thesis 2017:TIFX05

Probabilistic Population Coding in Convolutional Neural Networks

Testing Efficient Coding hypothesis and noise effects in Artificial Neural Networks performing visual tasks

LAURA MASARACCHIA

UNDER THE SUPERVISION OF
Prof. Dr. Matthias Bethge,
Werner Reichardt Centre for Integrative Neuroscience (CIN),
University of Tübingen,
Tübingen, Germany

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2017

Probabilistic Population Coding in Convolutional Neural Networks
Testing Efficient Coding hypothesis and noise effects in Artificial Neural Networks performing visual tasks

LAURA MASARACCHIA
laura.masaracchia@gmail.com

Supervisors:
Dr. Wieland Brendel, Werner Reichardt Centre for Integrative Neuroscience
Prof. Dr. Matthias Bethge, Werner Reichardt Centre for Integrative Neuroscience, University of Tuebingen.
Examiner:
Prof. Dr. Måns Henningson, Department of Physics, Department of Biological Physics, Chalmers University of Technology

Master's Thesis 2017:TIFX05
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

BethgeLab
Werner Reichardt Centre for Integrative Neuroscience
DE-720 76 Tuebingen
Telephone +49 7071 29 89018

Cover source: extremetech.com

Typeset in LaTeX[1]
Printed by [Name of printing company]
Gothenburg, Sweden 2017

Probabilistic Population Coding in Convolutional Neural Networks
Testing Efficient Coding hypothesis and noise effects in Artificial Neural Networks performing visual tasks
LAURA MASARACCHIA
Department of Physics
Chalmers University of Technology

Abstract

Computers are very practical in our everyday life and sometimes also fundamental for our work: they can store all our holidays pictures, solve quickly complex systems of equations and simulate the outcomes of experiments that cannot be executed in the real world. Computers, however, are not quite as good as *we* are when it comes to tell a joke, recognize a person or playing computer games. In fact, the ability to perform cognitive tasks and to process complex sensory stimuli is still a human brain exclusive.
The surprising skills that we owe to our brain have inspired fields like Computational Neuroscience, Machine Learning and AI. Computational neuroscientists are trying to gain a better understanding of the brain via mathematical modelling of its functions. Machine Learning and AI specialists aim to create machines able to perform accurately on those tasks where only humans excel.
A common denominator in these fields are Artificial Neural Networks (ANNs), computational models (loosely) inspired by the biology of the brain. Especially in visual tasks, like object classification, ANNs became very useful and popular: on one hand because they are able to (broadly) predict and match neuronal patterns in the visual cortex [4], [5], [13], on the other because they reach human accuracy in performance [25]. Unfortunately, each of these two properties does not imply the other. In fact, there can be ANNs resembling our brain firing patterns without solving the task and, conversely, neural networks exceeding human performance but not very informative about the brain.
In this work we want to examine the population code of a state of the art ANN for computer vision. At first we test the robustness of the network against Poisson noise. Successively, we test the Efficient Coding hypothesis on the inner activations of the network by finding sparse representations and analyzing their characteristics.

Keywords: Computational Neuroscience, Machine Learning, Image Classification, Convolutional Neural Networks, Poisson Noise, Sparse Coding.

Acknowledgements

The first big thanks goes to the BethgeLab, where all of this could happen. Witnessing the birth of interesting and creative project ideas from genial people, being surrounded by worldwide top quality scientists, having the privilege to be involved in hard core machine learning lunch conversations -worthy of the nerdiest imaginable environment-: being part of the *Matthias' Gang*, in all its features, was one of the most exciting, motivating and formative experiences I have ever done.
A special thanks to Matthias for being The great leader behind his amazing group. Never enough thanks to Wieland, who has been the clever eye from above, a very patient guide and a wise consultant.

The second big thanks is for Chalmers and Gothenburg: thanks for having been the Home I needed, shaping me as the person I am today. Thanks for teaching me that a better world can exist and it is our responsibility to create it.
Thanks to my best friends, that, even if far away, are making efforts for being part of my life.

Per concludere, milioni di grazie alla mia famiglia. Grazie alla saggezza silenziosa di Federica, all'inestinguibile voglia di conoscere (e di litigare) di Rita, alle sorprendenti doti del povero Giuseppe.
Soprattutto: grazie a mamma e papà. Grazie per tutti e 26 gli anni che mi sopportate. Grazie per sostenermi nelle mie decisioni, per tutti gli sforzi fatti in modo che io potessi seguire i miei sogni, per le chiamate Skype degli ultimi anni, per essere lì anche dopo i miei stupidi scatti nervosi.
Grazie per aver ascoltato tutte le interminabili descrizioni dei miei studi, nonostante "non ci capiate nulla". Questa tesi è per voi. Spero di riuscire a spiegare un pochino meglio "quello che faccio".

Laura Masaracchia, Tuebingen, August 2017

Contents

List of Figures

List of Tables

1
Introduction

1.1 The brain as a complex system

In this work we focus our attention on the study of a particular complex system: the brain. In this simplified approach to the study of the brain, we can identify some fundamental units: the neurons. Neurons respond to their input stimuli by generating characteristic electrical pulses called action potentials: voltage spikes that can travel down nerve fibers. Changing the frequency of their spike emissions neurons encode and propagate information about the input stimulus. Neurons interact with each other -through synapses-, giving rise to emerging properties of the whole system, our brain functions.
Learning, remembering, taking decisions, are only some of the functions that scientists have analyzed and emulated. Many other skills, like language understanding, creative thinking and complex multi-sensory tasks, are still not reliably simulated. In fact, the human brain is capable of solving very complicated tasks and in a such efficient way that no other machine is even comparable with. The tempting and exciting perspective of translating and exploiting the brain's "superpowers" in our computers is triggering many scientists' ambitions all around the world. For a review on the topic see [6].

1.2 The computational approach

The idea of creating a computational model for the brain is more than 70 years old: we could trace its origin to Alan Turing's work and to Warren McCulloch and Walter Pitts, who first created a neural network based on mathematics and algorithms called *threshold logic*, in 1943 [2].
Because of the very complicated brain structure, the enormous amount of connections in it and the fact that getting information on the inner part of the brain is ethically and practically very complicated, scientists started modeling those areas and functions that involve more superficial parts of the brain and other organs. Hence, computational models are mostly devoted to describe tasks that involve our senses: motor, vision and sound processing and perception (for a review on ANN in computational neuroscience see [3]).
Even if computational models were born long time ago, however, only in the last few years we witnessed giant steps forward to the goal of building a man-like machine, with the so called AI revolution. This could happen thanks to technological advances, to huge datasets available and especially to shared knowledge and collab-

orations among different scientific areas. Nevertheless, the understanding of natural intelligence (and what entails it) is still far from perfect.

1.3 Matching the Visual Cortex

In this thesis we are going to explore the properties of Artificial Neural Networks (ANNs) created to perform a visual task.

1.3.1 From the retina to the Inferior Temporal Cortex

When watching an image, a visual stimulus is transmitted from the retina to different brain areas and it reaches the Inferior Temporal Cortex (IT) before we are able to distinguish the object we see.
Each neuron in the retina processes a small portion of the whole visual field: adjacent neurons receive as input slightly different but overlapping subsets of the visual stimulus, so each cell contributes to form a "map" of the visual field, also called *retinotopic map*. In most parts of the Visual Cortex, cells are organized into retinotopic maps. This means that all the different areas of the Visual Cortex furnish a complete map of the visual field, representing the visual stimulus in terms of different sets of characteristics. The visual information is transmitted from the ganglion cells of the retina to the Primary Visual Cortex (V1), where, for example, neurons detect edges [7]. After V1, the stimulus is transmitted to the Secondary Visual Cortex (V2), where neurons encode angles [9]. Going deeper into the brain areas, neurons become less sensitive to the spatial characteristics (like orientation and scale) and more specific in expressing *concepts*. In IT, for instance, neurons may fire only in response to a certain objects in their receptive field [10].
So, within their receptive field, neurons in different areas of the Visual Cortex fire in response to different subsets of stimuli. This property is called neuronal tuning. Tuning is simpler in the earlier areas and becomes more complex in the higher areas of the Visual Cortex. This means that the deeper the brain area we are looking at, the more abstract concepts it extracts from a visual stimulus.

1.3.2 Convolutional Neural Networks

The implementation of ANNs for computer vision tasks is inspired by the retinotopic mapping to the Visual Cortex [5]. The way of representing the visual field in terms of a set of characteristics can be mathematically represented by a convolution operation. The input is convolved with a specific filter that encodes the features to extract. For this reason neural networks that simulate the processing of a visual input are called Convolutional Neural Networks (CNNs). Each adjacent neuron is represented by a computational unit, and its receptive field is defined by the amount of information (pixels of an image) that it processes. The synaptic connections in the brain are represented by *weights* that connect each computational units to another. The spiking of the units in response to the stimuli are mathematically represented with a Rectified Linear Unit function ($ReLU$).

All the neurons representing a certain area of the Visual Cortex are packed in a *computational layer* (or simply called *layer*). Computational layers are structured in sequence so to represent the sequential path of the visual stimulus in the brain. Modern high-performance neural networks sequentially stack many convolutional layers -between 20 and 1000- and are therefore called Deep Neural Networks (DNNs). Each convolutional layer is an homogeneous feature extractor: it slides the filter along the input so to form a map of the visual representation in terms of a specific feature. The different representations that a layer produces of the input are hence called *feature maps* (or *channels*). Going from an early layer toward a deeper layer, the input representation is modeled inducing a spatial reduction in favor of a larger amount of features.
Figure 1.1 (adapted from source: [4]), shows the intended correspondence between CNNs and the visual cortex:

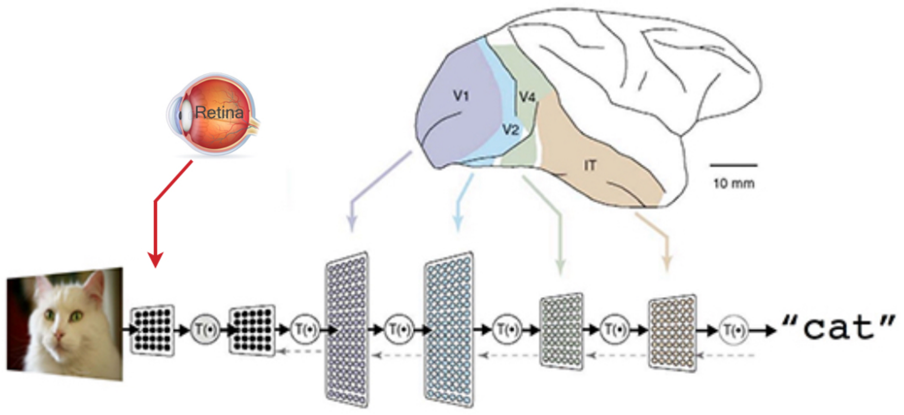


Figure 1.1: Neural Networks matching the Visual Cortex

Once the network is created, it has to learn the values of the weights that lead to successful performance on the task, in a process called training. Depending on the task the network has to perform, the training strategy can be very different. For this thesis, the task under consideration is object classification - recognizing the content of images. The network learns through a supervised process: during training the images come with a label indicating their content (the right *class* they belong to). The response of the network is compared with the labels and its weights adjusted such to lead to a better performance.

1.4 Properties of ANNs

In this section we want to summarize some important properties of Artificial Neural Networks.
On some sensory decision tasks, such as object classification, CNNs reach or even surpass human performance [25]. Moreover, a breakthrough in machine learning, in 2014, was the discovery that, within the same area of interest, a trained network is

able to generalize and perform a new task [15], [14]. For example, Long, Shelhamer and Darrel (in [14]) used some networks trained for object classifications and transferred their learned representations by fine-tuning the networks to the segmentation task (defining different components of a picture). This gave the impression that, to some extent, ANNs are able to *generalize*, just like our brain does, using the same knowledge for different tasks.
Neural Networks furnish very simplistic pictures of the vast panorama the brain actually comprehends. Nevertheless, CNNs are able to resemble brain firing patterns and to match the representations of different areas in the Visual Cortex with their inner layers activations [13], [12].
However, high performance and biological plausibility are often uncorrelated.
Especially the most successful networks, that became state of the art in machine learning, are set up by concatenating many layers of complex and highly nonlinear functions, and their inner outputs are hard to decipher in terms of meaning of the processed input. Understanding the inner representations of ANNs is still a very hard and unsolved problem [16], [17].
Ideally, coupling high performance and biologically inspired processing would lead to extra insight on how our brain works and move a step forward to the realization of a human-like machine.
Our aim here is test the effects of biologically inspired features applied to a state of the art CNNs in object classification.

1.4.1 Life is *noisy*

Even though noise techniques are much used during training for a more robust learning, during the test phase - when performance is checked - the most common CNNs remain still generally deterministic.
This means that, given a fixed input, a trained network will always produce the same activations. On the contrary, the representations in the brain are stochastic [18], [19], i.e. even when watching many and many times the same picture, the neurons in our Visual Cortex will fire with different rates every time.
To describe this form of variability, the theory of *Probabilistic Population Coding* has been developed [20], [21]. It claims that for coding a specific input, our nervous system uses large populations of imprecise neurons, rather than one or a few precise ones. Furthermore, activation patterns are generally over-complete (there is not only one way to encode the same stimulus). Broadly tuned neurons can code for sensory, motor, or perceptual events. Population coding produces precise outputs and is less vulnerable to injury than single-neuron coding.
To express it in the computational environment we need to add certain noise to the deterministic functions of the computational units.
The spiking of a neuron embodies the mathematical concept of a renewal process: a stochastic process describing the occurrence of an event in a certain interval of time. The firing rates can hence be modelled by a Poisson distribution [18], [19].
Another form of variability that is not generally taken into account is neurons' rest activity: a computational unit with zero input will have zero output, but our neurons have a background firing rate that is variable. In a mathematical environment this

noise can be simulated as a constant additive component to the activations.

1.4.2 The importance of being efficient

The Artificial Neural Networks used for machine learning competitions are relying on the power of super computers, analyzing all the data they can, and are not designed explicitly for optimal coding. The efficient coding hypothesis in the brain was proposed by Horace Barlow in 1961 [22] as a theoretical model of sensory coding. Barlow hypothesized that the visual processing system must have an efficient strategy for transmitting as much information as possible, due to constraints on the visual system such as the number of neurons and the metabolic energy required for the neural activity. In this view, information must be compressed as it travels from the retina back to the visual cortex, in order to maximize neural resources. Efficient coding occurs by minimizing the number of spikes needed to transmit a given signal (*Sparse Coding*). Sparse coding networks encode each stimulus through the strong activation of a relatively small set of units. Given a potentially large set of different input images, they attempt to automatically find a small number of representative patterns which, when combined in the right proportions, reproduce the original input [23]. Another sign for efficient coding is having neurons encoding *meaningful information* about the data. For example, DiCarlo and Cox in 2007 [4] showed that encoding semantic concepts corresponds to disentangled representations of the data, which are easier to discern (compared to the representations in terms of edges and colors) and crucial for recognition in our brain.

1.5 This work

The aim of this thesis is to introduce two biological features in the current state of the art CNNs and analyze their effects. A pre-trained CNN for object classification is chosen and used as a reference for all the analyses performed. Poisson noise and a certain fraction of bias (offset voltage) are introduced in the activations of the inner layers of the network, and their effect on the overall final performance tested.
An algorithm for optimal coding is used to find sparse representations of the layers activations. Sparse representations should help in understanding the fundamental components of layers activations and hence also the *concepts* that different layers encode. First, basic theory on CNNs, Poisson Noise and Sparse Coding is provided. The results are then presented and in the Methods chapter a detailed description of the work done will follow. Extra analyses, proves and evidences are reported in Appendix I.

2

Theory

2.1 Object Classification in CNNs

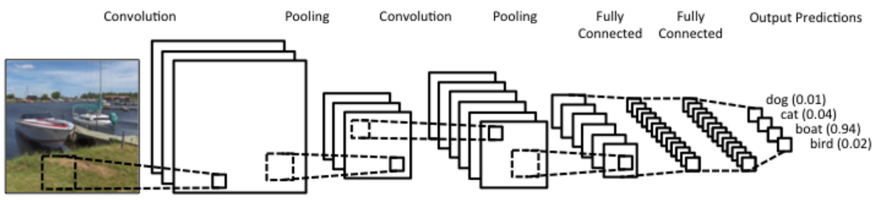


Figure 2.1: Visual example of data processing in a CNN

Figure 2.1 (source: [24]) represents the input processing in a CNN performing an object classification task.
The input is a 2D RGB image: an array of dimensions [X,Y,3], where X and Y are the width and height pixels, and one channel is used for each of the three colors. In a typical CNN the convolutional layers are used to give representations of the input in terms of different sets of characteristics (the feature maps). The spatial dimensions of these representations are reduced with some down sampling (*Pooling*) technique. After a series of Convolutions and Poolings, the input is passed to Fully Connected Layers, which collect the information and associate scores of confidence to the classes the image can belong to.

2.1.1 Convolutional layers

Mathematically, a convolution between two discrete functions $f(x_1, y_1)$ and $g(x_2, y_2)$ with a finite support ($x_1 \in [-X_1, X_1]$, $y_1 \in [-Y_1, Y_1]$, $x_2 \in [-X_2, X_2]$, $y_2 \in [-Y_2, Y_2]$) is given by:

$$(f * g)(i,j) = \sum_{k=-X_1}^{X_1} \sum_{l=-Y_1}^{Y_1} f(k,l)\; g(i-k, j-l) \tag{2.1}$$

A convolutional layer takes an input and convolves it with fixed *kernels* (the *filters*). The input has dimension [b, X, Y, c_i], where b = batch size, i.e. number of images analyzed per time, X = spatial width, Y = spatial height, c_i = input channels (or input feature maps). The kernels have dimension [3, 3, c_i, c_o]. The convolution

consists of sliding the 3x3 kernels along the spatial dimensions of the each image, transforming the c_i values of every input spatial point into c_o other values to form the output, giving an output of size [b, X, Y, c_o]. The kernels are the key part of the whole network. They are fixed for each layer, and their values are the parameters learned during training.
A visual representation of the operation done by a convolutional layer can be seen in figure 2.2 (source: [26]):

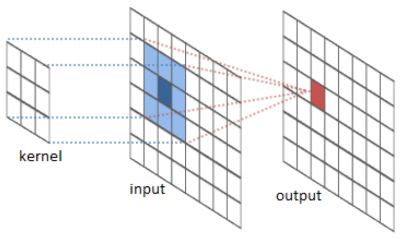


Figure 2.2: Visual example of convolution with kernel

After every convolution, a ReLU function is applied to ensure the output is non negative:

$$f(x) = max(0, x) \tag{2.2}$$

2.1.2 Max Pooling layers

The term *pooling* refers to a non-linear down sampling. A common form of pooling is the so called "max pooling": it consists of building an output extracting the maximum value in 2x2 patches of the input.
Example of a max pool operation is shown in figure 2.3 (source: [27]):

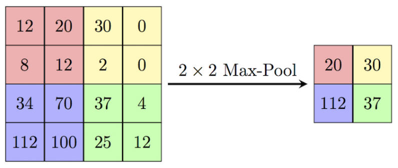


Figure 2.3: Visual example of max pooling

2.1.3 Fully Connected layers and Softmax

Given an N-dimensional input (column) vector $\bar{X} = [X_1, ...X_N]$, a C-dimensional biases (column) vector $\bar{b} = [b_1, ..., b_C]$ and a (CxN) weight matrix $\mathbf{W} = [W_{1,1}, ..., W_{1,N}, ..., W_{C,N}]$, the fully connected layer produces a C-dimensional output that is a linear combination of input, weights and biases:

$$\bar{y} = \mathbf{W}\ \bar{X} + \bar{b} \tag{2.3}$$

The output of the fully connected layer is a vector of the dimension of the number of classes C. The value of each entry of this vector is a score that tells how much the network is confident about the input image belonging to that corresponding class. Example of a fully connected layer in figure 2.4 (source: [28]):

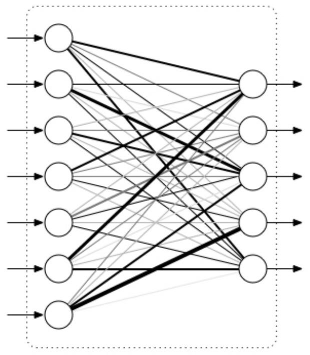


Figure 2.4: Fully Connected layer

These scores are then transformed into a probability distribution via a softmax function: for the vector of scores $\bar{y} = [y_1, ..., y_C]$ for each sample, the new vector containing the probability distribution of a sample belonging a specific class will be: $\bar{S}(y) = [S(y_1), ..., S(y_C)]$ where:

$$S(y_c) = \frac{e^{y_c}}{\sum_{c=1}^{C} e^{y_c}} \tag{2.4}$$

2.1.4 Loss Function and Weights Update

As already mentioned in the Introduction (section 1.4), the inputs come with labels associated to them, reporting the correct class, in the so called one-hot encoding: for example label vector $\bar{L_i} = [0, ..., 0, 1, 0, ..., 0]$ encodes the label of image i with 1 in the position of the class it belongs to and 0 otherwise.
Indicating with $\bar{S_i}$ the vector after softmax encoding the probability distribution of image i to belong to a specific class, the output of the network and the label have to be compared, and the performance evaluated. A way to express the accuracy of the network is comparing its predictions and the right class of the input via Cross Entropy:

$$D(\mathbf{S}, \mathbf{L}) = -\sum_i \bar{L_i} \; log(\bar{S_i}) \tag{2.5}$$

A Loss Function F is defined such that a lower loss corresponds to a better performance of the network. In object classification, the loss function generally includes the average cross entropy across samples. In a set of B samples, and C available classes it would be:

$$F = -\frac{1}{B} \sum_{i=1}^{B} \sum_{c=1}^{C} L_i(c) \; log(S_i(y_c)) \tag{2.6}$$

The weights of the network are iteratively optimized during training for a better performance. The most common algorithm used for it is Gradient Descent (with all its variants): it consists of finding the minimum of the loss function computing its derivative with respect to the weights. Then, at every iteration t the weights are updated according to:

$$w_{ij}^{(t+1)} \leftarrow w_{ij}^{(t)} - \eta \, \frac{\partial F^{(t)}}{\partial w_{ij}^{(t)}} \tag{2.7}$$

with η the *learning rate*.

2.1.5 VGG-19

The reference net we used for our experiments is the state of the art CNN in Supervised Object Classification: VGG-19. VGG-19 was implemented by the Visual Geometry Group at the University of Oxford for a study on performance of CNNs as a function of the number of layers [25]. The 19 refers to the number of layers with weights to be trained. Table 2.1 reports the structure of the whole network, including trainable and not trainable layers. In the table the name "convx-y" refers to convolutional layer, cycle x, subcycle y, "max-pool-x" to max pooling Layer x, "FC-x" indicates a fully connected layer with the number of units that constitutes it. It follows a description of each type of computational layer.

The specifications of the convolutional layers present in VGG-19 are shown in table 2.2.

Input: 224x224 RGB image	
1	conv1-1
2	conv1-2
max-pool-1	
3	conv2-1
4	conv2-2
max-pool-2	
5	conv3-1
6	conv3-2
7	conv3-3
8	conv3-4
max-pool-3	
9	conv4-1
10	conv4-2
11	conv4-3
12	conv4-4
max-pool-4	
13	conv5-1
14	conv5-2
15	conv5-3
16	conv5-4
max-pool-5	
17	FC-4096
18	FC-4096
19	FC-1000
softmax	
Output: probability dist of input class	

Table 2.1: VGG-19 skeleton

Layer	Input Size	Output Size	Kernel Size
conv1-1	(b, 224, 224, 3)	(b, 224, 224, 64)	(3, 3, 3, 64)
conv1-2	(b, 224, 224, 64)	(b, 224, 224, 64)	(3, 3, 64, 64)
conv2-1	(b, 112, 112, 64)	(b, 112, 112, 128)	(3, 3, 64, 128)
conv2-2	(b, 112, 112, 128)	(b, 112, 112, 128)	(3, 3, 128, 128)
conv3-1	(b, 56, 56, 128)	(b, 56, 56, 256)	(3, 3, 128, 256)
conv3-2	(b, 56, 56, 256)	(b, 56, 56, 256)	(3, 3, 256, 256)
conv3-3	(b, 56, 56, 256)	(b, 56, 56, 256)	(3, 3, 256, 256)
conv3-4	(b, 56, 56, 256)	(b, 56, 56, 256)	(3, 3, 256, 256)
conv4-1	(b, 28, 28, 256)	(b, 28, 28, 512)	(3, 3, 256, 512)
conv4-2	(b, 28, 28, 512)	(b, 28, 28, 512)	(3, 3, 512, 512)
conv4-3	(b, 28, 28, 512)	(b, 28, 28, 512)	(3, 3, 512, 512)
conv4-4	(b, 28, 28, 512)	(b, 28, 28, 512)	(3, 3, 512, 512)
conv5-1	(b, 14, 14, 512)	(b, 14, 14, 512)	(3, 3, 512, 512)
conv5-2	(b, 14, 14, 512)	(b, 14, 14, 512)	(3, 3, 512, 512)
conv5-3	(b, 14, 14, 512)	(b, 14, 14, 512)	(3, 3, 512, 512)
conv5-4	(b, 14, 14, 512)	(b, 14, 14, 512)	(3, 3, 512, 512)

Table 2.2: VGG-19 convolutional layers specifications

2.2 Poisson Distribution

The Probability Mass Function (pmf) of a Poisson Distribution with mean μ (hence also variance= μ) and support k $\in \mathbb{N}$ is:

$$P(X = k, \mu) = e^{-\mu}\ \frac{\mu^k}{k!} \tag{2.8}$$

The scale of the noise is determined by varying the variance of the distribution to sample from (see Methods, section 4.2, for more details).

2.3 Convolutional Sparse Coding

With the term *sparse representation* one indicates the sparse M-dimensional (column) vector $\bar{\Gamma}$ assuming that any N-dimensional signal $\bar{X}$ (column vector) can be described by a multiplication of a NxM matrix **D**, also called *Dictionary*, and $\bar{\Gamma}$. For a given signal $\bar{X}$ and a fixed dictionary **D**, the task of recovering its sparsest representation $\bar{\Gamma}$ is called *sparse coding*, and it attempts to solve the following problem:

$$min_{\bar{\Gamma}}\ ||\bar{\Gamma}||_0\ \ s.t.\ \ \mathbf{D}\ \bar{\Gamma} = \bar{X} \tag{2.9}$$

A convex relaxation of the problem can be formulated:

$$min_{\bar{\Gamma}}\ \frac{1}{2}\ ||\mathbf{D}\bar{\Gamma}\ -\ \bar{X}||_2^2\ +\ \beta\ ||\bar{\Gamma}||_1 \tag{2.10}$$

for some *positive* scalar β, the *sparse regularization term.* And one of the simplest approaches for solving (2.10) now is via the *soft thresholding algorithm*, finding as solution: $S_\beta(\mathbf{D}^T\bar{X})$.

S_β indicates the soft iterative thresholding operator:

$$S_\beta(z) = \begin{cases} z+\beta, & if \quad z < -\beta \\ 0, & if \quad -\beta \leq z \leq \beta \\ z-\beta, & if \quad \beta < z \end{cases} \tag{2.11}$$

Due to computational constraints, training the dictionaries is feasible only when dealing with low dimensional data. For more demanding computations, hence, Convolutional Sparse Coding is generally preferred. It attempts to reproduce the whole N-dimensional signal $\bar{X}$ as a convolution of global *convolutional dictionaries* $\bar{\mathbf{D}}$ (like the kernels of a convolutional layer, see section 2.1.1 for more details) and the sparse vector $\bar{\Gamma}$. Compared to the NxM dimensional dictionaries $\mathbf{D}$, the convolutional dictionaries $\bar{\mathbf{D}}$ are a smaller set of fixed weights sliding across the input to build the output. This implies fewer computations and weights to be learned.
In their paper on *Convolutional Neural Networks Analyzed via Convolutional Sparse Coding* (arXiv, October 2016) [29], Papyan, Romano et al. describe an algorithm of layered iterative sparse coding, which we used as reference algorithm for finding sparse representations of the activations of every VGG-19 layer (see Methods, section 4.3 for more details).

3
Results

For this thesis work, a pre-trained VGG-19 with average top 5 accuracy (i.e. the net is considered to perform correctly if the right class for the proposed image is among the 5 highest probability classes it outputs) of 88% is used as reference net. The experiments are applied only to the convolutional layers. Hence, throughout the Results and Methods sections, the layers under considerations will be only the convolutional ones, unless otherwise specified. The activations of the VGG convolutional layers are normalized, so that they have mean = 1.0 and variance = 1.0.

3.1 Noise experiments

At first, VGG-19 performance is tested for different amount of offset noise injected, to simulate the effect of background firing noise in the neurons. Figure 3.1 shows how the performance is affected as a function of offset only. A significant trend is shown: the accuracy of VGG does not decreases considerably until the 20% of offset is added, but with an amount of noise above this threshold the accuracy faces a drastic drop.

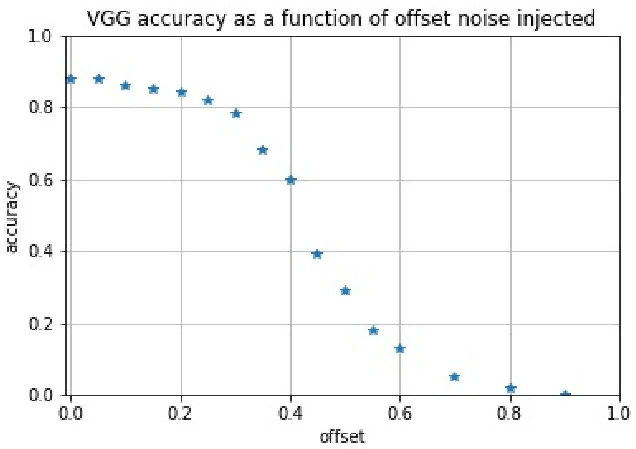


Figure 3.1: Performance of VGG-19 as a function of offset injected

Secondly, the performance of VGG-19 is tested as a function of the amount of Poisson distributed noise, when a certain offset potential is applied to its activations. The amount of Poisson distributed noise is controlled by varying λ, the *signal-to-noise* *ratio*, also the mean of the Poisson distribution from which noise is sampled. Noise scale λ indicates a Poisson distributed noise with actual variance λ^{-1}. See Methods, section 4.2, for more details.

An offset of 0.1 means a shift on the current activation value of 10%. Comparing the 0.1 offset case with the 0.0 offset, the drop in VGG performance (without extra noise) is negligible.

In figures 3.2, 3.3 and 3.4, with respectively 0.0, 0.1 and 0.25 offset is inserted, a "saturation regime" can be seen for the smallest amount of Poisson distributed noise introduced. It can be found a value of λ for which the accuracy of VGG recovers the baseline accuracy (with only offset inserted), that will be referred to as λ *threshold*. When $\lambda > \lambda$ *threshold* the accuracy drops sharply.

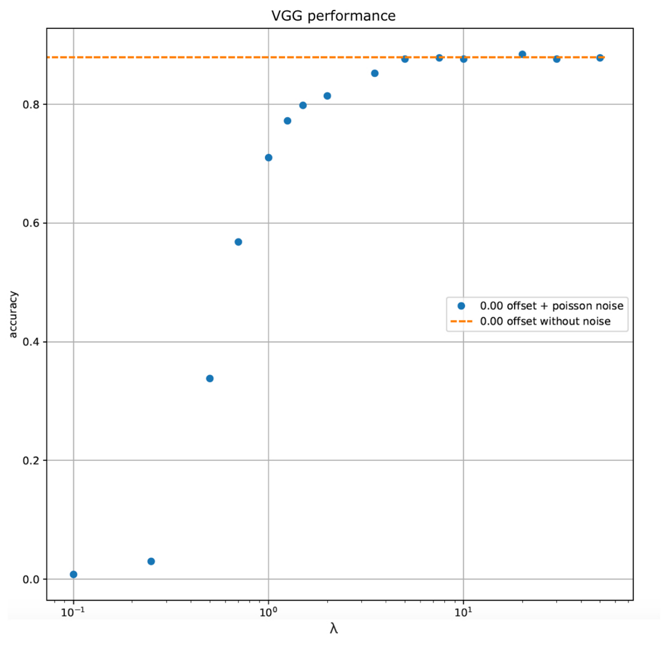


Figure 3.2: Performance of VGG-19 with and without Poisson noise and NO offset

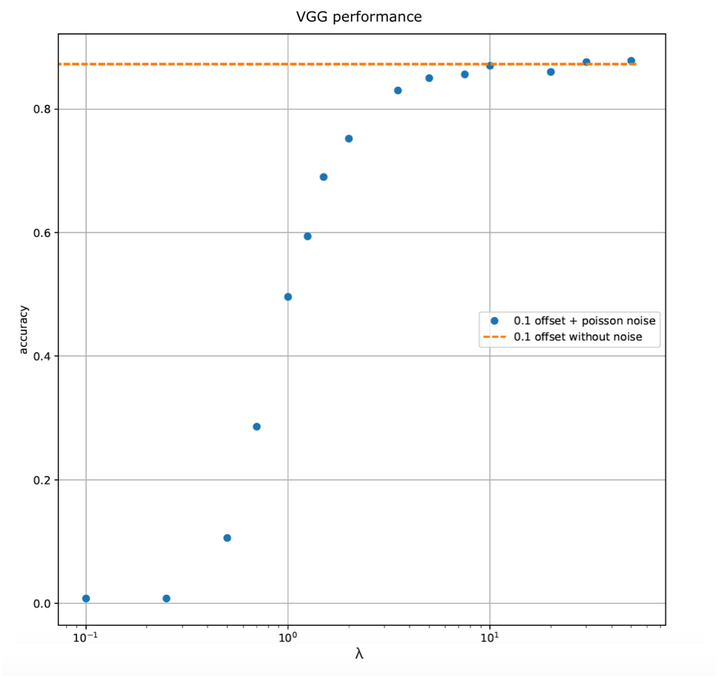


Figure 3.3: Performance of VGG-19 with and without Poisson noise, offset of 0.1

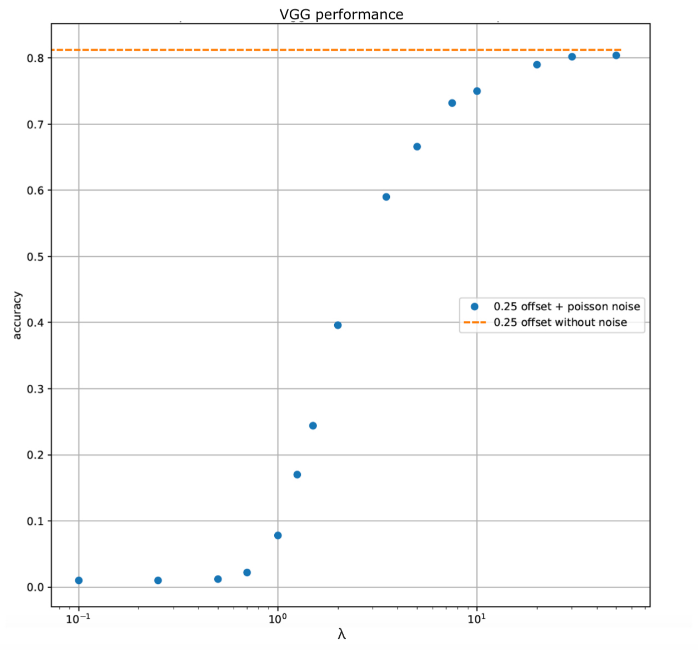


Figure 3.4: Performance of VGG-19 with and without Poisson noise, offset of 0.25

In figures 3.5 and 3.6, reporting accuracy of VGG as a function of Poisson noise when 0.4 and 0.5 offset is injected, the baseline accuracy is never recovered and λ *threshold* cannot be found.

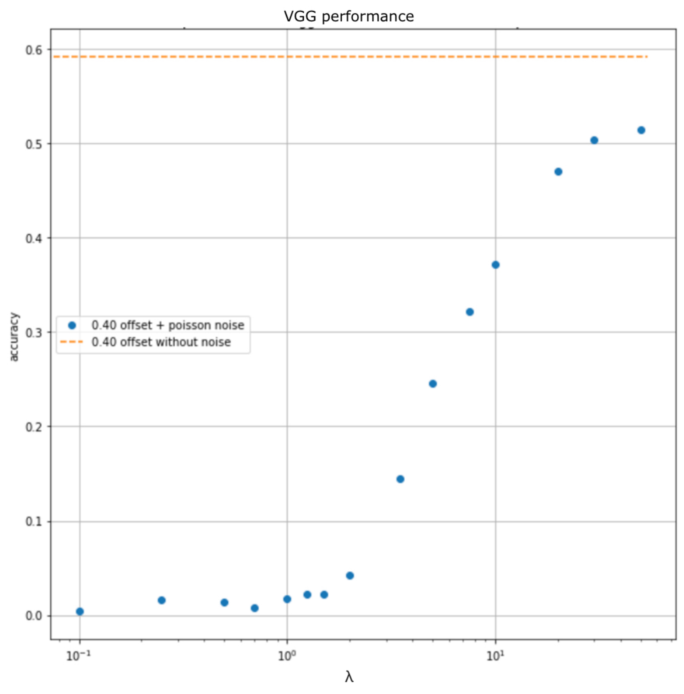


Figure 3.5: Performance of VGG-19 with and without Poisson noise, offset of 0.4

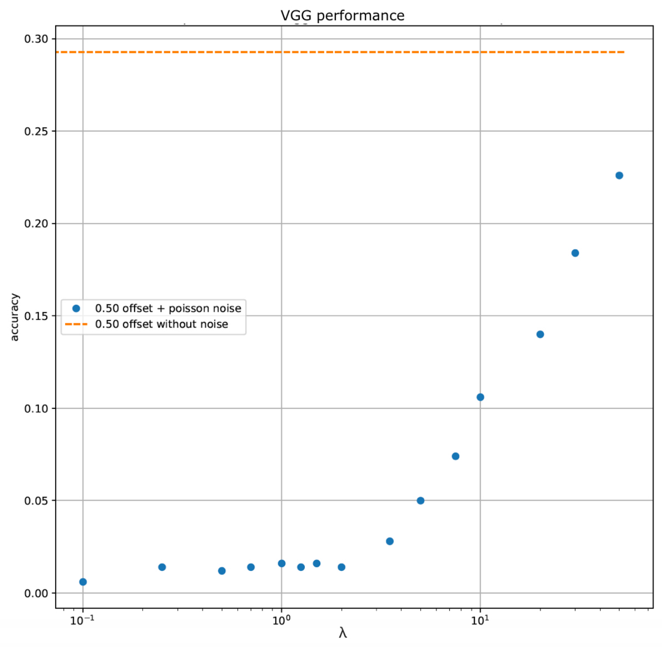


Figure 3.6: Performance of VGG-19 with and without Poisson noise, offset of 0.5

3.2 Sparse representations

In order to find sparse representations of the VGG activations, we trained the convolutional dictionaries to extract sparse features from VGG activations. According to Barlow's Efficient Coding hypothesis we would expect a smaller number of neurons used to encode information as we go deeper into the brain areas, hence smaller number of units active the deeper the layer.
Figure 3.7 shows the sparsity of the optimal Sparse representations found (more details on the training procedure and optimization constraints in section Methods). The sparsity of the original VGG activations is included as a mean of comparison. The sparsity of a set of activations i is defined as:

$$S_i = \frac{nbr\ of\ active\ units}{nbr\ of\ total\ units}$$

Although the sparsity of the original VGG activations changes across layers (very dense conv1-1, instead above 90% sparsity of conv5-4), unexpectedly, the overall sparsity of the final SPARSE representations does not vary much across layers.
It can be said that the optimal coding hypothesis is somehow valid for the standard VGG, but the sparse representations of VGG activations have an average homogeneous sparsity across layers.

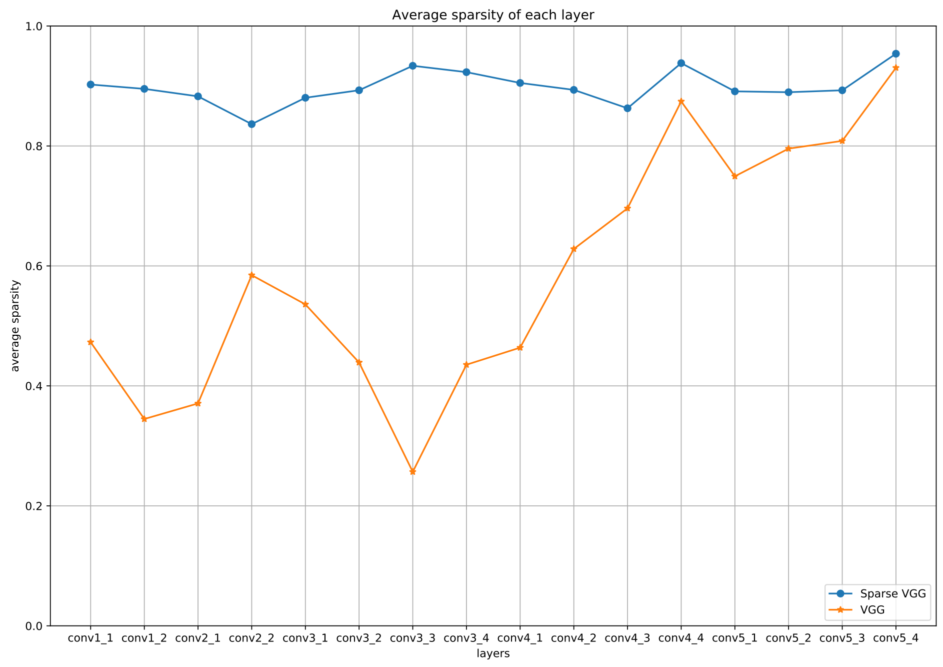


Figure 3.7: Sparsity of SPARSE VGG and original VGG

However, deeper layers contains fewer units, so that the sparsity measure alone does not furnish a complete picture. Figure 3.8 shows the actual number of units active in different layers for 100 samples tested. It can be seen that the total number of active units actually decreases across layers, confirming Barlow's hypothesis.

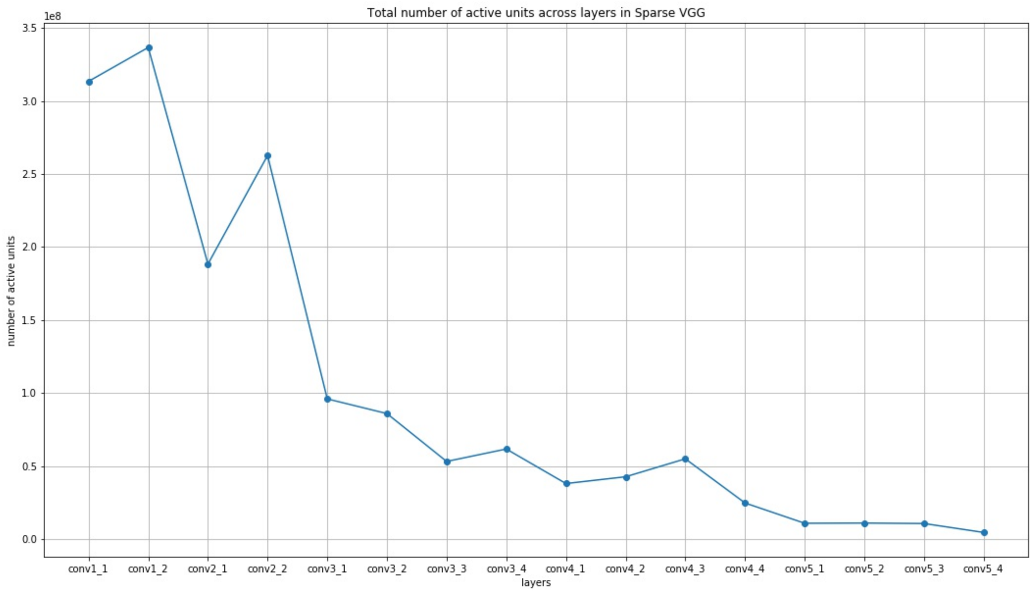


Figure 3.8: Actual number of active units for Sparse VGG

Layer	S_{VGG} (%)	ξ	R_{rel}	S_{Γ^K} (%)
conv1-1	48.0	0.65	0.014	90.2
conv1-2	34.6	0.55	0.014	89.5
conv2-1	36.0	0.4	0.017	88.3
conv2-2	58.5	0.7	0.030	83.6
conv3-1	54.2	0.7	0.040	88.0
conv3-2	43.8	0.7	0.051	89.3
conv3-3	25.7	0.75	0.045	93.4
conv3-4	44.2	0.7	0.037	92.3
conv4-1	46.4	0.8	0.050	90.5
conv4-2	62.0	1.2	0.050	89.3
conv4-3	70.0	1.25	0.057	86.3
conv4-4	87.3	3.4	0.054	93.8
conv5-1	75.0	1.95	0.070	89.1
conv5-2	79.0	1.7	0.041	89.0
conv5-3	80.0	1.9	0.041	89.3
conv5-4	92.0	3.4	0.018	95.4

Table 3.1: Optimal sparse representations characteristics

In table 3.1 are reported the salient information on the optimal solutions found for Sparse VGG, layer by layer.
The specifications shown in the table concern:
S^i_{VGG} (%) = percent of sparsity of the activations of layer i in the pre-trained VGG-19.
ξ_i = sparse regularization term for layer i.
R^i_{rel} = Relative Reconstruction Error of layer i, defined as: $\frac{||X_i - \bar{X}_i||_2^2}{||X_i||_2^2}$
$S_{\Gamma^K_i}$ (%) = percentage of sparsity of the optimal sparse representation of layer i.

The plots in figures 3.7 and 3.8 report average sparsity values across spatial points and channels for the activations of each layer. We now explore each of the sparse layers singularly to get some insight on the patterns of sparsity that may arise.

For each layer, we analyze sparsity, mean and standard deviation across channels, and we give an intuition of how these activation spatially look like. All the layers have been analyzed. For representative purposes, however, only three of them are reported here: the first, the last and one of the middle layers. The three selected layers are conv1-1, conv3-3 and conv5-4. The complete plots for all the layers can be found in Appendix I.

Plots in figures 3.9, 3.10 and 3.11 compare VGG and SPARSE VGG activations on the average mean, standard deviation and sparsity across channels. The channels are sorted by increasing mean. These measures are taken for 1000 samples.

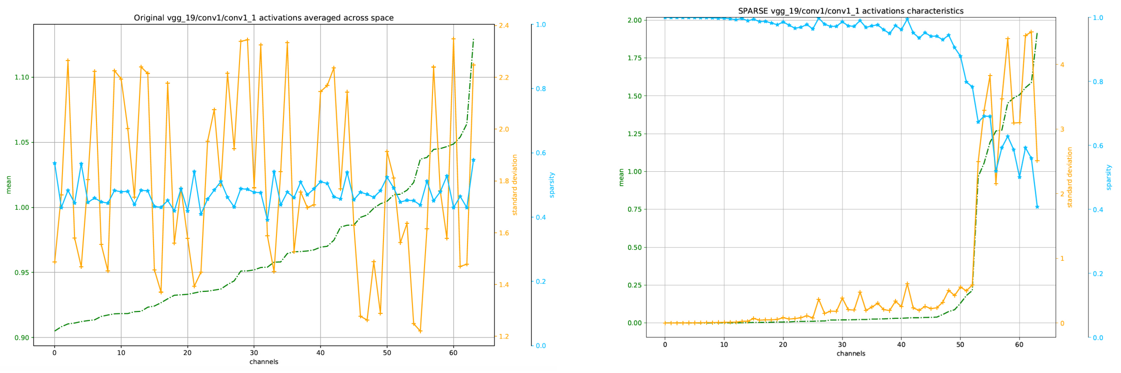


Figure 3.9: Comparison between VGG and SPARSE VGG activations characteristics in conv1-1

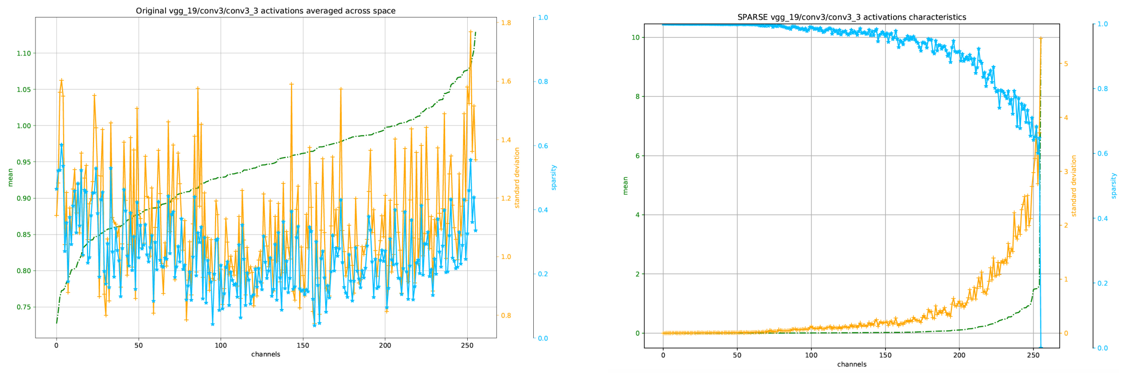


Figure 3.10: Comparison between VGG and SPARSE VGG activations characteristics in conv3-3

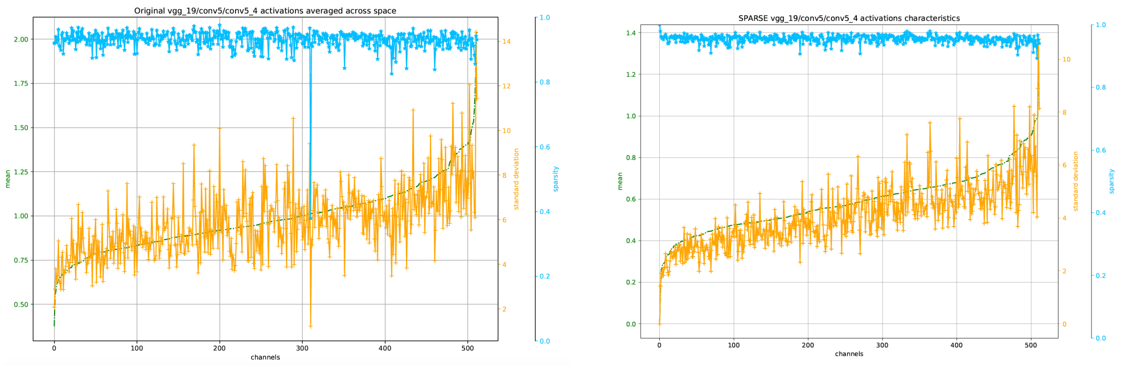


Figure 3.11: Comparison between VGG and SPARSE VGG activations characteristics in conv5-4

In figures 3.13, 3.14 and 3.15, the activations of the selected layers are visually shown for a given picture (3.12). The figures include the activations of the real VGG (X_i), reconstructed VGG ($\bar{X}_i$) and their difference, for a selected feature map, then a statistics of the sparsity across channels in the Sparse representations (Γ_i) and two examples of their activations, including distribution and visualization. The visual representations are chosen so to give an intuition of the main characteristics of both the VGG and the Sparse spatial activations.

Even though the correct visualization and the understanding of the layers activations are the subject of current vast studies themselves, it can be generally said that the first layers are extracting from the image some very basics features, like colors and edges (in particular, the conv1-1 VGG activations light up specific colors of the input). The deeper the layer, the more complex concepts it is supposed to encode. The activations of conv3-3 shown might capture the concepts of facial components (the active parts are those corresponding to eyes, ears and muzzle). The final layers spatial activations are very hard to interpret.

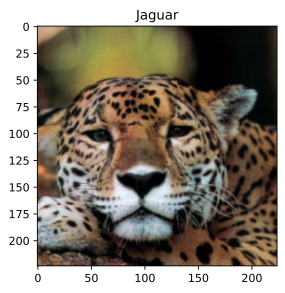


Figure 3.12: Example Image 1, classified as a Jaguar

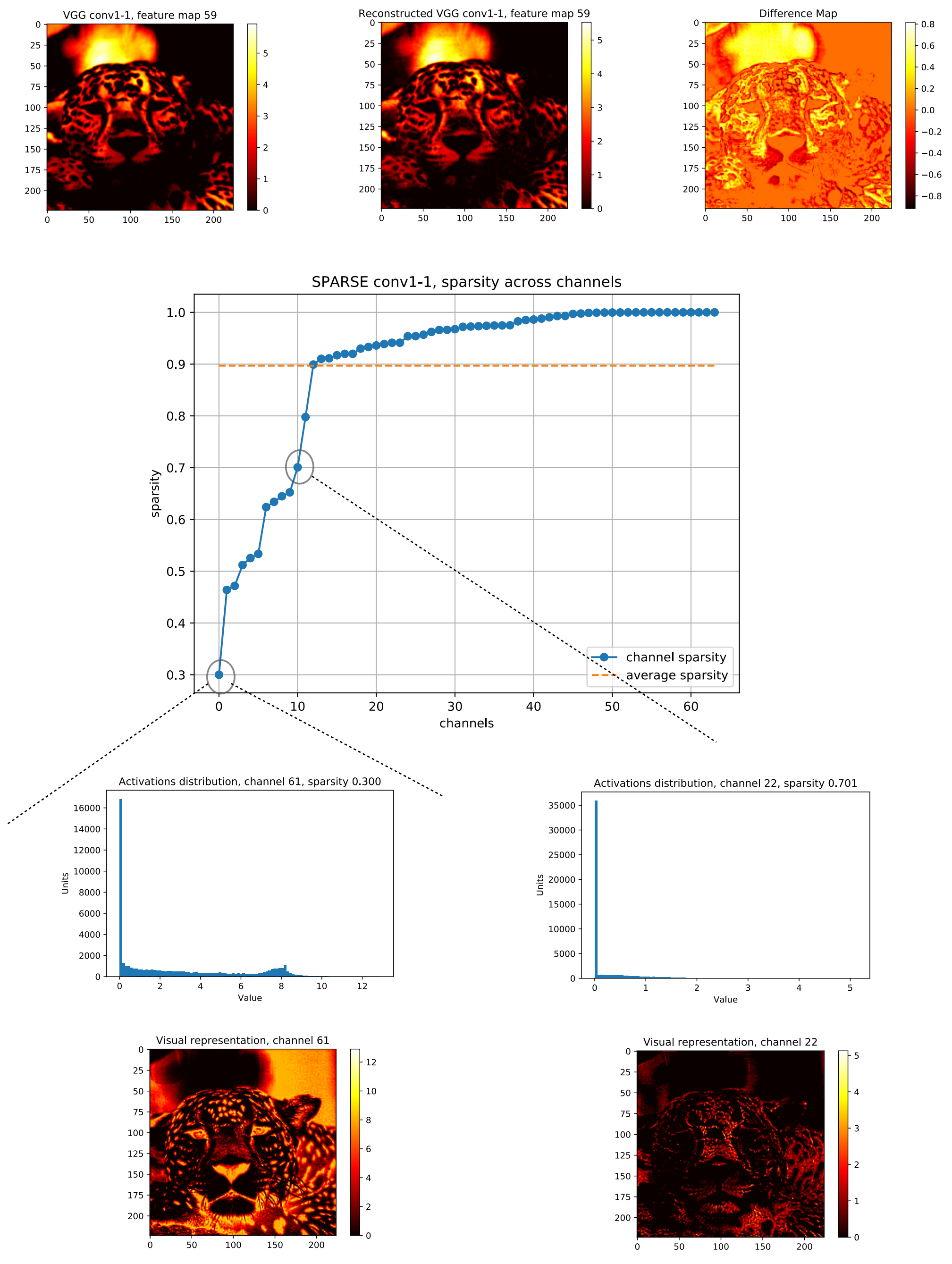


Figure 3.13: Example of activations distribution and visualization for Example Image 1, SPARSE conv1-1

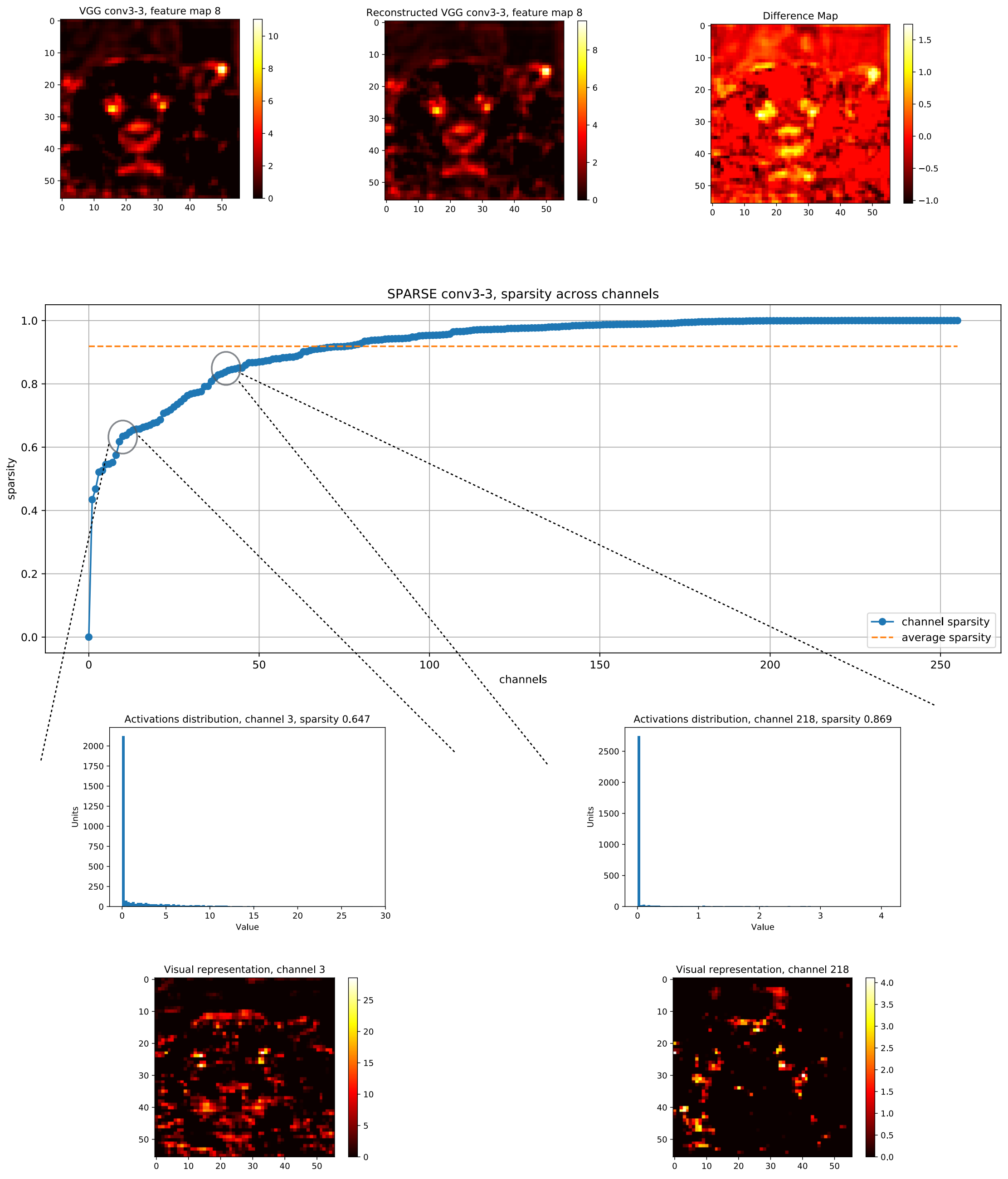


Figure 3.14: Example of activations distribution and visualization for Example Image 1, SPARSE conv3-3

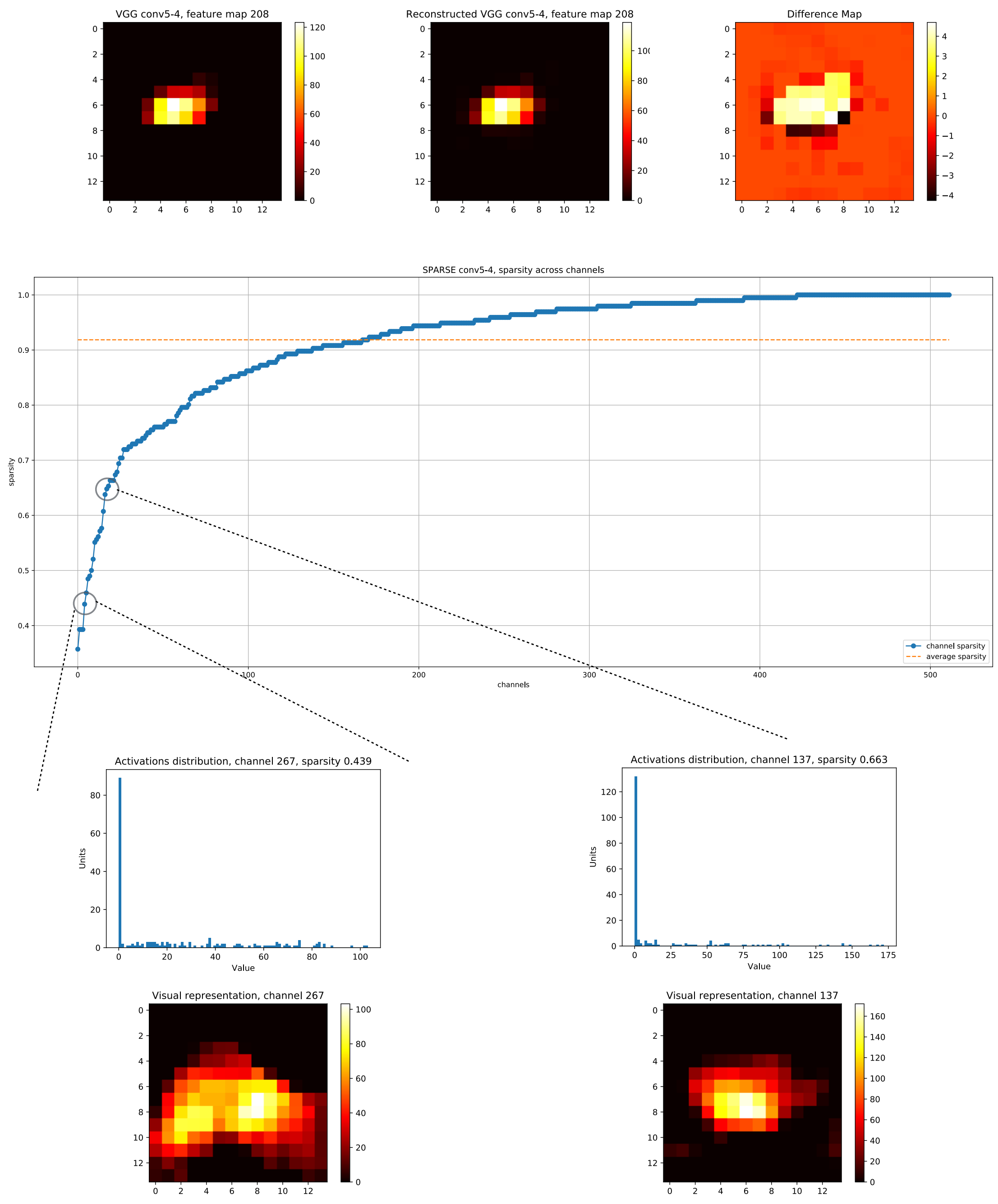


Figure 3.15: Example of activations distribution and visualization for Example Image 1, SPARSE conv5-4

3.2.1 Spatial localization

A general trend to outline is the spatial concentration of active regions. In the last layers the active units are more localized. It can be justified by thinking that, since the last layers are most probably representing some abstract concepts, the activations are focused on spatial areas containing specific characteristic. To exploit this hypothesis an extra analysis is conducted: we apply max pooling to each of the Sparse activations until they reach the same spatial dimensions of the last layer and test then the sparsity on the "pooled" layers. The general idea behind this analysis is that if the active units are spread around, pooling will maintain this pattern, resulting in a less sparse "pooled layer" when compared to the last layer.

Here are reported visual examples of some images for a selected feature map of the "pooled conv1-1" (conv1-1 sparse activations reduced to conv5-4 spatial dimension via max pooling), of the "pooled conv3-3" and of the real conv5-4. As it can be seen, the "pooled layers" don't present the same characteristics of the original sparse representations. The spatial activations in the pooled layers are spread all over the map (this results also in a much lower average sparsity), while the activations of conv5-4 are visibly more localized.

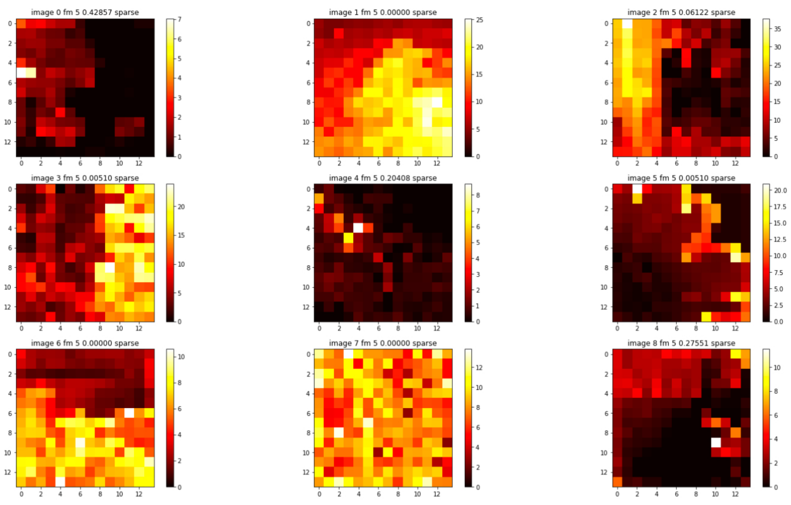


Figure 3.16: Visual examples pooled conv1-1

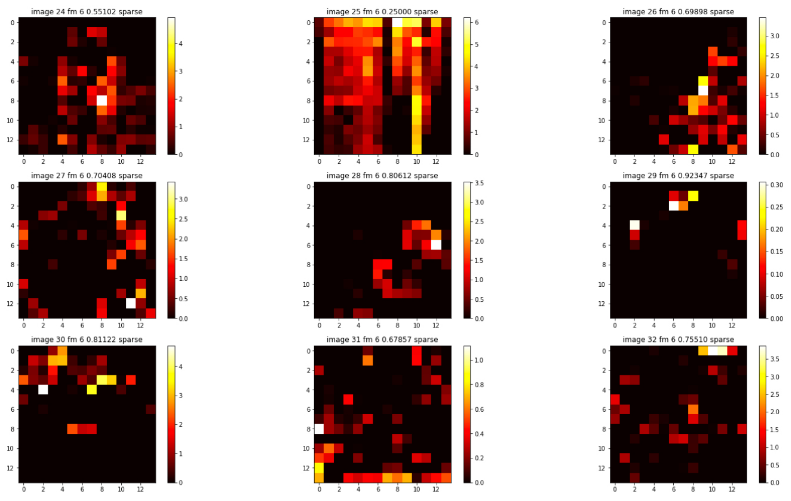


Figure 3.17: Visual examples pooled conv3-3

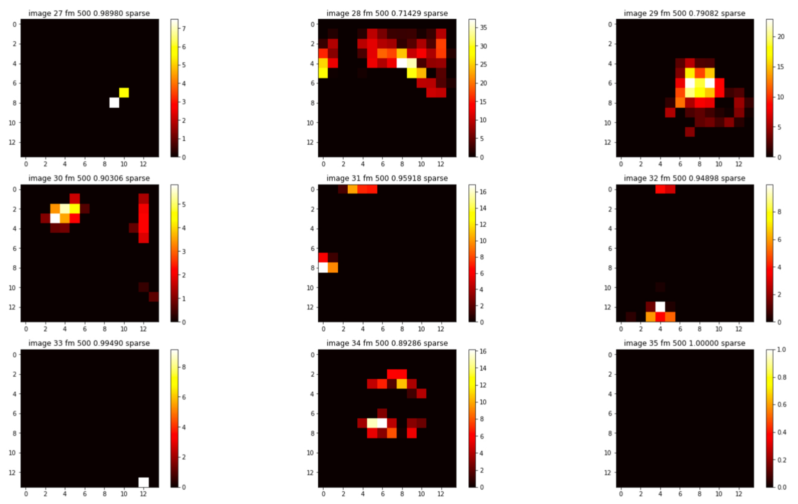


Figure 3.18: Visual examples conv5-4

The average sparsity across channels and space for the pooled sparse activations is computed and plotted against the sparse activations: figure 3.19 shows that, as expected, the pooled layers lower sparsity, increasing for deeper layers. Figure 3.20 reports the variance in sparsity across channels. Higher variance means more diversity in average sparsity across channels, while smaller variance indicates that the channels have homogeneous sparsity.

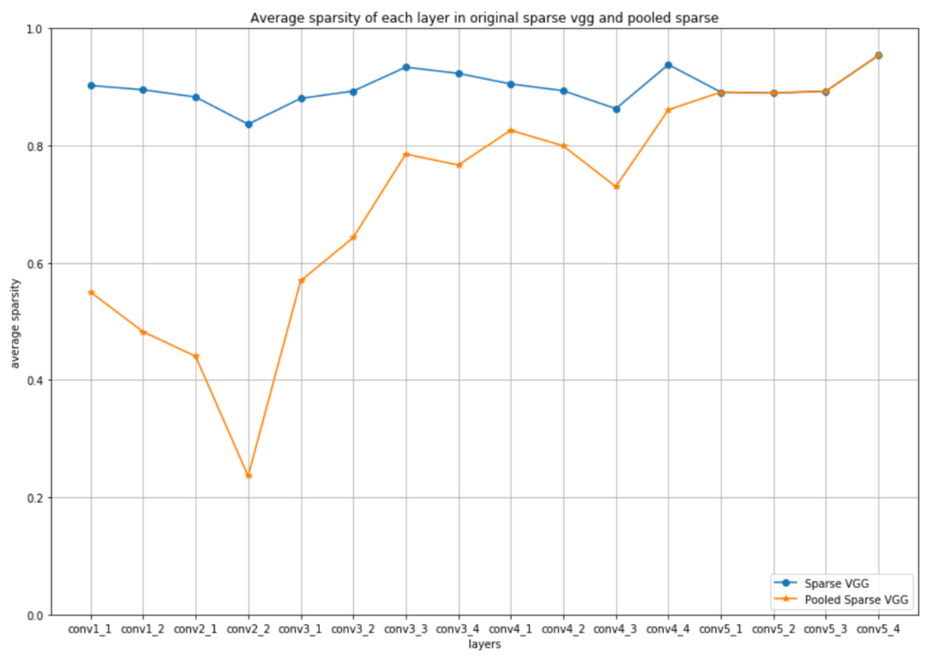


Figure 3.19: Average sparsity of the pooled layers compared to sparsity of original SPARSE representations.

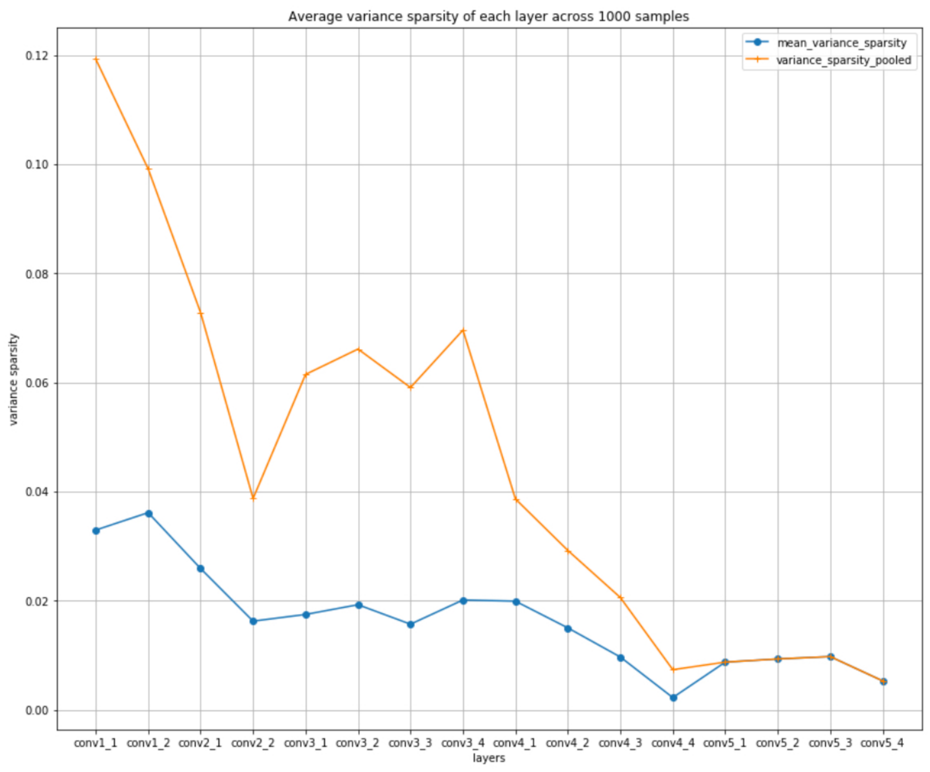


Figure 3.20: Variance of the sparsity of the pooled layers compared to the variance of the sparsity of original SPARSE representations.

3.2.2 Sparse homogeneity

Another trend that pops out is that sparsity is very diverse across channels in the first layers (e.g. some channels activations are zero for all the samples tested, some others have very low sparsity). Rather homogeneous sparse configurations appear instead in the deeper layers, where all the channels are active (i.e. each channel is active at least for one sample tested), with very high sparsity.

For a clearer view of differences (across the selected layers) in structural sparsity, 1000 images are analyzed, computing the percentage of active spatial units per each channel. Figures 3.21, 3.22 and 3.23 show how many samples have certain percentage of active spatial units across channels. The channels are sorted by increasing "less than 1% of units active" number.

The plot in figure 3.23 shows that units fire homogeneously in conv5-4. Units firing homogeneously are units firing independently, and hence encoding statistically independent concepts. The importance of this result lies in the awareness that the latent factors describing the data must be statistically independent, and hence our Sparse activations are able to encode meaningful information, confirming the efficient coding hypothesis that we wanted to test.

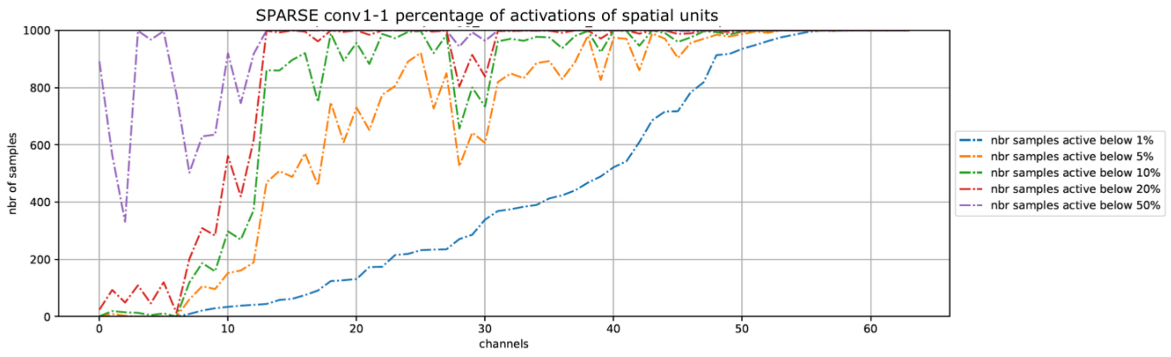


Figure 3.21: SPARSE conv1-1 percentage of spatial units activations

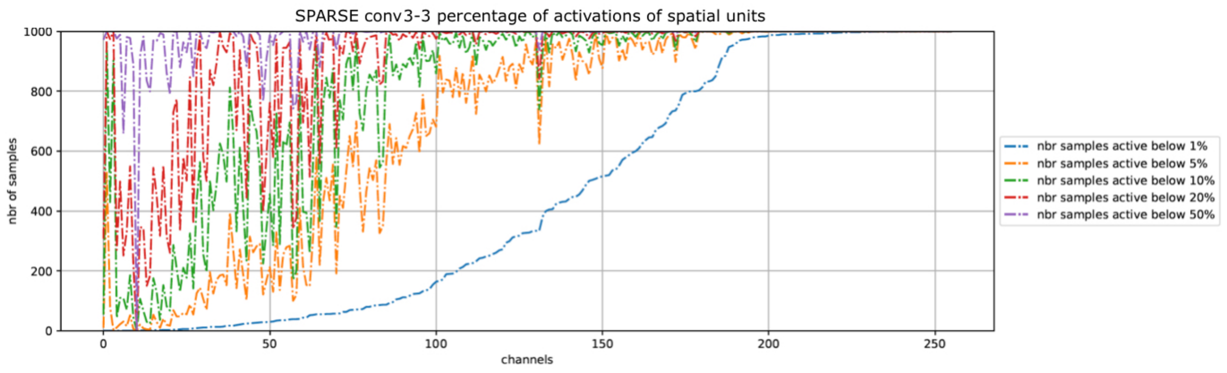


Figure 3.22: SPARSE conv3-3 percentage of spatial units activations

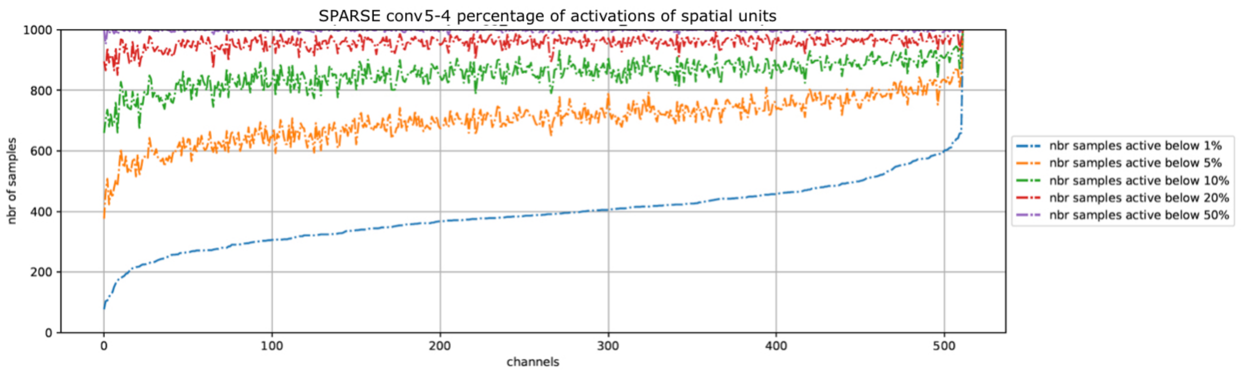


Figure 3.23: SPARSE conv5-4 percentage of spatial units activations

4

Methods

4.1 TensorFlow & ImageNet

The whole implementation work and training has been done using TensorFlow. TensorFlow is a Python based open source software for machine learning. It contains many packages devoted to deep learning, as well as complete networks that can be used to perform experiments. Training and data visualization are done through TensorBoard, TensorFlow's built-in visualization platform. The analyses performed using both TensorFlow and *numpy* in Python.

VGG-19 and the convolutional dictionaries are trained and tested on the ImageNet dataset. ImageNet contains around 14 million images taken from the internet, of 1000 different concepts (*classes*). The labels of images of each concept are human-annotated.

4.2 Poisson noise injection

The Poisson noise generation with a specific offset setting occurs as follows: for the value of any unit activation a (for all the convolutional layers), a noise scale λ and an offset value o,

```
1: Insert offset: $b = o + a$.
2: Determine a mean $\mu : b\ \lambda$.
3: Draw a sample $x$ from a Poisson distribution with mean $\mu$.
4: Re-scale the sample: update $a \leftarrow \frac{x}{\lambda}$
```

The performance of the net is tested for different values of signal-to-noise ratio λ and offset o. Note that the actual variance of the noise injected is λ^{-1}.

A visual example of the Poisson noise injection (assuming offset = 0.0) is reported in figure 4.1. Examples of different Poisson noise distributions are shown in figure 4.2:

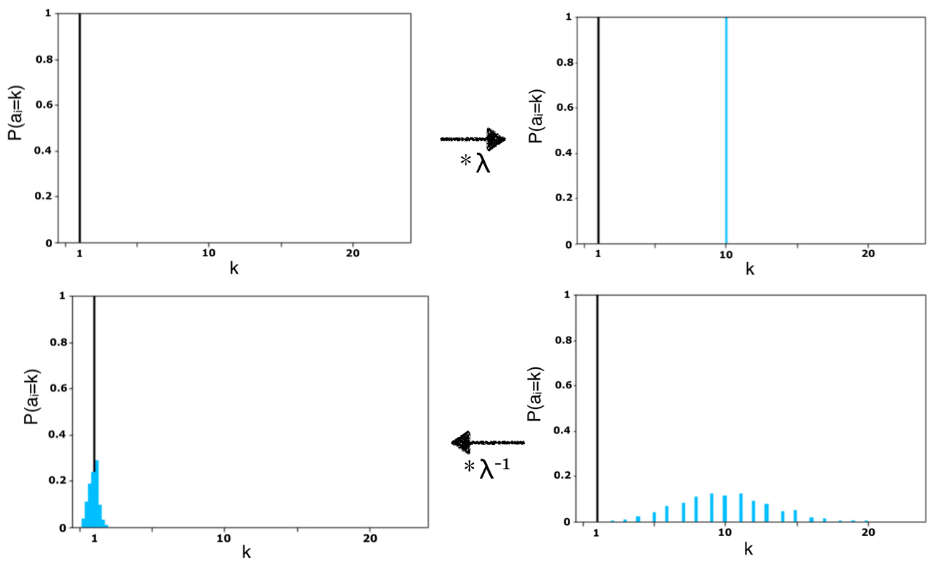


Figure 4.1: Visual example of Poisson noise injection for λ=10

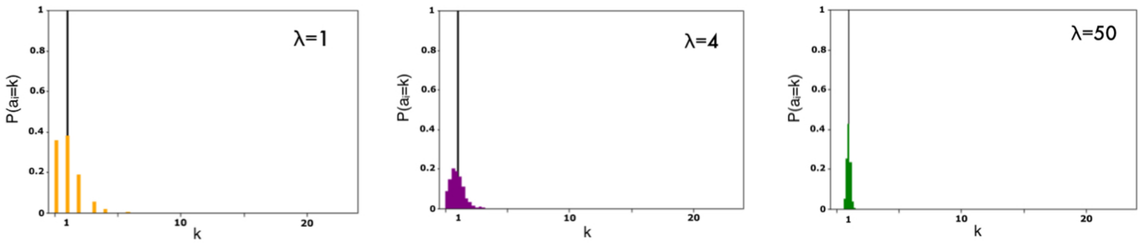


Figure 4.2: Visual examples of Poisson noise distribution for λ=1 (left), λ=4 (center), λ=50 (right)

4.3 Sparse VGG

The algorithm we used to produce the Sparse representations of VGG-19 is a modified version of the one described in Papyan and Romano's paper in [29]. It looks as follows:
For every VGG layer i, take activations X_i

```
1: $\Gamma_i^0 \leftarrow 0$
2: **for** $k = 1 : K$ **do**
3:     $\Gamma_i^k \leftarrow \mathbf{R}_{\xi_i}(\Gamma_i^{k-1} + D_i^T(X_i - D_i\,\Gamma_i^{k-1}))$
4: **end**
```

where:
K = number of inner iterations, in this case set to 20.
D_i = convolutional dictionaries, to be learned during trained.
ξ_i = sparse regularization parameter.
$\mathbf{R}_{\xi_i}(y) = ReLU(y - \xi_i)$. It is a non-negative soft thresholding operator with threshold proportional to ξ_i. The use of the non-negative soft threshold is chosen to restrict

sparse activations to positive values.
Γ_i^K = the sparse representation of X_i achieved after training.
$\bar{X}_i = \mathbf{R}(D_i\ \Gamma_i^K)$ will be referred as the "reconstructed" VGG activations.

Every layer i is independent from the others. During training they are hence treated independently. The sparse activations will still keep the same dimensions of the layers they are representing, and they will be referred to as SPARSE VGG activations (or simply SPARSE activations).

4.3.1 Training

The aim of the training is to find the sparsest representations of the VGG-19 (convolutional) layers activations.
The Loss function defined to accomplish it is:

$$L = \sum_{i\ \in\ layers} L_i^{(rec)}\ +\ L_i^{(sp)}\ +\ L_i^{(CE)} \tag{4.1}$$

where:

$L_i^{(rec)} = \frac{1}{2}||X_i - \bar{X}_i||_2^2$. It is the reconstruction error, the L_2 norm on the difference between VGG and Reconstructed VGG. It ensures that the SPARSE activations are generated to reproduce the VGG activations.

$L_i^{(sp)} = \xi_i\ ||\Gamma_i^K||_1$. It is called sparse loss and it is constituted by an L_1 penalty on the SPARSE activations. It is the contribution that pushes for sparsity.

$L_i^{(CE)} = CE[\bar{X}_i]$. It indicates the Cross Entropy of VGG predictions when substituting activations of layer i with the corresponding reconstructed ones. It induces the network to find solutions that increase the task performance.

The convolutional dictionaries D_i are trained using Adam (a variant of Gradient Descent) optimizer. Fine tuning occurs modifying the sparse regularization term and checking on the accuracy constraint.
It follows a schematic description of the procedure used for training:

1. Images are sent as input to a pre-trained VGG-19. All the inner layers activations and the predictions on the image classes are collected.
2. The predictions are compared with the actual labels of the input and the accuracy is computed. This is referred to as **A**.
3. The activations X_i are used by the Convolutional Dictionaries D_i to produce the corresponding sparse representations Γ_i^K through the Sparse coding algorithm defined in the previous section.
4. The reconstructed VGG $\bar{X}_i$ is computed and fed to VGG instead of its original activations X_i.
5. The predictions of the network are computed again. The cross entropy between the new predictions and the labels ($CE[\bar{X}_i]$) are included in the Loss Function.

6. The accuracy of VGG with activations $\bar{X}_i$ is referred to as $A_{\bar{X}_i}$. The difference in accuracy $\Delta^i_{acc} = \mathbf{A} - A_{\bar{X}_i}$ is calculated and used for an extra constraint on the problem.
The optimal sparse representations Γ^K_i are the sparsest that fulfill the requirement $\Delta^i_{acc} \leq 0.01$.
A visual example a training step is shown in figure 4.3:

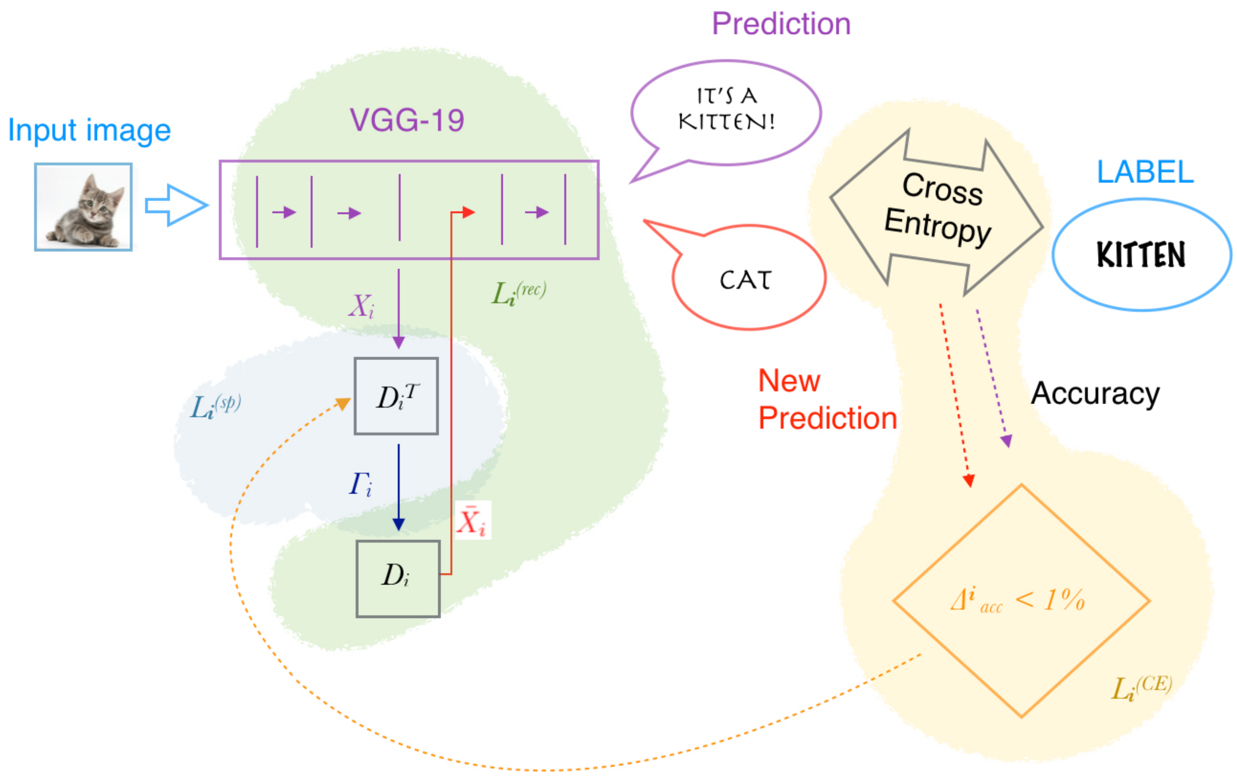


Figure 4.3: Sparse VGG training scheme

5

Conclusion

5.1 Robustness against noise

The experiments involving the introduction of offset and Poisson distributed noise have shown to what extent VGG-19 is robust against different types of noise.

Salient points:
1: The original performance is not considerably affected when the network is subjected to an offset up to 10% of the value of its activations.
2: An drastic decrease in accuracy occurs when the offset level increases.
3: There is the tendency to form a "saturation regime" in performance when applying Poisson distributed noise.
4: The variance of Poisson noise that VGG can tolerate before a drop in accuracy depends on the corresponding level of offset applied to its activations.

The results are summarized in table 5.1. In the table, λ threshold indicates the signal-to-noise ratio of the Poisson distributed noise at which the performance deviates less than 5% from the reference performance, i.e. performance of VGG with the corresponding offset.

VGG with Poisson noise of λ= 3.5 and offset of 10% has an accuracy of 86%. Figure 5.1 shows the distribution of VGG activations with this setting. Considering that the accuracy drop is negligible with this setting, we conclude that VGG is fairly robust against this kind of noise.

Offset	Accuracy	λ threshold
0.0	0.88	3.5
0.1	0.86	3.5
0.25	0.82	20
0.4	0.6	–
0.5	0.29	–

Table 5.1: VGG-19 robustness against noise injection

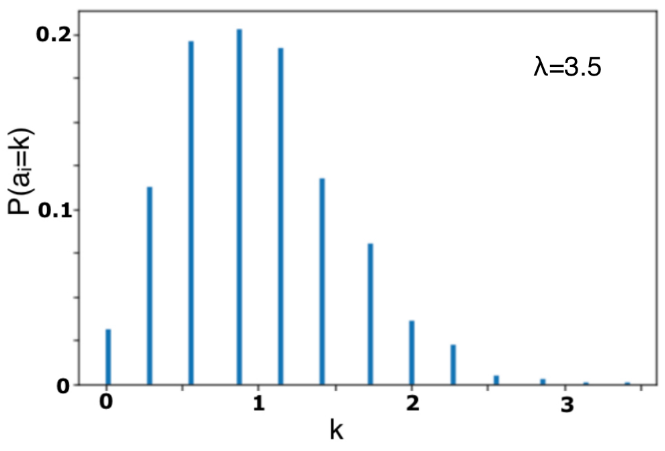


Figure 5.1: Distribution on VGG activations with offset=0.1 and λ=3.5

It is worth to try some comparisons with the brain. VGG performs a recognition task, which is done by our brain in the time used by the visual stimulus to go from the retina to the IT. Hence a Poisson mean λ describes an average number λ of spikes during this task. A λ of 3.5 corresponds to 35 Hz. Recordings from the primate brain show a firing rate between 1 and 10 Hz when recognizing objects. We can conclude that there is a substantial mismatch between VGG and the brain patterns. However, it is important to note that the number of neurons in the Visual Cortex (above 10 billion) is much larger than the number of units in VGG-19 (around 15 million), so the comparison om the performance is not reliable.

5.2 Evidences for efficient coding

From the sparse coding experiment, the main results can be summarized as follows. Going toward deeper layers, the corresponding sparse activations:
1. use fewer units to encode information;
2. exhibit spatial localization;
3. encode more statistically independent concepts.
While evidence 1. is straightforward for efficient coding, 2. and 3. are together clear hints that (the deeper the layer, the more) sparse representations encode semantic concepts. We can conclude that our sparse activations confirm Barlow's hypothesis.

Bibliography

[1] Frisk, D. (2016) A Chalmers University of Technology Master's thesis template for LaTeX. Unpublished.

[2] Warren S. McCulloch and Walter H. Pitts, *A logical calculus of the ideas immanent in nervous activity.* Bulletin of Mathematical Biophysics, 1943.

[3] Tim C. Kietzmann, Patrick McClure and Nikolaus Kriegeskorte, *Deep Neural Networks in Computational Neuroscience.* bioRxiv, 2017.

[4] James J. DiCarlo, David D. Cox. *Untangling invariant object recognition.* Trends in Cognitive Sciences, 2007.

[5] Serge O. Dumoulin and Brian A. Wandell, *Population receptive field estimates in human visual cortex.* Neuroimage, 2008.

[6] Marcel Van Gerven, *Computational foundations of natural intelligence.* bioRxiv, 2017.

[7] David H. Hubel, Torsten N. Wiesel, *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex.* Journal of Physiology, 1962.

[8] David H. Hubel, Torsten N. Wiesel, *Receptive fields and functional architecture of monkey striate cortex.* Journal of Physiology, 1968.

[9] Ryan J. Rowekamp and Tatyana O. Sharpee, *Cross-orientation suppression in visual area V2.* Nature Communications, 2017.

[10] R. Quian Quiroga et al., *Invariant visual representation by single neurons in the human brain.* Nature, 2005.

[11] Robert Desimone, Thomas D. Albright, Charles G. Gross and Charles Bruce, *Stimulus-selective properties of inferior temporal neurons in the macaque.* The Journal of Neuroscience, 1984.

[12] Daniel L. K. Yamins and James J. DiCarlo, *Using goal-driven deep learning models to understand sensory cortex.* Nature Neuroscience, 2016.

[13] Daniel L. K. Yamins et al., *Performance-optimized hierarchical models predict neural responses in higher visual cortex.* PNAS, 2014.

[14] Jonathan Long, Evan Shelhamer and Trevor Darrell, *Fully convolutional networks for semantic segmentation.*

[15] Jason Yosinski, Jeff Clune, Yoshua Bengio, Hod Lipson, *How transferable are features in deep neural networks?.* NIPS, 2014.

[16] Chiyuan Zhang et al., *Understanding deep learning requires re-thinking generalization.* ICLR, 2017.

[17] Matthew D. Zeiler, Rob Fergus, *Visualizing and understanding convolutional networks.* ECCV, 2014.

[18] http://www.scholarpedia.org/article/Neuronal noise

[19] Wei Ji Ma, Jeffrey M. Beck, Peter E. Latham and Alexandre Pouget, *Bayesian inference with probabilistic population codes.* Nature Neuroscience, 2006.

[20] Arno Onken, P. P. Chamanthi R. Karunasekara, Christoph Kayser and Stefano Panzeri, *Understanding Neural Population Coding: Information Theoretic Insights from the Auditory System.* Advances in Neuroscience, 2014.

[21] Federico Carnevale et al., *An Optimal Decision Population Code that Accounts for Correlated Variability Unambiguously Predicts a Subject's Choice.* Neuron, 2013.

[22] Horace B. Barlow, *Possible principles underlying the transformation of sensory messages.* Sensory Communication, 1961.

[23] Julien Mairal, Francis Bach and Jean Ponce, *Task-driven dictionary learning.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2012.

[24] https://www.clarifai.com/technology

[25] Karen Simonyan and Andrew Zisserman, *Very deep convolutional networks for large-scale image recognition.* ICLR, 2015.

[26] http://intellabs.github.io/RiverTrail/tutorial/

[27] https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html

[28] http://machinethink.net/blog/convolutional-neural-networks-on-the-iphone-with-vggnet/

[29] Vardan Papyan and Yaniv Romano et al., *Convolutional neural networks analyzed via convolutional sparse coding.* arXiv, 2016.

[30] https://en.wikipedia.org/wiki/Poisson-distribution

A

Appendix 1

A.1 Complete plots

Here are reported the complete plots of the different analyses performed:

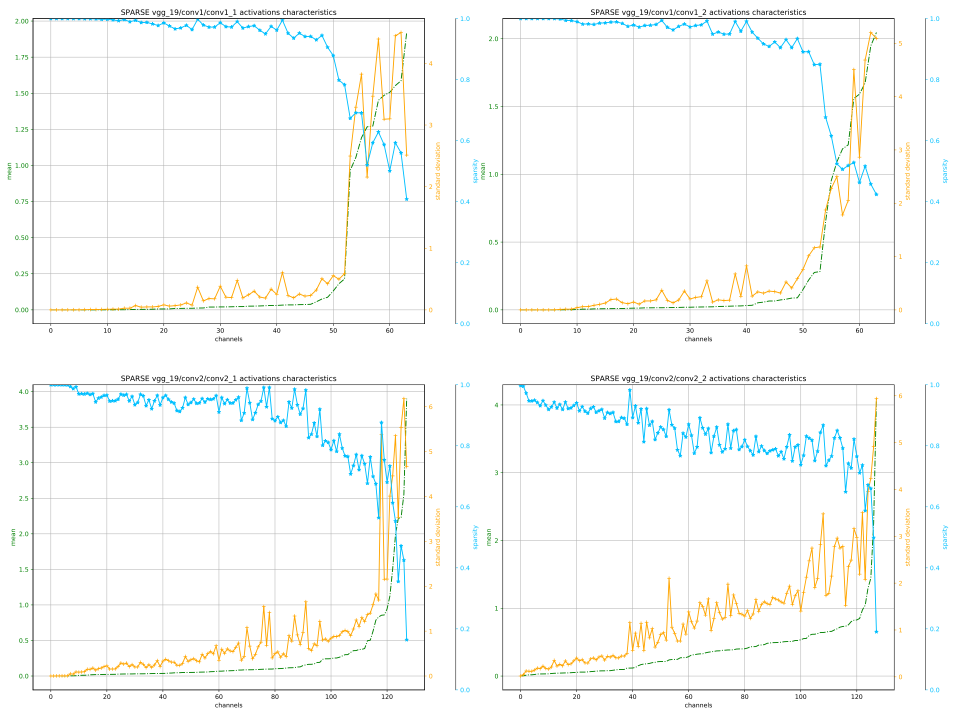


Figure A.1: Mean, standard deviation and sparsity across channels of SPARSE activations from conv1-1 to conv2-2, sorted by increasing mean

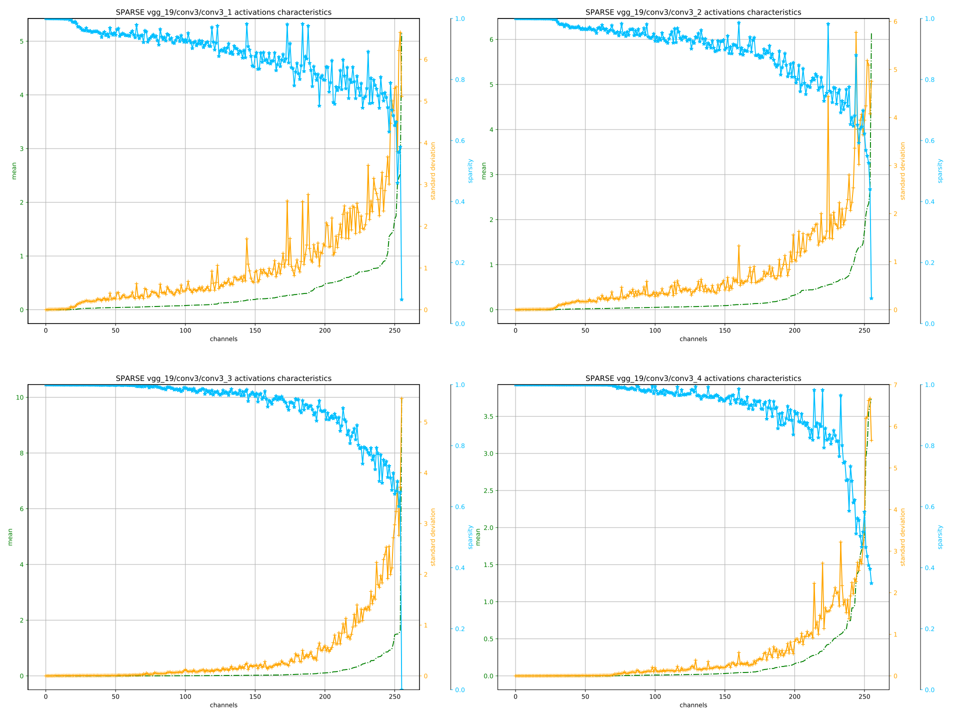


Figure A.2: Mean, standard deviation and sparsity across channels of SPARSE activations from conv3-1 to conv3-4, sorted by increasing mean

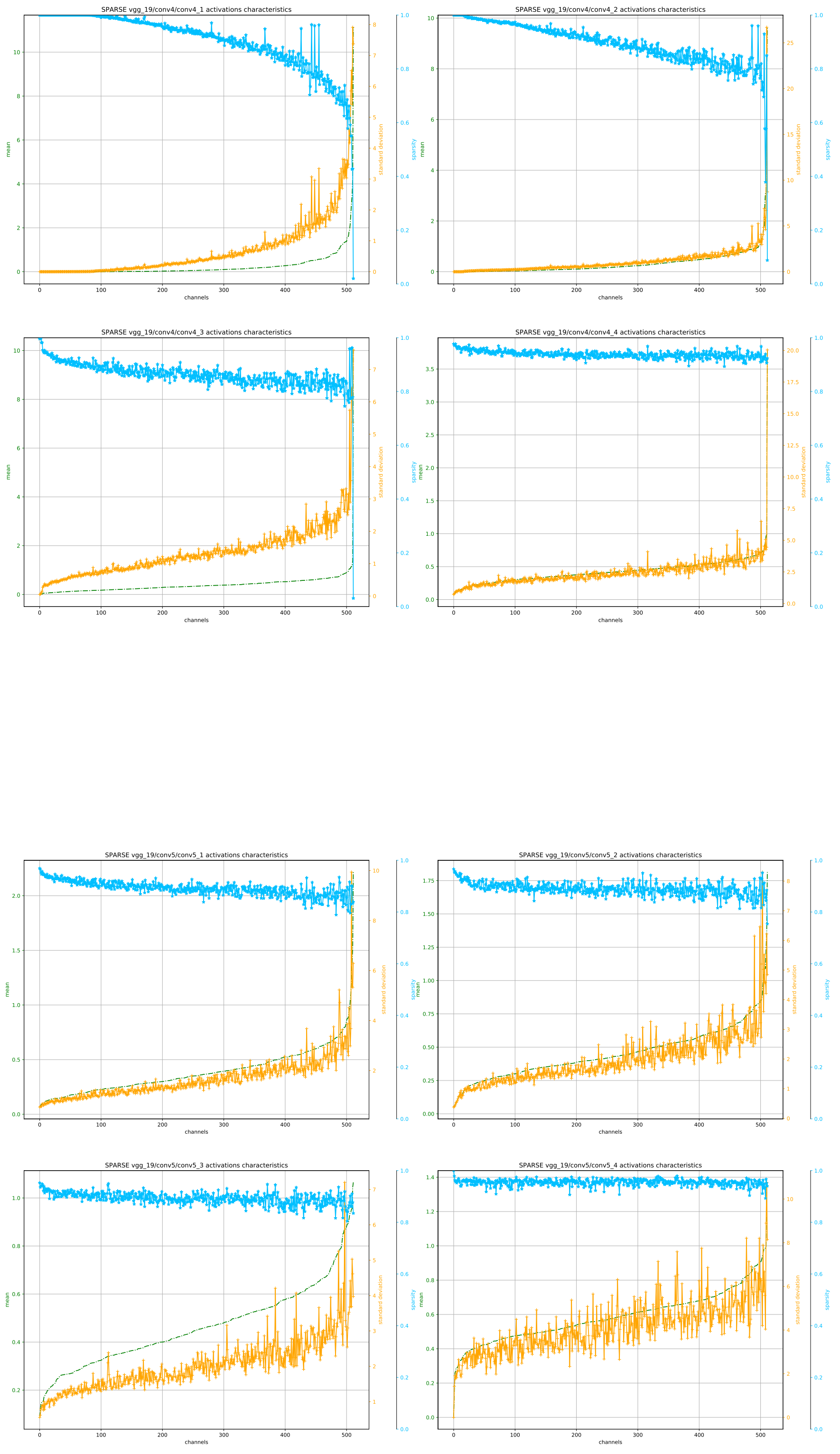


Figure A.3: Mean, standard deviation and sparsity across channels of SPARSE activations from conv4-1 to conv5-4, sorted by increasing mean

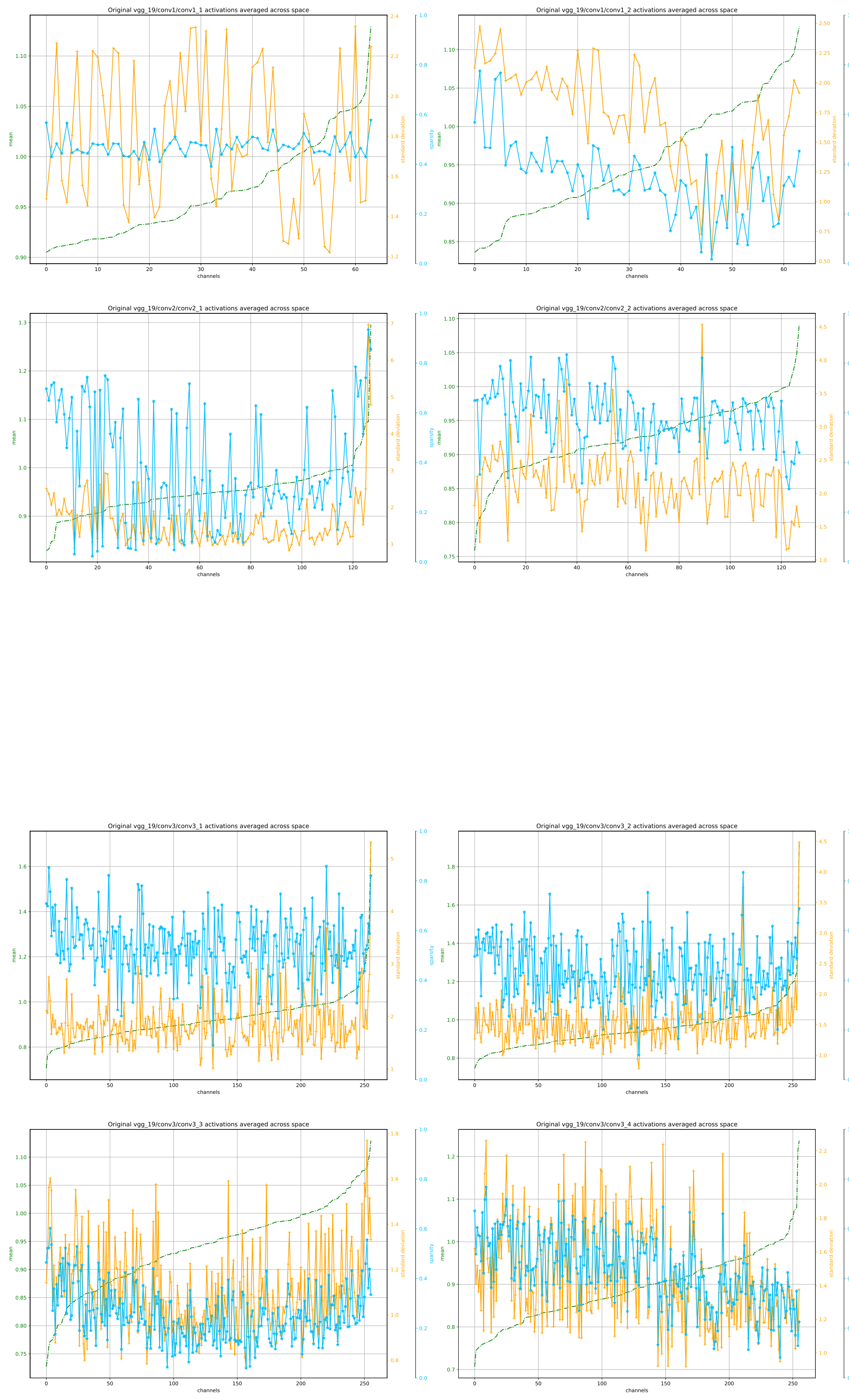


Figure A.4: Mean, standard deviation and sparsity across channels of VGG activations from conv1-1 to conv3-4, sorted by increasing mean

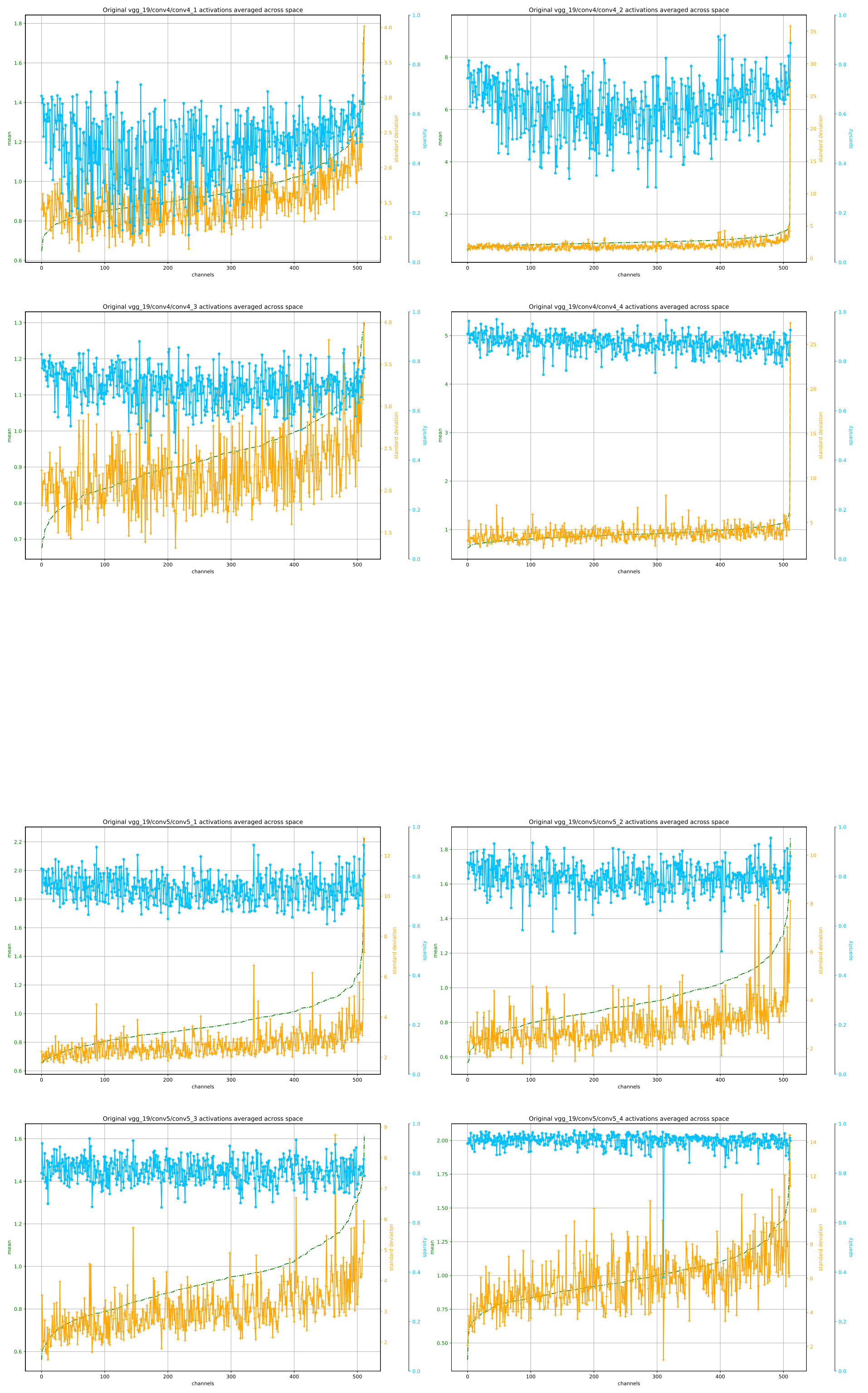


Figure A.5: Mean, standard deviation and sparsity across channels of VGG activations from conv4-1 to conv5-4, sorted by increasing mean

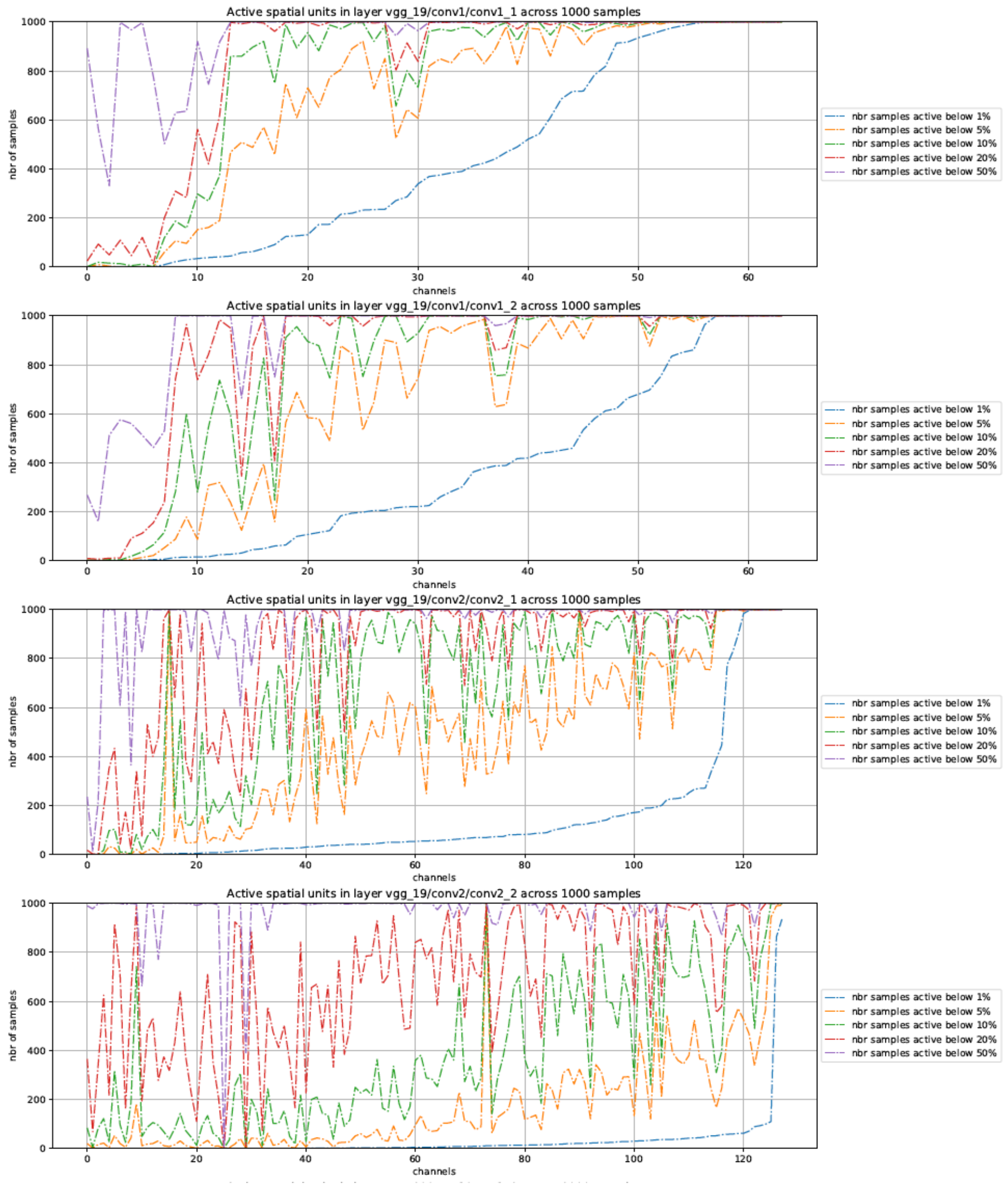


Figure A.6: Spatial activations percentage from SPARSE conv1-1 to SPARSE conv2-2

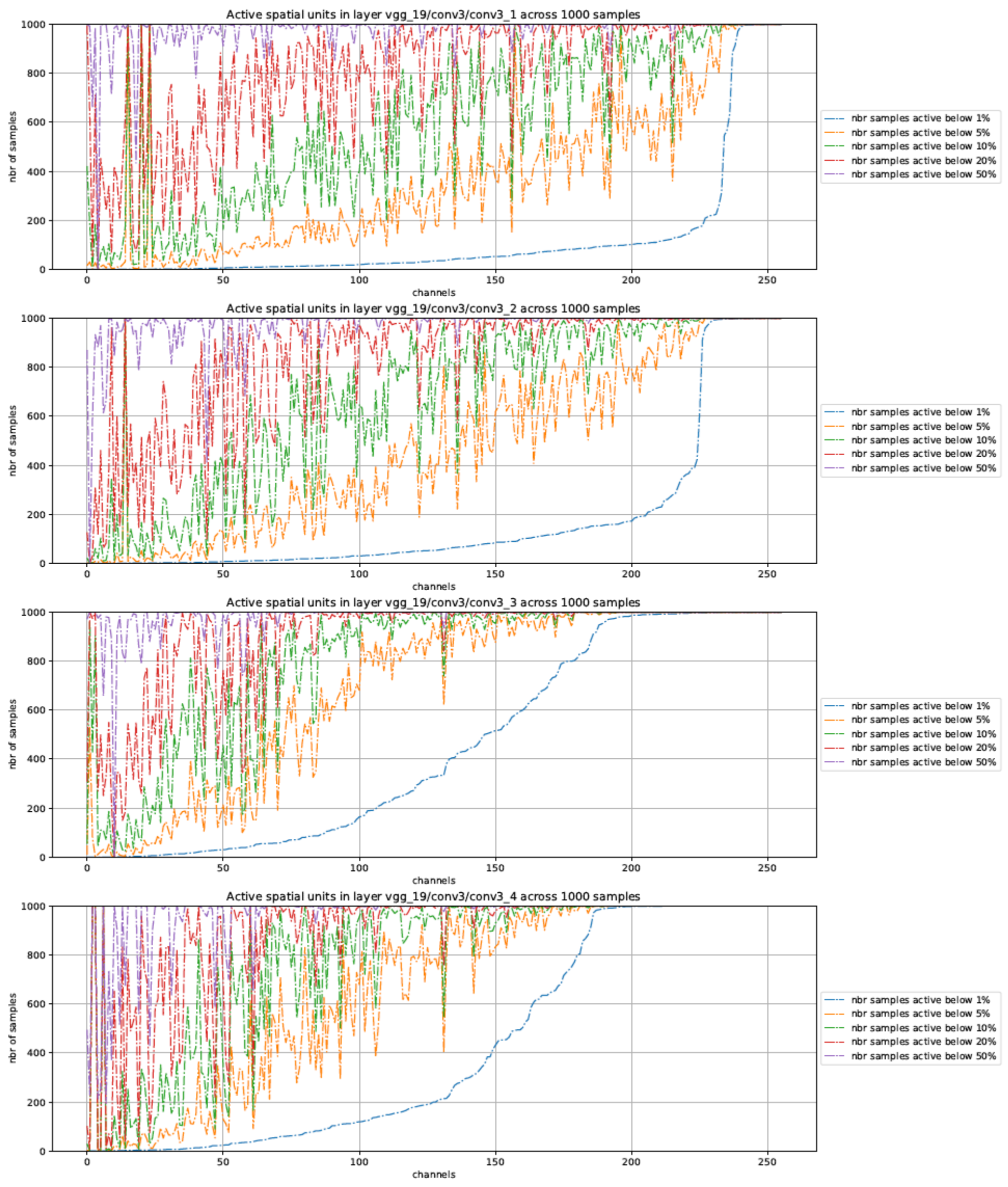


Figure A.7: Spatial activations percentage from SPARSE conv3-1 to SPARSE conv3-4

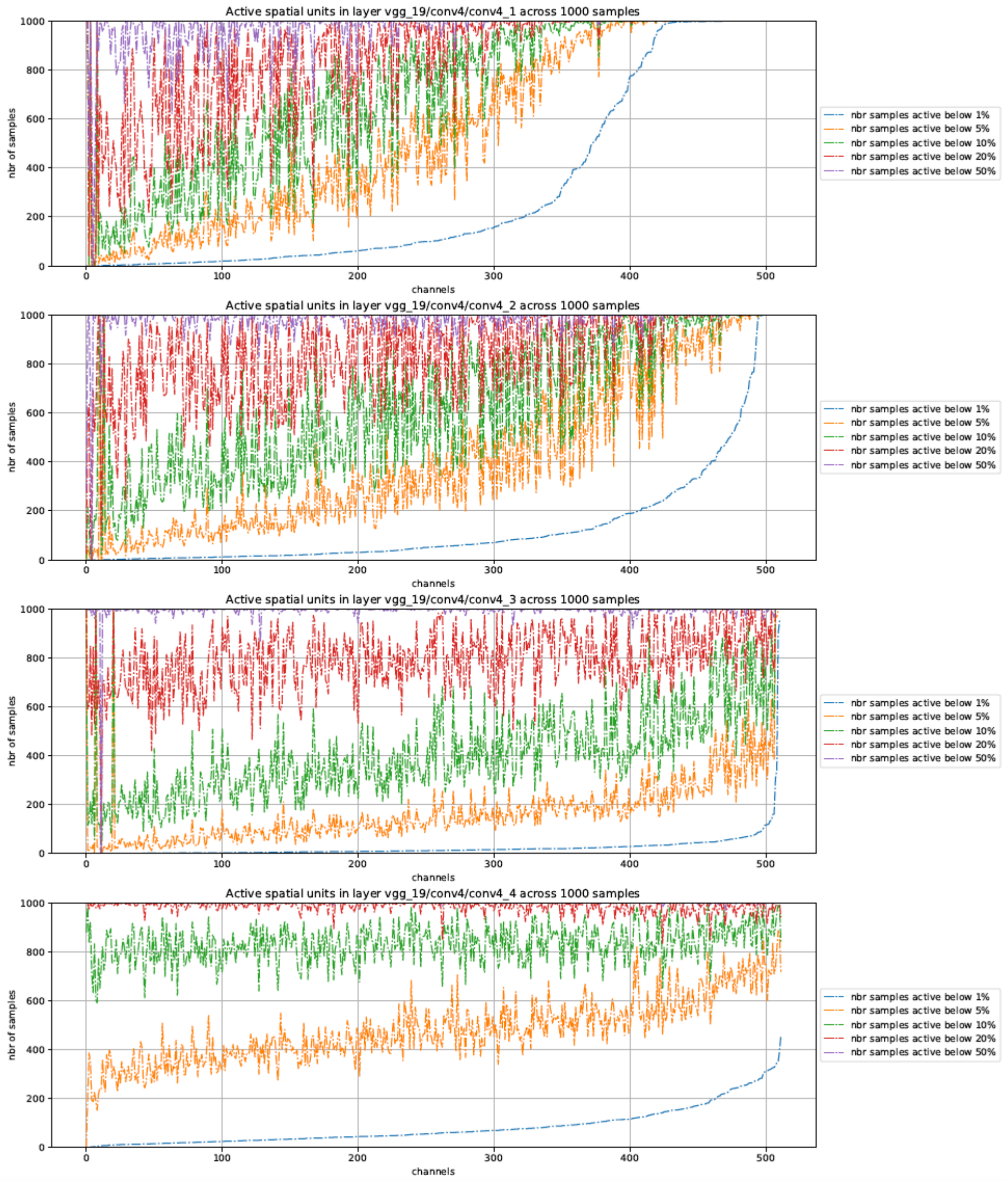


Figure A.8: Spatial activations percentage from SPARSE conv4-1 to SPARSE conv4-4

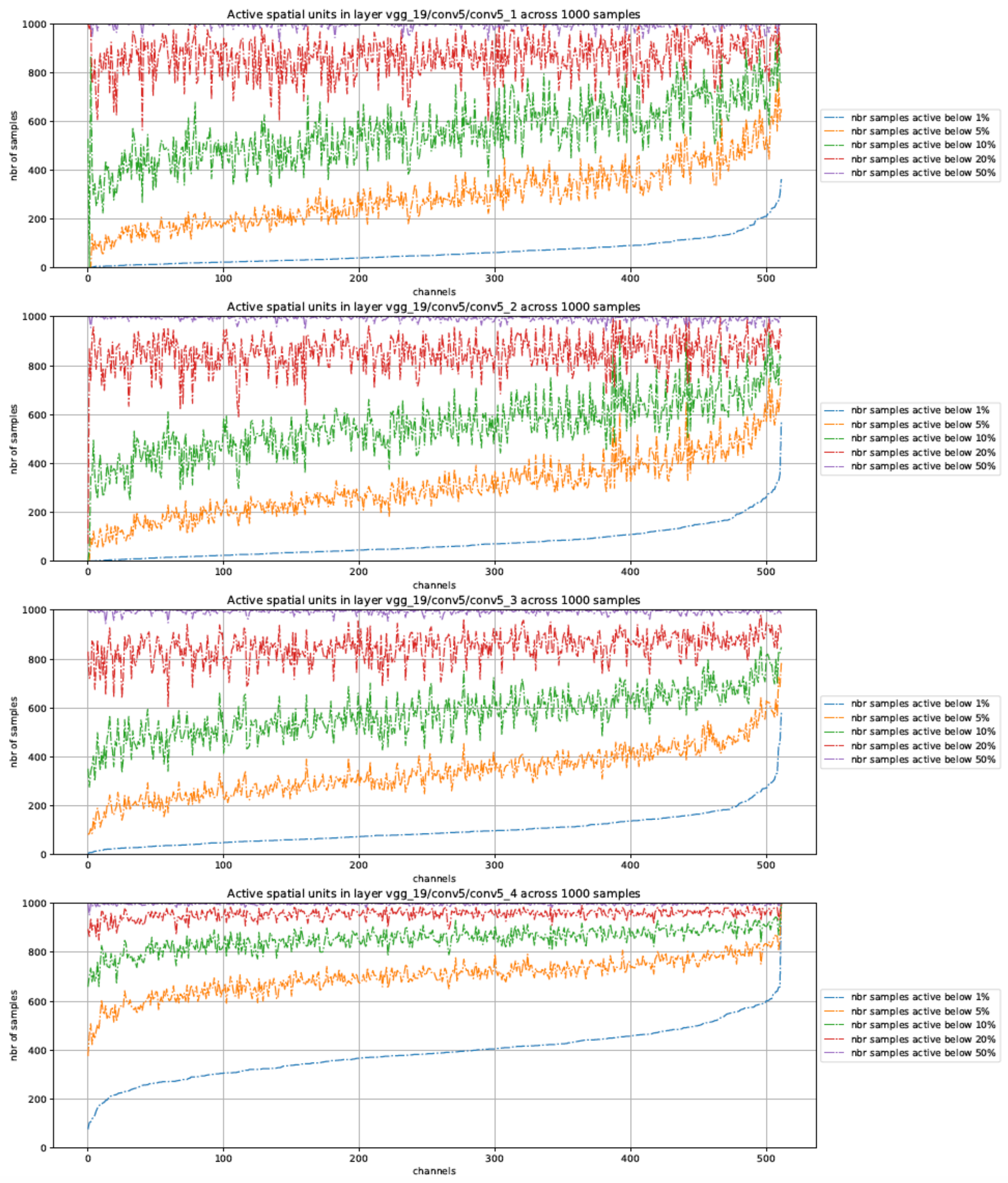


Figure A.9: Spatial activations percentage from SPARSE conv5-1 to SPARSE conv5-4