Extending Layer-wise Relevance Propagation using Semiring Annotations

Antoine Groudiev
L3, DI, ENS

Abstract

Recently, neural networks allowed computers to solve numerous problems from diverse machine learning fields, such as natural language processsing and computer vision. Compared to traditional algorithms, machine learning models have proven both more successful and more difficult to interpret. Neural networks are considered as black boxes unable to easily explain themselves, that is justifying the reasons that led them to make a prediction. Layer-wise Relevance Propagation (LRP) is a technique that has been introduced to provide explanability by identifying the input features relevant to the output choice. In parallel, research in the databases field developed annotations techniques to compute provenance for queries. In this paper, we extend LRP propagation rules to semiring-based provenance annotations of the network, and implement semiring-based propagation rules for computer vision models of different scales.

1 Introduction

1.1 Problem statement

Deep neural networks have proven successful for solving with high accuracy machine learning problems. The expressivity of the class of functions generated by neural networks, combined with the relative simplicity of their training, make such models versatile tools to learn the relationship between the inputs and outputs of a dataset.

However, this versatility comes at the cost of poor interpretability: a neural network simply represents a function from one high-dimensional space to another, but provides no justification nor explanation for a given execution. If metrics such as the accuracy over a testing set provide confidence in the fact that the model is able to correctly classify inputs similar to the training set, no guarantee is given that the model generalizes well. Real-world examples show that networks can overfit the input data, or even take shortcuts instead of learning the intended solution [4]. For the user to have confidence in its predictions, a neural network should therefore be able to highlight the patterns in the input data that it actually learned.

1.2 Layer-wise Relevance Propagation

Layer-wise Relevance Propagation (LRP) [1] has been introduced as a technique to explain an execution of a neural network. LRP is a procedure propagating the output of the function backward in the network, using diverse rules to compute the *relevance* of a neuron depending on the relevances of the neurons of the upstream layer. LRP introduces the notion of *relevance score* for a neuron, intuitively quantifying the contribution of this neuron to the classification of final classification. A high relevance score indicates that the neuron led to the activation of the considered output; a negative relevance score represents neurons that increased the activation of another output neuron instead of the one considered.

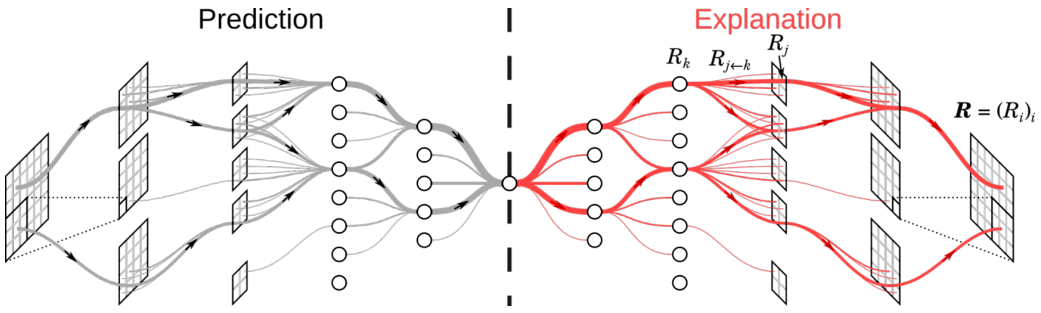

Figure 1.1: Illustration of LRP. The leftmost part corresponds to the forward pass, while relevance propagation is visualized in the rightmost part.[1]

1.2.1 Setup and notations

In the following, we consider a deep neural network used for a classification task. We assume that it uses the rectifier activation function[2], which is the case in most applications. To ease the notation, we will not consider biases but instead assume that the first neuron of each layer represents the bias.

Let L be the number of layers of the network, including the input and output layers. We denote by $\left(a_k^{(l)}\right)_k$ the activations of the network. Notably, $\left(a_k^{(1)}\right)_k$ is the input data, and $\left(a_k^{(L)}\right)_k$ is the ouput prediction. We denote by $\left(w_{j,k}^{(l)}\right)_{j,k}$ the weights connecting the l-th layer to the $(l+1)$-th layer. To simulate the biases using only weight matrices, we set:

$$\forall l \in [\![1, L]\!], \quad a_0^{(l)} = 1$$

and we define $w_{0,k}^{(l)}$ to be the bias of the k-th neuron of the $(l+1)$-th layer. The forward propagation rule of a deep rectifier network is therefore:

$$\forall l \in [\![1, L-1]\!], \forall k, \quad a_k^{(l+1)} = \text{ReLU}\left(\sum_j a_j^{(l)} w_{j,k}^{(l)}\right) = \max\left(0, \sum_j a_j^{(l)} w_{j,k}^{(l)}\right) \tag{1.2.1}$$

We denote by $R_j^{(l)}$ the relevance of the j-th neuron of the l-th layer. We assume that the output layer represents a one-hot encoding, that is that the belonging of the input to the i-th class is represented by an output vector is of the form $(0, \ldots, a_i^{(L)}, \ldots, 0)$, where the only non-null coefficient is in the i-th position. Finally, we denote by y the label of a classified input. To the label $y = i$ is associated the output vector $(0, \ldots, a_i^{(L)}, \ldots, 0)$.

1.2.2 Propagation rules

Relevance scores are initialized for the output layer, and are set to the output activation for the correct class, that is:

$$R_i^{(L)} = \begin{cases} a_i^{(L)} & \text{if } i = y \\ 0 & \text{otherwise} \end{cases} \tag{1.2.2}$$

The simplest LRP rule is called LRP-0. It propagates the relevance to a neuron of the lower layer proportionnaly to its contribution to each of the neuron of the next layer:

$$R_j^{(l)} = \sum_k \frac{a_j^{(l)} w_{j,k}^{(l)}}{\sum_{j'} a_{j'}^{(l)} w_{j',k}^{(l)}} R_k^{(l+1)} \tag{1.2.3}$$

[1] Figure adapted from "Layer-wise Relevance Propagation" – Fraunhofer Institute for Telecommunications

[2] That is $\text{ReLU}(x) = \max(0, x)$, where ReLU stands for *Rectified Linear Unit*.

The denominator $\sum_{j'} a_{j'}^{(l)} w_{j',k}^{(l)}$ guarantees a conservation property, that is that for any layer l:

$$\sum_j R_j^{(l)} = \sum_k R_k^{(l+1)}$$

This allows to keep the information regarding the total activation of the final layer.

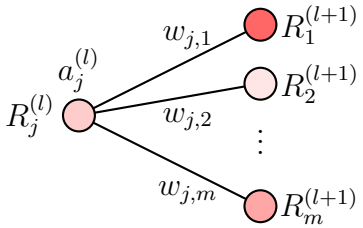

Figure 1.2: Node-level illustration of relevance propagation.

The application of this simple rule can lead into noisy results that do not scale well. An overview of variations of LRP-0 is provided by [9]; not all rules are suitable for all layers. Complex deep neural networks architectures benefit from enhanced rules such as LRP-ε or LRP-γ, which provide more stable explanations.

Notably, the input layer must be handled using a different rule, since it does not receive its input from ReLU activations, but directly from the input data. The $z^{\mathcal{B}}$ rule is used in [9] to propagate from layer 2 to 1 (input layer):

$$R_j^{(1)} = \sum_k \frac{x_k w_{j,k} - l_j w_{j,k}^+ - h_j w_{j,k}^-}{\sum_{j'} x_k w_{j',k} - l_j w_{j',k}^+ - h_j w_{j',k}^-} R_k^{(2)} \tag{1.2.4}$$

where $(\cdot)^+ = \max(0, \cdot)$, $(\cdot)^- = \min(0, \cdot)$. The parameters l_i and h_i respectively define the theoretical minimum and maximum values of the inputs x_i. For instance, we might have $l_i = 0$ and $h_i = 255$ for pixels over 8 bits.

1.2.3 LRP results and pertinence

Previous works ([1], [9]), as well as the methods that we will introduce later on, were experimentaly tested on models trained over two datasets: fully-connected networks trained on the MNIST handwritten digits dataset [6], and deep convolutional networks pre-trained on the ImageNet visual database such as VGG-16 [15].

Figure 1.3 shows relevance for a simple model: a fully-connected deep rectifier network with layers of sizes 28×28, 300, 100 and 10, trained on the MNIST dataset. Pixels highlighted in red have a positive relevance while blue pixels have a negative relevance. As expected, the relevant pixels to classify this image as a `0` are the white pixels in the input image.

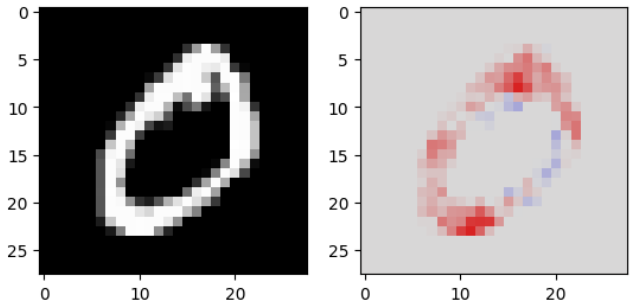

Figure 1.3: Input image and pixel-wise explanation of the output neuron `0`

Figure 1.4 provides a visualization for a more complex example, an execution of the VGG-16 network over a 224×224 image.

Figure 1.4: Input image and relevance for the class "`castle`" of the VGG-16 network

Note that the castle part of the image is highlighted in red as intended. Furthermore, both the street sign and the street light have strong negative relevance; those two objects correspond to other classes of the ImageNet dataset (`street sign (919)` and `traffic light (920)`). Those two elements of the image would have positive relevance for LRP starting from the output neurons `919` and `920`.

1.3 Semiring-based provenance annotations

In parallel, the notion of data provenance in databases theory developed formal solutions to a similar problem to ours. Data provenance aims at *explaning* a query by highlighting the tuples in the original database that led to the presence of a certain tuple in the query result. If contexts are different, deep neural networks explanation and data provenance share the same general setup: identifying a subset of the input that directly implied a certain output.

A framework to approach data provenance is *provenance semirings*, introduced in [5], annotates tuples using abstract elements of a semiring and apply semiring operations to the tuples appearing in the query. As the query is executed, information about the provenance of the intermediate results is aggregated, resulting in an abstract formula that can be concretized by substituting abstract elements and operations by a concrete semiring. Similarly, in the context of graph databases, edges can be annotated to derive a variety of properties of the query result. [10]

In the following, we provide a mathematical definition of a semiring as well as semiring examples suited for the application to deep neural networks.

Definition (Semiring)**.** A *semiring* is an algebraic structure generalizing the notion of rings. A semiring $(\mathbb{K}, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ is composed of a set $\mathbb{K}$, binary operators $\oplus$ and $\otimes$ such that $\otimes$ distributes over $\oplus$, verifying the following properties:

- $(\mathbb{K}, \oplus, \mathbb{0})$ is a commutative monoid
- $(\mathbb{K}, \otimes, \mathbb{1})$ is a monoid such that $\mathbb{0}$ is absorbing

Example. $(\mathbb{R}, +, \times, 0, 1)$ is a semiring. While it has no direct interpretation is the context of databases, we will see that is corresponds to the basic real-valued LRP.

Example (Boolean semiring)**.** $(\mathbb{B}, \vee, \wedge, \bot, \top)$ where $\mathbb{B} := \{\bot, \top\}$ is a semiring. Its use in databases provenance interprets as the existence of a path between two vertices, using edge weights as the number of different paths between two adjacent vertices.

Example (Counting semiring)**.** $(\mathbb{N}, +, \times, 0, 1)$ is a semiring. For a non-cyclic graph database, its use allows to compute the total number of paths between two vertices, using edge weights as the number of different paths between two adjacent vertices.

Example (Viertbi semiring)**.** $([0, 1], \max, \times, 0, 1)$ is a semiring. For a non-cyclic graph database where the annotations are interpreted as a "confidence" measure, its use allows to compute the confidence score of the result of a query.

2 Extending LRP using Semiring Annotations

We aim at extending Layer-wise Relevance Propagation by annotating the computational graph of a neural network with semiring elements.

2.1 Semiring generalization of the LRP rules

LRP rules all inherit the same structure, in which the relevance of a pixel can be expressed as the weighted sum of the relevance of pixels of the next layer:

$$R_j^{(l)} = \sum_k d_{j,k}^{(l)} \cdot R_k^{(l+1)} \tag{2.1.1}$$

where the $d_{j,k}^{(l)}$ are some coefficients proportional to the extent to which neuron j of the l-th layer has contribued to make neuron k of the $(l+1)$-th layer relevant.

Equation 2.1.1 is the special real-valued case of a more general rule. Let $(\mathbb{K}, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ be a semiring, and $(\Theta^{(l)})_l$ be annotation functions, associating to each matrix of coefficients $d^{(l)}$ a matrix of semiring elements $\Theta^{(l)}(d)$. In particular, note that $d_{i,j} \mapsto \Theta^{(l)}(d)_{i,j}$ does not need to be a mapping: to identical coefficients can be associated different annotations. In the following examples, we will mostly use element-wise annotations functions of the form $\mathbb{R} \longrightarrow \mathbb{K}$.

We call $\mathbb{K}$-relevance for output neuron y the quantity $R \in \mathbb{K}$ inductively defined by:

$$R_i^{(L)} := \begin{cases} \mathbb{1} & \text{if } i = y \\ \mathbb{0} & \text{otherwise} \end{cases} \tag{2.1.2}$$

and

$$R_j^{(l)} := \bigoplus_k \Theta^{(l)}(d)_{j,k} \otimes R_k^{(l+1)} \tag{2.1.3}$$

Note that choosing $(\mathbb{R}, +, \times, 0, 1)$ as a semiring and $\Theta^{(l)} = x \mapsto x$ retrieves classical LRP up to a multiplicative factor of $a_i^{(L)}$ on each layer (which does not appear on the semiring initialization).

Equations 2.1.2 and 2.1.3 can be intuitively interpreted in two ways. Firstly, they can be seen as the abstraction of an LRP computation using formal semiring elements. Computing $\mathbb{K}$-relevance for an execution of the network results in an abstract formula in terms of elements of $\mathbb{K}$ and operations $\oplus$ and $\otimes$. While this might be useful in the context of graph provenance, visualizing abstract formulas is difficult because of the size of computational graphs associated to neural networks. Furthermore, the general structure of the formulas is always the same for a specific model, since neural networks usually have very simple computational graphs structures.

Therefore, a second approach that fits the graph properties of neural networks is to interpret semiring LRP as operations on an annotated circuit, similarly to [13]. For instance, in the case of the boolean semiring, computing $\mathbb{B}$-relevance consists in taking for each node the logical conjunction of the annotations of its outgoing edges.

Similarly to the implementation of provenance in databases by tools such as ProvSQL [14], we implemented a Python module on top of PyTorch[3], providing an interface to compute $\mathbb{K}$-relevance for any custom semiring $\mathbb{K}$.

2.2 Results over the MNIST dataset

We provide experimental results of the implementation of semiring-based LRP. In the following, we use the same fully-connected network as in Figure 1.3, with layers of sizes 28×28, 300, 100 and 10 and trained on the MNIST dataset.

2.2.1 Boolean Semiring

Boolean relevance can be computed using the boolean semiring $(\mathbb{B} = \{\bot, \top\}, \vee, \wedge, \bot, \top)$, and an element-wise annotation function Θ of the form:

$$\begin{aligned} \Theta : \mathbb{R} &\longrightarrow \mathbb{B} \\ x &\longmapsto \begin{cases} \top & \text{if } x \geqslant \theta \\ \bot & \text{otherwise} \end{cases} \end{aligned} \tag{2.2.1}$$

where θ is a hyperparameter called the threshold. A higher threshold reduces the number of $\top$ coefficients, ultimately reducing the number of input neurons having a relevance equal to $\top$. Choosing $\theta = 10^{-9}$ filters out null and negative contributions to the output while maintaining a sufficient amount of activated inputs for visualization.

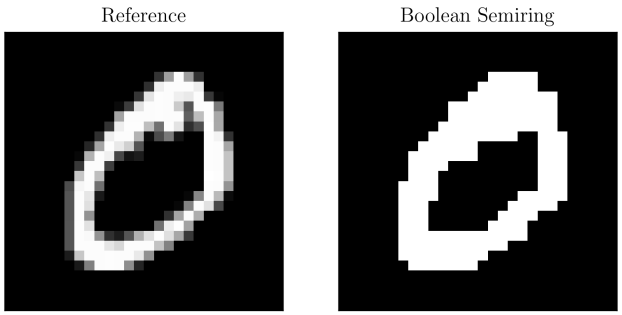

Figure 2.1: Input image and $\mathbb{B}$-relevance for the output neuron `0` for $\theta = 10^{-9}$

[3] `https://pytorch.org`

The full propagation equation becomes:

$$R_j^{(l)} := \bigvee_k \Theta\left(\frac{a_j \cdot w_{j,k}}{\sum_{j'} a_{j'} \cdot w_{j',k}}\right) \wedge R_k^{(l+1)} \tag{2.2.2}$$

$\mathbb{B}$-relevance provides a higher level explanation, highlighting large zones of the input image which contribute the most to the final classification. Naturally, information about nuances in the contribution is lost. Intuitively, activated pixels (input neurons with relevance $\top$) are neurons such that there exists a "relevant" path from this neuron to the output neuron of the class `0`. A relevant path is a path in which all edges have a weight higher than the threshold θ. While this condition might seem quite restrictive, note that computation graphs for neural networks are densely connected: there is $784 \times 300 \times 100 = 23\,520\,000$ different paths connecting one input pixel to the output neuron of the class `0`.

2.2.2 Counting Semiring

The counting semiring $(\mathbb{N}, +, \times, 0, 1)$ extends boolean relevance by enumerating the number of relevant paths starting from each input pixel. Its element-wise annotation function Θ is mostly identical:

$$\begin{aligned} \Theta : \mathbb{R} &\longrightarrow \mathbb{N} \\ x &\longmapsto \begin{cases} 1 & \text{if } x \geqslant \theta \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{2.2.3}$$

but operations $+$ and $\times$ bring more expressivity to the framework, bringing more nuance in the highlighted zones.

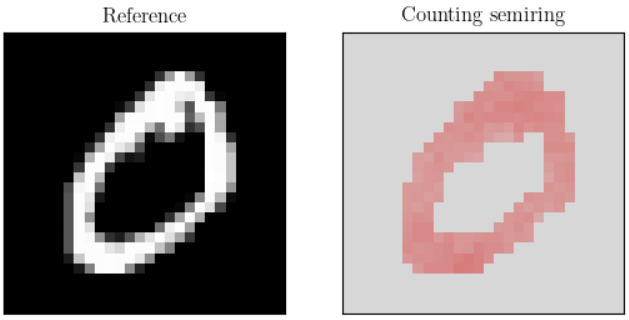

Figure 2.2: Input image and $\mathbb{N}$-relevance for the output neuron `0` for $\theta = 10^{-9}$. Highest relevance is 2098.

The full propagation equation becomes:

$$R_j^{(l)} := \sum_k \Theta\left(\frac{a_j \cdot w_{j,k}}{\sum_{j'} a_{j'} \cdot w_{j',k}}\right) \times R_k^{(l+1)} \tag{2.2.4}$$

2.2.3 Viterbi Semiring

The Viterbi semiring $([0, 1], \max, \times, 0, 1)$ replaces the $+$ of the classical real semiring by a max. In a context where neurons can have the same weighted sum of relevance, taking the maximum of the relevance emphasizes the most important neurons, giving a more contrasted visualization than classical LRP.

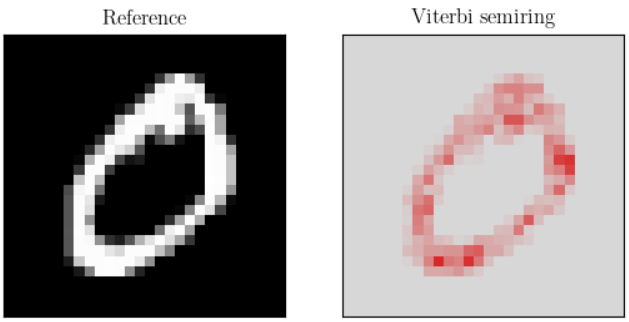

Figure 2.3: Input image and $[0, 1]$-relevance for the output neuron `0`

To ensure that values belong to the $[0, 1]$ set, the following rule is used:

$$R_j^{(l)} = \max_k \underbrace{\left(\frac{|a_j \cdot w_{j,k}|}{\max_{j'} |a_{j'} \cdot w_{j',k}|} \right)}_{\in [0,1]} \cdot R_k^{(l+1)} \tag{2.2.5}$$

In practice, a small amount such as `1e-9` can be added to the denominator to prevent from dividing by zero in the case where each $|a_{j'} \cdot w_{j',k}|$ is null.

3 Applications

We explore multiple applications of semiring-based LRP, going beyond the scope of simple execution-wise explanations. While LRP was intended as a tool to interpret a single execution of the network, we believe that averaging relevance scores of neurons over multiple executions can produce a meaningful metric of the "importance" of each neuron. Through three applications, we put forwards both qualitative and quantitative arguments supporting this view.

3.1 Image mask computation

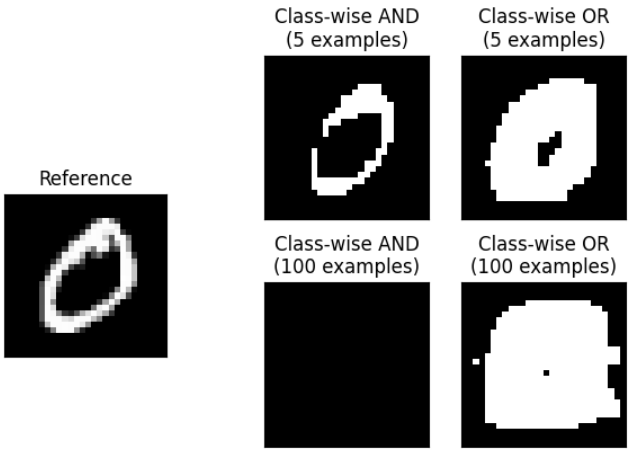

Figure 3.1: Class-wise mask for $\mathbb{B}$-relevance of the class `0`

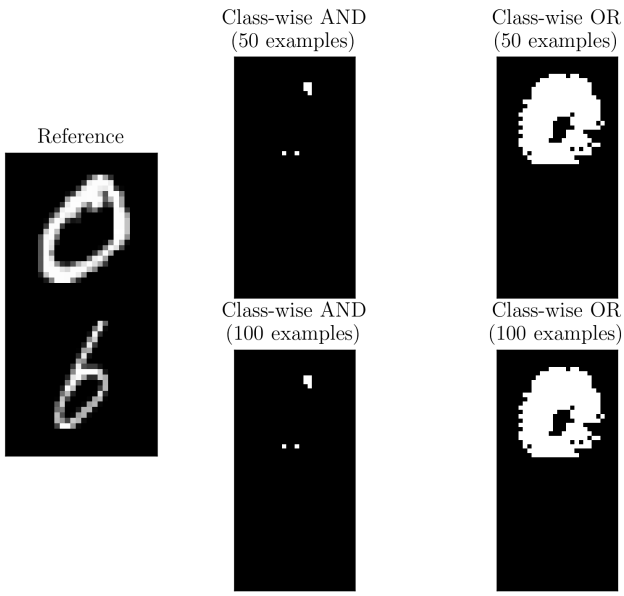

Figure 3.2: Class-wise mask for $\mathbb{B}$-relevance of the class `0`

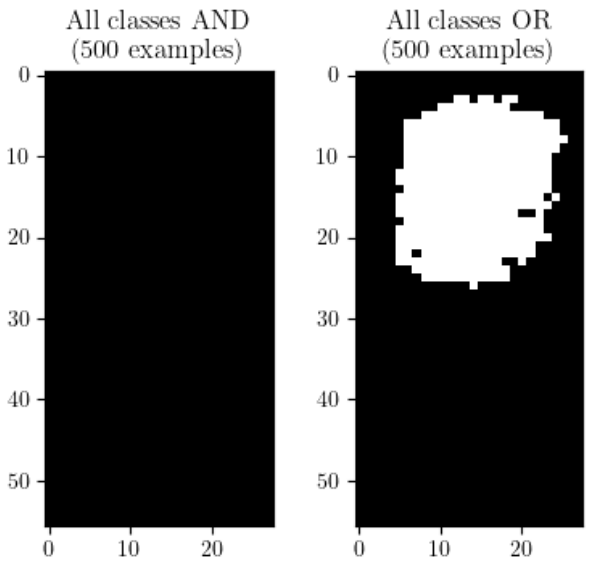

Figure 3.3: Class-wise mask for $\mathbb{B}$-relevance of the class `0`

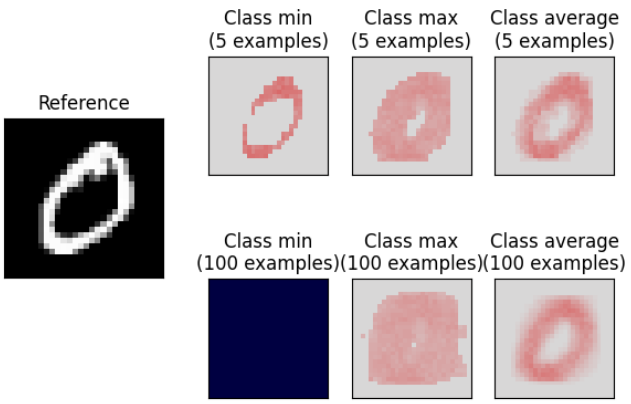

Figure 3.4: Class-wise mask for $\mathbb{N}$-relevance of the class `0`

Figure 3.5: All classes mask for ℕ-relevance

3.2 Network pruning using LRP ranking

The accuracy of modern neural networks often come at the cost of large model size, increasing both storage, computational cost and inference time. To reduce the size of such models, an efficient technique called *network pruning* has been studied [2, 8], in which neurons are removed from the network. A simple approach to selecting which neurons to prune is to select those with the smallest ℓ_1 or ℓ_2 norm: that is, those whose associated column in the weight matrix has the smallest norm. While this selection method yields good results, we show that removing neurons with the smallest average relevance – in the sense of LRP – allows the network to perform better.

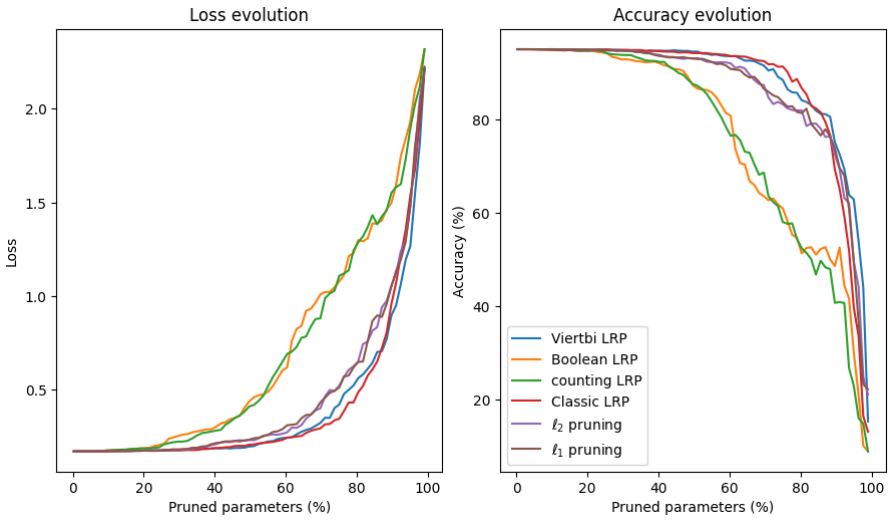

Figure 3.6: Accuracy and loss evolution during network pruning for different neuron selection methods. Neurons were pruned on the first layer of the same fully-connected network trained on the MNIST dataset. Accuracy and loss are computed over the MNIST test set.

Figure 3.6 compares multiple selection methods. For each method, the network was evaluated multiple times on a subset of the MNIST test set, each time with an increasing number of neurons pruned, selected using the specific method. For LRP-based methods, the relevance of each neuron is computed by taking its average relevance on LRP results over images of the train set, and neurons with the lowest absolute relevance are pruned first.

Three curve clusters can be identified: boolean and counting LRP perform much worse than other selection methods, which can be explained by the lack of granularity in the explanation that these techniques provide. ℓ_1 and ℓ_2 norm selection methods provide a baseline to compare LRP techniques to, and both perform similarly well. Finally, classic LRP and Viterbi LRP outperform the norm-based methods by a small amount: this confirms the intuition that LRP provide a more subtle explanation than simply the magnitude of the weights connected to a neuron.

3.3 Comparison to image perturbation

In parallel to LRP, other methods aiming at providing a visual explanation of the decision of a classifier have emerged, such as [7, 16]. An interesting approach, introduced in [3], identifies pixels relevant to the classification by adding perturbations to the image. Intuitively, a pixel is relevant if the classification of the image changes when this pixel is modified.

In Figure 3.7, we reproduced this idea by measuring the accuracy drop when different regions of the image are perturbed. Zones of the image of size 4×4 are modified by applying the transformation $x \longleftarrow 255 - x$ for pixels x with values in the $[\![0, 255]\!]$ range.

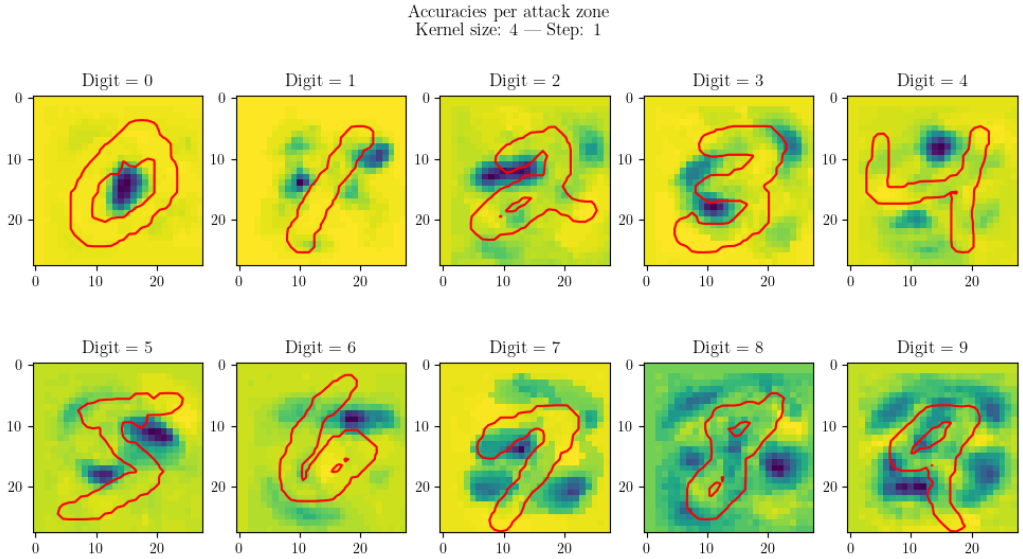

Figure 3.7: Accuracy sensibility to perturbation by spatial region of the image. Darker zones show drops of accuracies when this zone of the image is perturbed.

This results in a visualization of the critical zones of the image, qualitatively different from LRP. While LRP highlights the zones containing the classified object, perturbation methods mostly selects zones outside of the object, where change can lead to confusion with another class: for instance, the zone above the 4 is strongly highlighted since perburbation in this area might trick the model into believing that the digit is a 9 instead.

To evaluate to what extent LRP highlights pixels vulnerable to perturbations, we studied the impact of modifications in the zones of high relevance compared to low relevance zones. To do so, we measured the accuracy of the model while progressively perturbating more and more pixels of the image. In Figure 3.8, we compare the accuracy of the model when pixels are uniformly selected in the image, to the accuracy when pixels are selected with a probability distribution depending on the relevance. If X is the random variable corresponding to the pixel

selected to be modified, then:

$$\mathbb{P}(X = i) = \frac{e^{-R_i}}{\sum_j e^{-R_j}} \tag{3.3.1}$$

In other terms, the probability distribution is the softmax of the opposite of the relevance; the higher the relevance, the lower the probability that the pixel is affected by a perburbation.

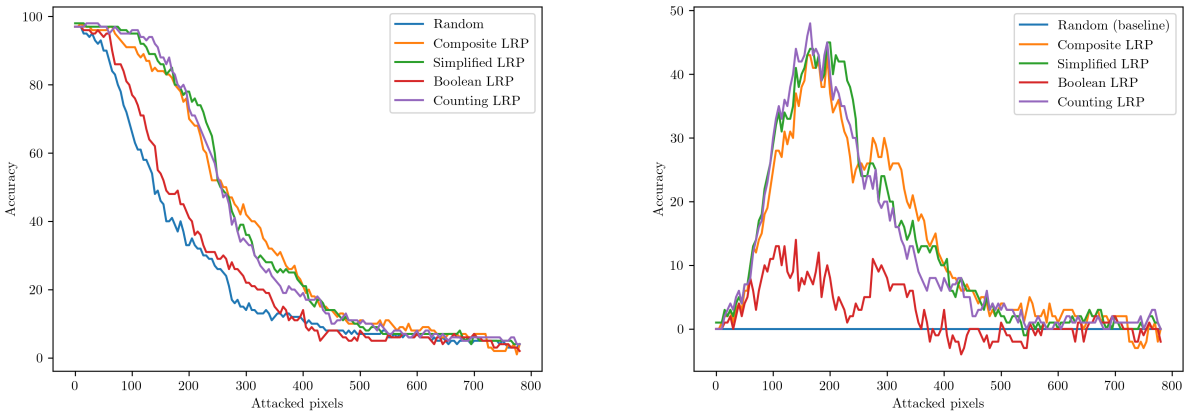

Figure 3.8: Accuracy drop for multiple pixels attacks strategies.

Results of Figure 3.8 show that all LRP distributions outperform the uniform baseline. While LRP does not directly highlight vulnerable pixels, pixels with less relevance are most likely to be resilient to perturbations.

4 Handling Convolutional Neural Networks

If LRP rules are well-suited for simple, feedforward neural networks, most use cases of explanations occur with larger models, namely deep convolutional neural networks (CNNs). We explain how previously introduced rules can be adapted to CNNs, and provide visualization examples for the VGG-16 model.

4.1 Computing relevance for convolutional layers

Convolutional layers are special types of linear layers, in which an output neuron of a layer does not depend on every single neuron from the input. While in a rule of the form

$$R_j^{(l)} = \sum_k d_{j,k}^{(l)} \cdot R_k^{(l+1)}$$

the sum Σ_k relates to every neuron of the output layer, the neuron j might have no influence at all on most of the neurons in this sum. Therefore, we can restrict the sum to only neurons of the output layer which depend on the j-th neuron of the input layer. This can be seen in LRP rules, since the coefficients $d_{j,k}^{(l)}$ are proportional to $w_{j,k}$, which is null if there is no connection between neurons j and k.

If applying directly the formula to convolutional layers is theoretically possible, it is prohibitely expensive to do so in practice. Two methods allow to efficiently compute relevance for convolutional layers. We will first review an efficient gradient-based approach, that requires automatic differentiation; secondly, we will propose a slower method that can be used without depending on the gradient computation.

4.1.1 Gradient expression of LRP

Classical LRP rules can be easily and efficiently implemented. A general rule encapsulating LRP-$0/\varepsilon/\gamma$ can be written as:

$$R_j^{(l)} = \sum_k \frac{a_j \cdot \rho(w_{j,k})}{\varepsilon + \sum_{j'} a_{j'} \cdot \rho(w_{j',k})} \cdot R_k^{(l+1)} \tag{4.1.1}$$

where $a = a^{(l)}$, $w = w^{(l)}$. Four steps, described in [9], allow for an efficient and effortless computation of relevance:

$$\begin{aligned} \forall k : z_k &= \varepsilon + \textstyle\sum_{j'} a_{j'} \cdot \rho(w_{j',k}) \\ \forall k : s_k &= R_k / z_k \\ \forall j : c_j &= \textstyle\sum_k \rho(w_{j',k}) \cdot s_k \\ \forall j : R_j &= a_j \cdot c_j \end{aligned} \tag{4.1.2}$$

These steps provide two majors benefits. Firstly, it allows for the costly divisions by z_k to be performed only once per k, independently of j. Secondly, the third step can be expressed as the computation of a gradient:

$$c_j = \left[\nabla \left(\textstyle\sum_k z_k(a^{(l)}) \cdot s_k\right)\right]_j \tag{4.1.3}$$

where the gradient ∇ is taken with respect to the vector $a^{(l)}$ and where the s_k are treated as constants. This allows for a simple and efficient implementation, since the gradient can be computed using automatic differentiation: the high-level code is not only concise (two lines of PyTorch code), but also effortlessly generalizable to any layer implementing backward pass, and highly optimized using GPU operations.

4.1.2 Implementing LRP without automatic differentiation

While the gradient-based expression is convenient for classical LRP, it does not allow for the overloading of sum and product operations, making it impossible to use with a semiring of our choice. To overcome this limitation, we explicitely derive the dependency between input and output neurons of the convolutional layer, resulting in less flexible code.

Notice that the sum of contributions of one neuron to the output can be expressed as a convolution over the output. That is, in a formula of the form:

$$R_j^{(l)} = \sum_k \frac{z_{j,k}}{\sum_{j'} z_{j',k}} \cdot R_k^{(l+1)}$$

the sum in the denominator, $\sum_{j'} z_{j',k}$, can be expressed as a forward pass of the convolutional layer, that is a convolution over the input; in contrast, the main sum, $\sum_k$, can be expressed as a convolution over the output relevance.

4.1.3 Other CNN layers

Most deep convolutional neural networks contain other special layers. We provide a general description of how these were handled in our experiments on the VGG-16 model.

Average Pooling Such layers can be seen as convolutional layers with fixed weights, in which each weight of the kernel is identical. In practice, most average pooling layers are used with a stride equal to the kernel size K, creating no overlapping. Therefore, the propagation rule for average pooling layers can be simplified as:

$$R^{(l)}[i][j] := \Theta(d_{i,j}) \otimes R^{(l+1)}[i/K][j/K] \tag{4.1.4}$$

where K is the kernel size.

Batch Normalization Layers At test time, normalization layers act as a linear operatioin, and can therefore be fused with the previous linear layer. LRP and semiring-based LRP can then be applied directly to the fused layer.

4.2 Results for the VGG-16 network

We conducted experiments on the VGG-16 network [15], trained on the ImageNet-1K dataset, containing 1 281 167 images from 1 000 classes. Its very deep architecture makes it a challenging model for relevance propagation, since minor changes in the deeper layers can ruin the visualization of the first layers.

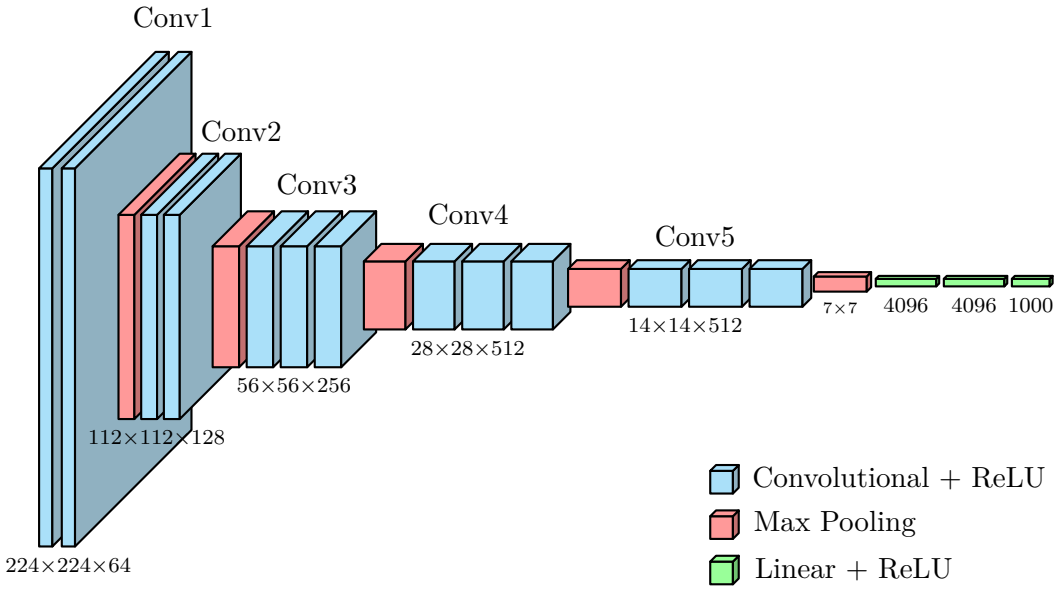

Figure 4.1: Architecture of the VGG-16 network.

A necessary ajustement compared to the simpler fully-connected neural networks is to be able to choose a different annotation function for each layer. Instead of using the same Θ at every layer, a family of functions $(\Theta^{(l)})_l$ is used. In practice, since we used threshold-based annotation functions, this means that each layer will have a different threshold. We discuss the choice of thresholds and the correlation to other parameters of the network in subsection 4.3.

4.2.1 Boolean semiring

$\mathbb{B}$-relevance over VGG-16 is coherent with results on the MNIST dataset: it specifies a main zone of importance in the image, similarly to explainations of tools such as Grad-CAM [12].

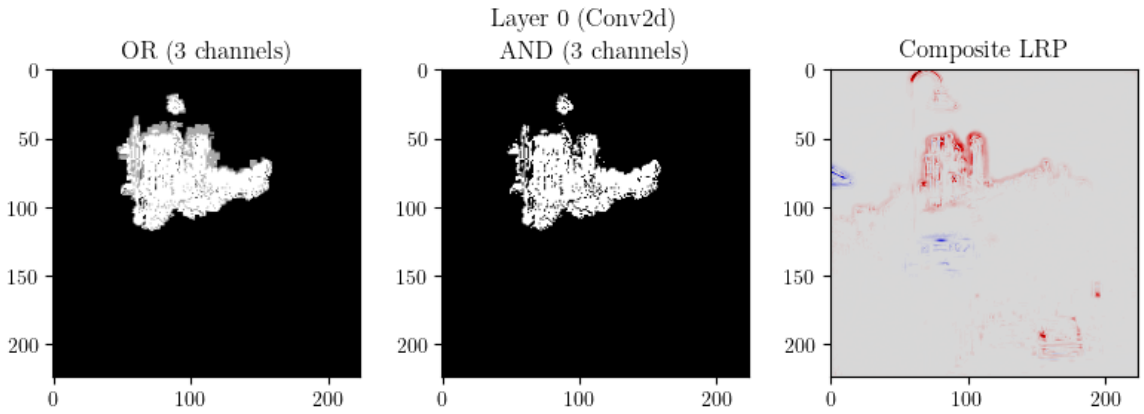

Figure 4.2: $\mathbb{B}$-relevance of the class `castle`. The input layer having three channels (R, G, B), they are aggregated using OR/AND operations.

While classical LRP provides a more precise explanation by focusing on the contours of the relevant object, $\mathbb{B}$-relevance selects all the pixels of the object, including its center.

4.2.2 Counting semiring

Counting semiring produces an explanation qualitatively similar to classical LRP.

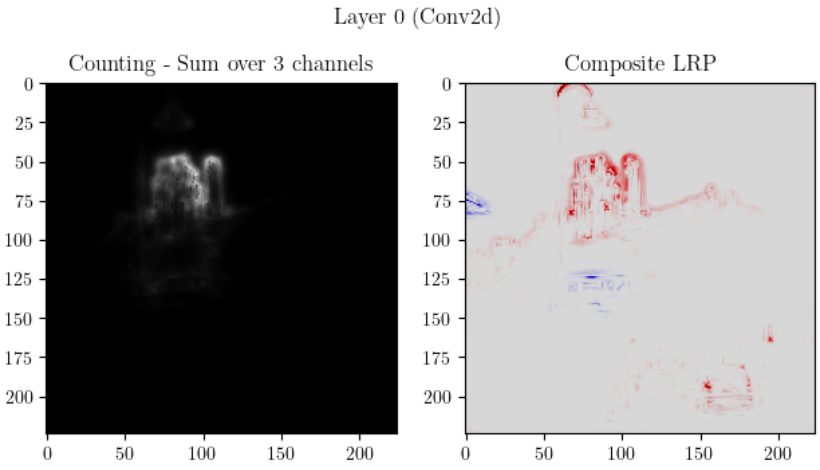

Figure 4.3: $\mathbb{N}$-relevance of the class `castle`. The three channels are aggregated using a sum, like for composite LRP.

4.2.3 Viterbi semiring

4.3 Thresholds choice

Equations such as 2.2.2 and 2.2.4 introduce thresholds which must be ajusted to provide meaningful explanation. Selecting such parameters optimally requires a proper definition of explanation quality, which is a separate research domain [11]. Nevertheless, only some ranges of threshold values guarantee a final result that is humanly interpretable. Threshold values that are too high result in visualizations containing little to no information; relevance is "lost" in the deepest layers and only a few input neurons are marked as relevant. Threshold values that are too low result in over-saturated visualizations; for instance, boolean relevance selects most neurons as relevant, loosing meaningful information in the amount of activated pixels.

We propose threshold values, used in Figure 4.2, that offer a balanced explanation. Modifiying these values by small amounts allow the user to choose between a more precise or more general explanation.

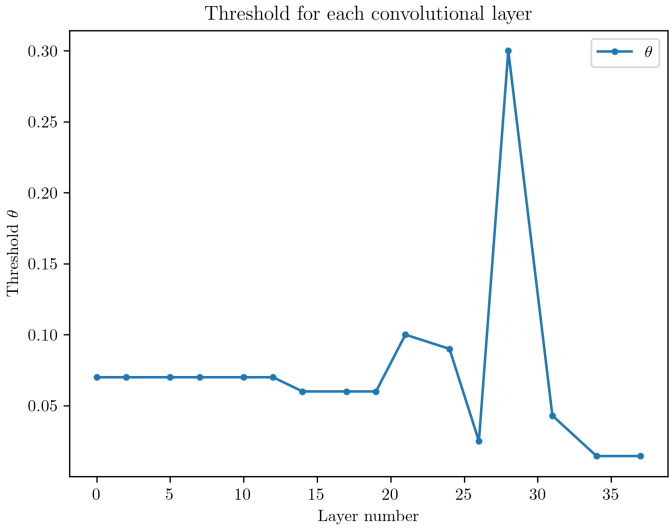

Figure 4.4: Variation of the threshold of convolutional layers used to compute $\mathbb{B}$-relevance

While those parameters were handpicked, they appear to be correlated to parameters of the layers.

4.4 Notes on implementation

5 Conclusion

We introduced Semiring-based Layer-wise Relevance Propagation to extend LRP to a broader range of explanations. Semiring-based LRP benefits from the capabilities of classical LRP, while letting the user choose customized semirings for explanations that better suit the applications of its model.

We implemented Semiring-based LRP for PyTorch models containing Linear, Convolutional, Spatial pooling, and Activation layers; the user can simply implement its own semiring by overloading the addition, multiplication, $\mathbb{0}$ and $\mathbb{1}$ elements and annotation function. Experimentally, boolean and counting semiring provide scalable explanations that qualitatively differ from classical LRP.

Finally, we explored three examples of applications of semiring-based LRP: mask computation, network pruning, and vulnerable pixels detection. We believe that semiring-based LRP has the potential to provide more than an explanation for a single execution, and those applications tend to show that combining LRP results over multiple executions produces meaningful metrics for the overall relevance of neurons in the network.

Acknowldegements

I would like to thank Silviu Maniu for his guidance and support throughout this internship. I thank Pierre Senellart for his feedback and advices. Thanks to the LIG and specifically to the

members of the SLIDE team. In particular, thanks to Alan Gany, Nassim Bouarour for their help, and thanks to Paul Landrier for his suggestions.

References

[1] Sebastian Bach et al. 'On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation'. In: *PLOS ONE* (2015), pp. 1–46. URL: https://doi.org/10.1371/journal.pone.0130140.

[2] Hongrong Cheng, Miao Zhang and Javen Qinfeng Shi. 'A Survey on Deep Neural Network Pruning-Taxonomy, Comparison, Analysis, and Recommendations'. In: (2023). URL: https://arxiv.org/abs/2308.06767.

[3] Ruth C Fong and Andrea Vedaldi. 'Interpretable explanations of black boxes by meaningful perturbation'. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 3429–3437. URL: https://arxiv.org/abs/1704.03296.

[4] Robert Geirhos et al. 'Shortcut learning in deep neural networks'. In: *Nature Machine Intelligence* 2 (2020), pp. 665–673. URL: https://arxiv.org/abs/2004.07780.

[5] Todd J Green, Grigoris Karvounarakis and Val Tannen. 'Provenance semirings'. In: *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 2007, pp. 31–40.

[6] Yann LeCun. *The MNIST database of handwritten digits*. 1998. URL: http://yann.lecun.com/exdb/mnist/.

[7] Aravindh Mahendran and Andrea Vedaldi. 'Understanding deep image representations by inverting them'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 5188–5196. URL: https://arxiv.org/abs/1412.0035.

[8] Pavlo Molchanov et al. 'Pruning Convolutional Neural Networks for Resource Efficient Inference'. In: (2017). URL: https://arxiv.org/abs/1611.06440.

[9] Grégoire Montavon et al. 'Layer-Wise Relevance Propagation: An Overview'. In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer International Publishing, 2019, pp. 193–209. URL: https://doi.org/10.1007/978-3-030-28954-6_10.

[10] Yann Ramusat, Silviu Maniu and Pierre Senellart. 'Provenance-Based Algorithms for Rich Queries over Graph Databases'. In: *EDBT 2021 - 24th International Conference on Extending Database Technology*. 2021. URL: https://inria.hal.science/hal-03140067.

[11] Wojciech Samek et al. 'Evaluating the visualization of what a deep neural network has learned'. In: *IEEE transactions on neural networks and learning systems* 28.11 (2016), pp. 2660–2673. URL: https://arxiv.org/abs/1509.06321.

[12] Ramprasaath R Selvaraju et al. 'Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization'. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626. URL: https://arxiv.org/pdf/1610.02391.

[13] Pierre Senellart. 'Provenance and probabilities in relational databases'. In: *ACM SIGMOD Record* 46.4 (2018), pp. 5–15. URL: https://inria.hal.science/hal-01672566.

[14] Pierre Senellart et al. 'ProvSQL: Provenance and probability management in postgresql'. In: *Proceedings of the VLDB Endowment (PVLDB)* (2018). URL: https://inria.hal.science/hal-01851538.

[15] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. URL: https://arxiv.org/abs/1409.1556.

[16] Matthew D Zeiler and Rob Fergus. 'Visualizing and understanding convolutional networks'. In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer. 2014, pp. 818–833. URL: https://arxiv.org/abs/1311.2901.