Ilies Benhabbour

Promo 2022 – POST MASTER'S DEGREE IN SECURITY IN COMPUTER SYSTEMS AND COMMUNICATIONS

iABG

Einsteinstrasse 20, 85521 Ottobrunn, Germany

15.09.2021 / 15.03.2022

Explanability for Neural Networks

EURECOM advisor (Pr. Melek Önen)

Company Advisor (Dr. Philip Trautmann, Dr. Zardosht Hodaie)

Confidential thesis report
YES ☐ *NO* ☒

EURECOM

EURECOM
Sophia Antipolis

DECLARATION POUR LE RAPPORT DE STAGE
DECLARATION FOR THE MASTER'S THESIS

Je garantis que le rapport est mon travail original et que je n'ai pas reçu d'aide extérieure. Seules les sources citées ont été utilisées dans ce projet. Les parties qui sont des citations directes ou des paraphrases sont identifiées comme telles.

I warrant, that the thesis is my original work and that I have not received outside assistance. Only the sources cited have been used in this report. Parts that are direct quotes or paraphrases are identified as such.

À Biot, *in Biot*
Date : 01.03.2022

Nom Prénom : Benhabbour Ilies
Name First Name

Signature :

Abstract

Français : Les réseaux de neurones sont utilisés aujourd'hui dans de nombreux domaines tels que la classification ou la détection d'objets, la reconnaissance vocale, le traitement du langage naturel, etc. Bien que cette technique soit de plus en plus célèbre en raison de sa précision croissante et des différentes bibliothèques existantes facilitant son implémentation comme Tensorflow ou Pytorch pour ne nommer que les deux plus célèbres; les modèles créés sont souvent considérés comme des boîtes noires. En effet, après la phase d'apprentissage, il est extrêmement difficile de comprendre comment un modèle se comporte compte tenu d'une entrée en raison du très grand nombre de paramètres et de la complexité des couches qui ne sont pas faciles à comprendre pour les non experts. Récemment, les chercheurs se sont de plus en plus intéressés sur le fait de trouver des moyens de comprendre leurs mécanismes sous-jacents. En conséquence, l'Explainable Artificial Intelligence (XAI) est née de cette motivation et tente de fournir des outils qui permettront à ces mêmes chercheurs ainsi qu'aux data engineers d'améliorer leurs modèles pour diverses raisons. L'objectif de mon stage est lié à l'une de ces raisons: Comment pouvons-nous expliquer les réseaux de neurones convolutifs (CNN) afin d'améliorer notre compréhension de leurs prédictions à des fins de sécurité. Les objectifs qui m'ont été donnés par IABG, l'entreprise avec laquelle j'ai travaillé, ont été clairement définis comme suit: comprendre et trouver différentes façons d'atteindre l'explainability de la vision par ordinateur en lisant des documents de recherche de pointe dans un premier temps. Ensuite, en se concentrant sur le développement d'un logiciel dans lequel des utilisateurs peuvent essayer les différentes approches qui ont été découvertes afin d'expliquer un réseau de neurones. Il est important de souligner que la plupart des méthodes qui ont été étudiées et décrites plus loin dans ce rapport sont utilisées pour des CNN mais peuvent également être appliquées à différentes architectures. D'un point de vue personnel, ce stage vise à augmenter mes compétences dans un sujet majeur de nos jours qui est l'apprentissage automatique. En effet, j'aimerais travailler à l'avenir avec des réseaux de neurones pour la sécurité informatique et de la communication. Par conséquent, l'explainability est une excellente première étape car elle m'aide à comprendre en profondeur les réseaux de neurones : comment ils fonctionnent et comment ils peuvent être trompés par les attaquants.

English: Deep Neural Networks are used today in many fields such as object classification or detection, speech recognition, natural language processing and so on. Although neural networks are more and more famous because of their increasing accuracy and the existing frameworks that facilitate their implementation like Tensorflow or Pytorch to only name the two most famous; the models which are built are often considered as blackboxes. Indeed, after the learning process, it is extremely difficult to understand how a model behaves given an input because of the number of parameters, and the complexity of the layers which are not easy to comprehend for non neural network experts. Recently, people are more and more interested in finding ways to understand the underlying mechanisms of neural networks. As a result, Explainable Artificial Intelligence (XAI) was born from this incentive and tries to provide tools that will allow researchers and data engineers to improve their models for various reasons. The goal of my internship is related to one of these reasons: How can we explain decisions of convolutional neural networks (CNN) in order to improve our understanding of their predictions for security purposes. The objectives that I was given by IABG, the company that I worked with, were clearly defined as the following: understand and find different ways to achieve explainability in computer vision by reading state-of-the-art research papers at first. Then, focus on developping a piece of software in which users can try the different approaches that were discovered in order to explain decisions of a neural network. It is important to highlight that most of the methods which were studied and described later in this report are used for CNNs but can be also applied to different architectures. From a personal perspective this internship aims at increasing my skills in a major topic nowadays which is machine learning. Indeed, I would like to work in the future with neural networks for IT and communication security. Hence, explainability is a great first step as it helps me understand neural networks in depth: how they work and how they can be fooled by attackers.

Contents

1 Introduction

1.1 General presentation

In my previous semester project, I have seen that neural networks are sensitive to adversarial attacks. DeepBillBoard [ZLZ+20] creates a perturbation that misleads a car by increasing its steering angle to the right or to the left depending on the perturbation. It made IABG want to learn more in depth how those attacks work, how they behave inside neural networks and how they can be prevented or detected. Indeed their project SafeAI aims at providing more secure AI solutions to end users. This problematic is not just bound to the automobile industry but to every field in which machine learning is used. Explainability also often referred as intepretability gives researchers insights into operations that take place in neural networks, and stands out as the solution to these questions. During my internship, I tried my best to support IABG to understand what explainability is both at a high and low level, what are the different approaches and their limits. Finally I did some implementations of the existing methods to illustrate the explanation methods.

1.2 Company presentation

IABG is a German company headquartered in Ottobrunn, Bavaria. Founded in 1961 on the initiative of the Federal Republic of Germany as an analysis and test center facility for aviation industry and the Ministry of Defense. Privatized in 1993, the company now has in its ranks about 1000 employees and is currently headed by Prof. Dr.-Ing. Rudolf F. Schwarz. Despite ever more competitive sectors in the technical fields, IABG stands from its 215 million euros income in 2019, as a pillar in the field of German engineering expertise. Among other things, the company has worked in the aeronautical field on projects in collaboration with AirBus, in particular on the world-renowned A380 model. Its skills in the space field are not to be outdone, in fact, it is one of the architects of the European rocket Ariane 5. In the automotive field, IABG offers test and development services for car suppliers and manufacturers. It is also one of the main security companies in Germany. For example, it provides prevention and protection solutions against the growing threats of cyber-attacks. IABG is also positioned as a key player in the field of science. By placing research as a predominant point in its strategy via its "Innovation Center" (IZ) or through its relations with universities and research centers such as the Bundeswehr University, TUM, Eurecom or Fraunhofer. It offers its customers the excellence of European researchers and engineers as well as the cutting-edge technologies associated with them.

1.3 Department and project goals

The internship took place in the department for predictive modeling and decision support (IZ60) under Dr. Martin Glas. This department is constituted of more than twenty researchers and engineers. Its aim is to provide artificial intelligence research and solutions not only to the rest of the company but also to customers. Several cutting-edge projects are currently being done dealing with various field of application like natural language processing or cybersecurity. During my internship, I worked with two supervi-

sors: Dr. Philip Trautmann and Dr. Zardosht Hodaie. The former is a mathematican and was helping me on the theoretical part of the internship. Indeed, neural networks involves important mathematical background, thus it was really helpful to have him assisting my research during the beginning of the internship. Dr. Hodaie as a computer scientist focused on the practical part. That means, that he helped me developing the software framework by giving useful advice. The goal of my internship was initially to do some research on both explainability and adversarial attacks in order to develop tools and knowledge on how IABG can build more robust models for the automobile industry. However, after having seen how broad those two subjects were, we decided to focus solely on explainability as it may be more relevant for the moment. Therefore, my role as a trainee, was to do high and low level state-of-the-art research on this topic. This involved to create a taxonomy on which the department can rely and try to implement some generic tools that can afterwards be used. As a matter of fact, XAI can be used as a powerful instrument to address many issues such as debugging and improving, detect anomalies like adversarial attacks or simply giving an explanation of what really happened for example in a car accident involving an autonomous vehicle. I will describe afterwards in more details the steps I have to follow in order to succeed but it would also be interesting to adapt my research to a use case concerning self-driving cars in further works. From a personal point of view, this internship represented a great opportunity as it allowed me to increase my skills in a domain that is greatly attractive to me. I also want to add that in my previous semester project, I did not focus on the mathematical side and how neural networks actually behave. That represents one of the key point I wanted to improve on this time. Moreover, explainability constitutes the fact to understand low level information and transform this information into comprehensive facts. Therefore, succeeding in this internship with this kind of subject was really important to me because it would mean that I succeeded in understanding how neural networks work in depth.

1.4 Milestones

In order to work efficiently in a project, my supervisors highlighted the importance to have a clear temporal guideline to follow. As I mentioned before, my internship was separated into two main phases which can be themselves decomposed into many subphases. The first one was to review interesting papers that would help me understand what explainability is, how it can be provided and why it is so important. Surveys are really good papers in this case as they provide up-to-date information of what has been done across years with a good understanding of what the challenges are. After having an accurate idea of the methods that were used, their pros and cons, I could make a selection of which papers were relevant to me and read them in order to understand the technical aspects behind them. The second part of my internship consisted in developing a tool which can be important to the company. We discussed a lot the nature of the tool. Two main ideas were possible: either we implement a library that can be then used by a developer or we create a software tool in which a user can play around with some methods. As my supervisors and me were really interested in the idea of developing an interactive explainability toolbox, we decided to implement a web application which the employees of the department can use. For example, given a model and an input, that software tool would provide indications of what is going on inside the

model (activation maps at a specified layer, class visualisation of the predicted class...) and gives also some explanations such as saliency maps, a method presented later in the report. Some tools that were previously presented gave me a good idea of what can be achieved and how I can implement mine [CdMP21],[YCN+15]. An important date to schedule my milestones is the 10^{th} of November as I decided to give a presentation to the department on Explainability. Following this constraint, it gave us the following schedule.

- From 15.09.2021 to 15.10.2021 (5 weeks): Understanding the high-level view of Explainability and the most famous methods applied to CNNs.
- From 18.10.2021 to 03.12.2021 (7 weeks): Dedicating time to a personal taxonomy to present to the department. Focusing on different methods that could be implemented afterwards.
- 10.11.2021: Presentation to the department.
- From 06.12.2021 to 18.02.2022 (11 weeks): Developing a software tool.
- From 21.02.2022 to 15.03.2022 (3 weeks): Finalizing code, documentation and reports.
- 02.03.2022: Hands-on session on how to implement explainable methods.

As we can see in the milestones, we decided to give a hands-on session as it was really important for us to provide the other members of the department the opportunity to work on explainability themselves. I also purposely organized it as late as possible because I wanted to give them the possibility to see and understand the toolbox we created during the internship. Developing the software tool was a bit longer than expected as it took me actually 13 weeks. However the milestones were correctly defined and respected overall. Apart from the two weekly appointments with my supervisors, I was also invited by Dr. Lukas Höhndorf to attend every two weeks a meeting regarding the standardization of AI. During these gatherings, the participants take turns presenting ISO standards related to Artificial Intelligence. I was responsible for the explainability norm and presented it on the 11^{th} of March 2022.

2 Research

2.1 Prerequisites

In my previous semester project, I did not really focus on how neural networks work. In this internship, the subject involves to go in depth on how each neuron and layer react. In Figure 1, we see a multilayer perceptron where each input is connected to the output through a hidden layer. The inputs are forwarded and are weighted across the network in order to make the prediction.

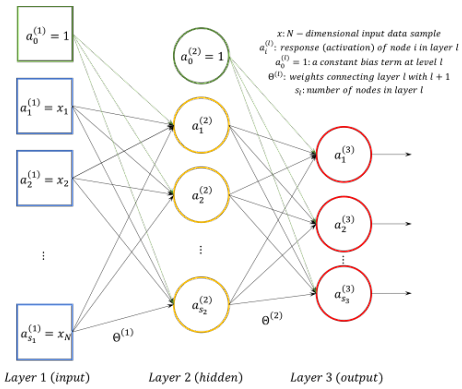


Figure 1: Neural network with one hidden layer in yellow, the bias is in green, the inputs in blue and finally the outputs in red.

I worked on convolutional neural networks (CNN) which are used in computer vision. The characteristic of CNNs is that they use convolutional layers that capture features. At the lower layers those features are simple like edges or colors, however going deeper will lead to more complex features. We can see in Figure 2 the global architecture of a CNN, the MaxPooling layers are used to decrease the dimension of the output of the convolutional layers, the fully connected layers are placed after the last convolutional layer and are forwarded to the output using a softmax final activation function to calculate the probability of each class. In the same figure, the activation function which is used for dense and convolutional layers is the Rectified Linear Unit (ReLU), this function is equal to $f(x) = max(0, x)$. Activation functions are responsible for the non linearity of the neural networks. ReLU is often used because of its performance compared to sigmoid or tanh for example. As mentioned before, I was currently focusing on CNNs and more specifically the classification task. The last important thing to know before trying to understand the operations that occur in neural networks are the concept of backpropagation and loss functions. Neural networks are great to approximate complex functions. Nevertheless, approximating those functions needs a fundamental process which is called learning. This operation is done in supervised learning by giving a set of (x, y) values where x is for example an image and y its label. The neural network will try to minimize the difference between the function approximated for prediction and the real value. This difference is characterized by what is called a loss function (for example cross-entropy for VGG16 or mean squared error for Dave in DeepBillboard). The operation that tries to minimize the loss function is handled by backpropagation. During this operation the gradient of the loss function is calculated. Obtaining the gradient and updating it is pretty simple using the chain rule.

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial b}\frac{\partial b}{\partial c}\cdots\frac{\partial y}{\partial z}\frac{\partial z}{\partial w_1}$$

Here L represents the loss function, the letters stand for the gradient calculated at different layers and w_1 is the weight to be updated.

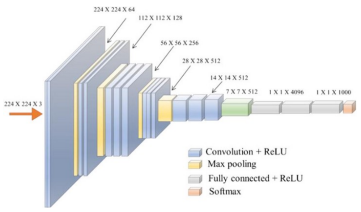


Figure 2: Famous CNN proposed by K. Simonyan and A. Zisserman [SZ15], composed of 13 convolutional layers.

2.2 Explainability surveys

Explainability is a highly discussed topic yet researchers do not agree on a common definition. Some people tend to describe explainability as the sum of interpretability and completeness which respectively express the fact to be understandable for a domain specific task and the number of information given in order to explain the behaviour of the model. On the other hand some say that explainability, interpretability, understandability and transparency are the same concept. After having read some surveys, I sticked to the simple idea that those terms are equivalent but that explanations can still be different. These differences are represented in Figure 3. Completeness can hence be achieved for a model by applying all those methods.

Dimension 1 — Passive vs. Active Approaches	
Passive	Post hoc explain trained neural networks
Active	Actively change the network architecture or training process for better interpretability
Dimension 2 — Type of Explanations (in the order of increasing explanatory power)	
To explain a prediction/class by	
Examples	Provide example(s) which may be considered similar or as prototype(s)
Attribution	Assign credit (or blame) to the input features (e.g. feature importance, saliency masks)
Hidden semantics	Make sense of certain hidden neurons/layers
Rules	Extract logic rules (e.g. decision trees, rule sets and other rule formats)
Dimension 3 — Local vs. Global Interpretability (in terms of the input space)	
Local	Explain network's *predictions on individual samples* (e.g. a saliency mask for an input image)
Semi-local	In between, for example, explain a group of similar inputs together
Global	Explain the network *as a whole* (e.g. a set of rules/a decision tree)

Figure 3: Taxonomy of methods proposed by [ZTLT21]

I will describe Figure 3, that in my opinion gives a good idea of the methods available to provide explainability. The survey I took that image from suggests that the methods can be seen in a 3D plan. In the first dimension, we have the model inherent interpretability degree. If the model is built in order to provide interpretability easily, then it means that we are in the case of an ad-hoc explanation. The opposite is called post-hoc interpretability in which you have a highly non understandable model that can be considered as a blackbox. In this case, researchers try to use the properties of neural networks to find ways to present factual information from those models. The problem for the active approach (ad-hoc) is that there is a tradeoff between accuracy and

explainability, if the model is more understandable then the accuracy drops. Thus, it encourages research in the case of blackbox models. The second dimension describes the type of explanation. For example, saliency map methods that will be presented further, attribute different importance to the inputs. In the case of CNNs, pixels are attributed a score, we therefore call it an attribution method. Hidden semantic techniques try to extract layer and neuron information. We can suppose that the most highly activated neuron might be responsible for the face detection if the classifier predicts a human face. Rule explanations give a set of if and $then$ conditions like a decision tree. If the model detects the presence of the sea then we can assume that it is not possible for the model to classify the picture as a computer. Example methods can also provide some ideas of what the model is looking for by giving similar inputs. Finally, the third dimension represents the globality of the explanation, local means that for a given x we explain the output $f(x)$ which is the predicted label in a classification task. Global on the other hand gives an idea on the whole model f not only for one input. The in-between is represented by the semi-local or semi-global explanations in which is described the behaviour of a set of inputs.

2.3 Saliency maps

In the previous paragraph, I described the difference between ad-hoc and post-hoc explainability as well as local and global explanations. I will now present one of the most well-known methods to give a local post-hoc explanation. This technique called saliency maps is used in CNNs to tell the human which pixels are important for the neural network to give its prediction. They are mainly gradient related and provide a good idea of what the model is actually looking at. They were first introduced by [SVZ14]. In the case of a classification task, after having given an image to a neural network, we compute the gradient with respect to the pixels and obtain a $color \times width \times height$ map that can afterwards give a 2 dimension heatmap using a normalization process. Figure 4, shows the saliency maps that let us understand what the neural network considers as important to increase or decrease its prediction. This provides a great tool to understand how a neural network gives its prediction. For example here we can see that in order to detect if it is actually a boat, the model need to have the presence of the sea. We can thus wonder what would be the prediction if the boat was not on the sea.

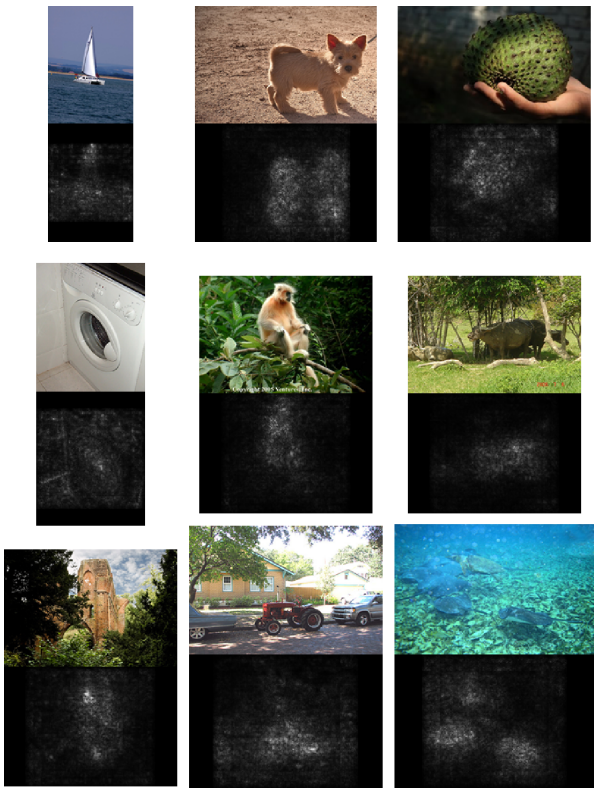


Figure 4: Vanilla saliency maps proposed by [SVZ14]

Although this first method to calculate saliency maps provides interesting results, we can discern a bad trend when it comes to this vanilla gradient saliency map method. The problem is based on the noise. Indeed this noise will give a scattered heatmap which is not really accurate. As a result a lot of research mostly based on gradient saliency maps followed. Here is a non exhaustive list of methods that I will describe and that I was able to study.

- Integrated gradient[STY17]
- Smooth gradient[STK+17]
- Deconvolutional network[ZF14]
- Saliency map by occlusion [ZF14]

Integrated gradient is a method based on 2 axioms that the authors consider necessary in order to provide a good explanation, First they consider that if a feature changes the output then the gradient must change accordingly. For example they back up their point using a one layer function $f(x) = 1 - ReLU(1-x)$. In that case, it holds $f(0) = 0$, $f(2) = 1$ however the gradient are both 0. This represents a problem and they try to decrease its impact. The second axiom is that similar networks need to be invariant in their explanations if they are equivalent. To do so they approximate the gradient using a baseline and integrate the gradient accordingly using the Riemann approximation of the following integral:

$$IntegratedGrads_i(x) = (x_i - x_i^{'}) \times \int_{\alpha=0}^{1} \frac{\partial F(x^{'} + \alpha \times (x - x^{'}))}{\partial x_i} d\alpha$$

In Smooth gradient, the authors attempt to decrease the noise in the image by adding gaussian noise to a sample of copies of the original image. They compute the vanilla saliency maps for all the noisy images and compute the average for all the images. This results in smoother saliency maps that focus more on the real predictive pixels. This method is great because it can be combined with other methods such as Integrated gradient and give sharper visualisations. However, it needs to find the correct parameters σ^2 and n which are respectively the variance of the noise added and the number of samples to average.

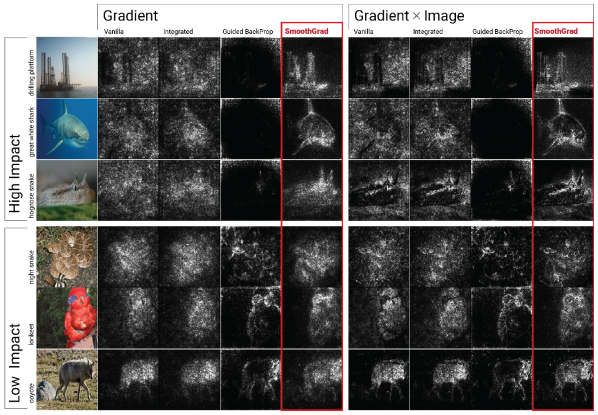


Figure 5: Different saliency maps methods

Figure 5, represents the different methods to find the heatmaps that are important to the prediction. The last two methods I will be discussing were proposed in [ZF14]. The first method is easy to understand and intuitively straight forward. For each pixel of the image you add a gray box (to be as neutral as possible) and see the prediction score. This gray box area must be superior to 1 pixel otherwise the result will not be as explicit as we need it to be. Using this procedure we can draw a picture of the pixel scores heatmap, if the predicted score decreases a lot then it means that the feature was important to the output. Though this technique is simple to understand and gives a good trend of which pixels are important, it has two important drawbacks. We need to compute a prediction for all the pixel. So for a model like VGG16 it means 224^2 predictions which is slow. The second disadvantage is that we tamper with the input by modifying the original image and thus this can lead to wrong indications in some cases. The last method I will be talking about are deconvolutional networks. These methods are great to identify what a particular activation map at a specified layer is looking at. To do so, the authors give the following procedure. First we need to select the feature map we want to study (usually a highly activated one). Then, the algorithm needs to zero out all the other activation maps on the same convolutional layer that are not the one we want to study. Using deconvolutional layers which are transpose convolutional layers we go back to the input. Of course, information is lost during downsampling in the forward phase and as a result the authors propose to save the feature position which was extracted during downsampling to be able to recover the information during upsampling. The process is explained in Figure 6 and results can be seen for the two methods discussed in Figure 7.

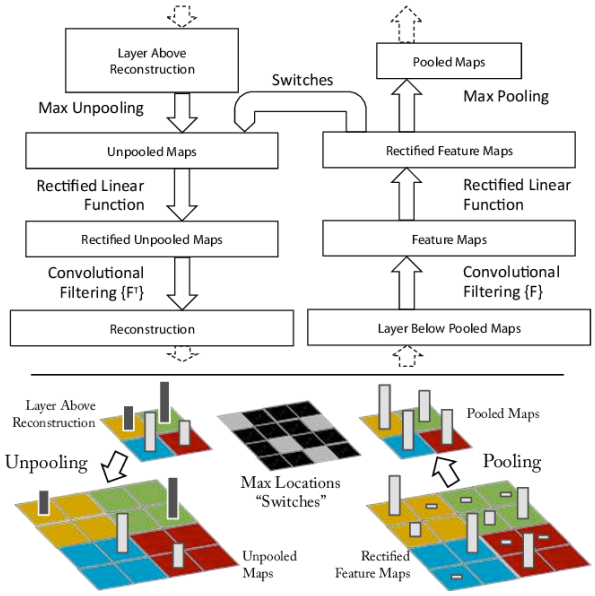


Figure 6: Deconvolution method

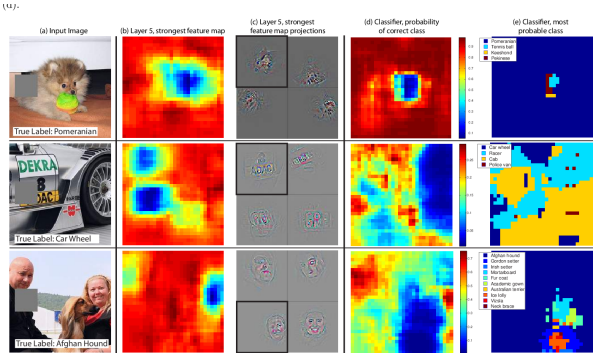


Figure 7: Occlusion and deconvolutional neural network heatmaps

Other methods can be used to find saliency maps like for example [SDBR15] also known as guided backpropagation in which you apply a ReLU function during the backpropagation to have smoother results. The idea behind using a ReLU is that you want to remove the values that would decrease the score in your prediction. Here we want to have the important pixels for the good class and that is why they remove the negative values. CAM [ZKL+16] and its generalization grad-CAM [SDV+19] are also famous since they give a class activation mapping. It means that for a specific output, the algorithm will give you a continuous map of important features that where useful to the prediction. Grad-CAM is especially interesting as it can be applied in many CNN applications and provide plenty of relevant explanations. The idea of grad-CAM is that you compute the forward pass and the backward pass until the convolutional layer which is of interest (usually, the last one). The algorithm then computes the average of each activation map gradient and multiplies the result with each corresponding activation map of the forward pass of the same convolutional layer. Having done that for each position of the maps you can compute the sum of the maps and go through a ReLU function to keep the values which are interesting for the prediction with the same idea of

guided backpropagation. It is also important to note that you can multiply the weights by -1 before the ReLU to obtain the pixels which are decreasing the class score. To finalize, the algorithm, the authors use bilinear interpolation to obtain the same size as the input image.

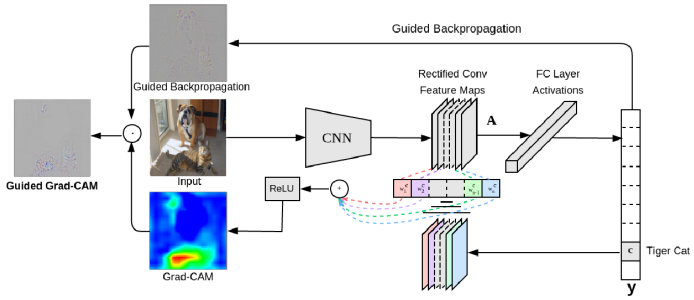


Figure 8: Grad-CAM algorithm for the cat prediction

Figure 8 represents the algorithm for a classification task, here the map obtained gives us which pixels where important to the prediction of a cat. This method can then be coupled with guided backpropagation using a dot product to have a gray map of the explanation, it is called guided grad-CAM. A newer version of grad-CAM called grad-CAM++ was presented in [CSHB18]. This new method tries to address the problem that grad-CAM faces: for same class instances in an image, the largest object will be the only one highlighted by the algorithm. The problem comes from the fact that in the former algorithm, the average of the gradients tends to favor large areas and as a consequence decreases the explanation areas for the same class objects which are smaller in the input space. In Grad-CAM++, the authors generalize the average by creating an α that will weigh back the gradient areas to the same values and it will thus correct the problem of same instances of different size in the input image.

Saliency methods are great when it comes to explain an input, however the explanations can be misleading as it is shown in [AGM+18]. In that paper they randomize the parameters for specific layers and data during the training phase. It appeared that some methods were in fact edge detectors instead of real explanations. In Figure 9 we can see that for guided backpropagation and guided grad-CAM which is a variant of guided backpropagation, the randomization of the weights has no impact to the explanation. In other words, we do not have an explanation of what the neural network has learnt but it is an edge detector that is able to see shapes in the input image. That lead the authors to the conclusion that we definitely should not also think of explanations as blackboxes but we should also try to see if the methods are trustworthy by using:

- Randomization of parameters
- Randomization of data

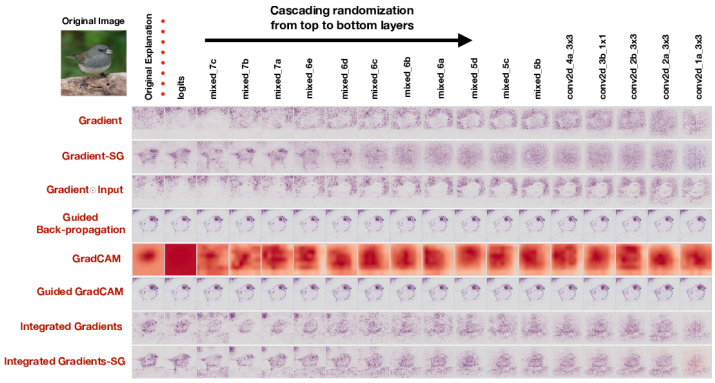


Figure 9: Explanations of different gradient based algorithm for the prediction using cascading randomization from top layers

A good library which I used to implement some techniques above and which is framework agnostic can be found in the following link https://github.com/PAIR-code/saliency.

Layer-wise Relevance Propagation or LRP [BBM+15] is a famous method that is analog to gradient saliency techniques, the objective of this approach is to recover a heatmap of a relevance score back to the input space. Like gradient saliency methods, LRP needs to compute a forward and a backward pass. The difference however, lies in the fact that we do not use the gradient in the backward pass but instead compute iteratively a relevance score using the next layer. In other words, the forward pass is a normal one with a small change that it saves the output of each neuron for all the layers to obtain what is called a neuron activation. This activation represents the result of the previous layer operation using its own activations and weights for that neuron. For example, in a dense layer the activation will be the result of the sum of all the activations multiplied by their associated weight. When the inference is done, we can retrieve the prediction score $f(x)$ which will be equivalent to the relevance score of the last layer for the predicted class. Using it we will try to redistribute that score iteratively until we reach the input. We refer to Figure 10 to illustrate.

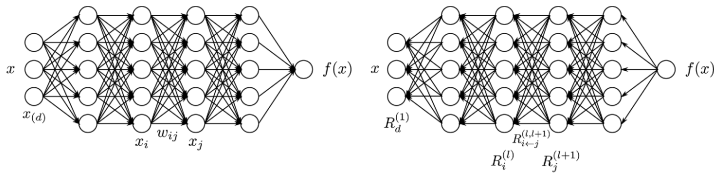


Figure 10: Forward and backward pass of LRP in a feed forward neural network

In the forward pass we have activations x_j^l:

$$x_j^{l+1} = \sum_{i=0}^{n} x_i^l w_{i,j}^l + b^l$$

And in the backward pass we compute the relevance score for each neuron $R_i^{(l)}$ iteratively:

$$R_i^{(l)} = \sum_{j=0}^{m} R_{i\leftarrow j}^{(l,l+1)}$$

Where:

$$R_{i\leftarrow j}^{(l,l+1)} = R_j^{(l+1)} \frac{x_i^l w_{i,j}^l + \frac{b^l}{n}}{\sum_{h=0}^{n} x_h^l w_{h,j}^l + b^l}$$

Here $R_{i\leftarrow j}^{(l,l+1)}$ represents the impact of the relevance score of neuron j at layer $l+1$ to the relevance score of neuron i at layer l. It is also important to notice that the sum of the relevance scores at each layer is conservative. That is to say that the sum of the saliency pixels of the explanation should be equivalent to the prediction score $f(x)$ associated to the targeted output neuron. The equation below represents this rule:

$$f(x) = ... = \sum_{d\in l+1} R_d^{(l+1)} = \sum_{d\in l} R_d^{(l)} = ... = \sum_{d\in 1} R_d^{(1)}$$

In the aforementioned example, I explained only for dense layers but it also means that you can do it for all kind of layers. If we consider flatten layers, this is simple as we just have to reshape the tensor, for pooling layers it seems easy to say that we can just redistribute the values like it is done for the deconvolutional method. However, dealing with convolutional layers is trickier and more complex in terms of computation operations. This will be explained later in this report. In Figure 11 we see a representation given by LRP where red areas show high importance for the prediction *cat* and blue areas show the opposite behavior.

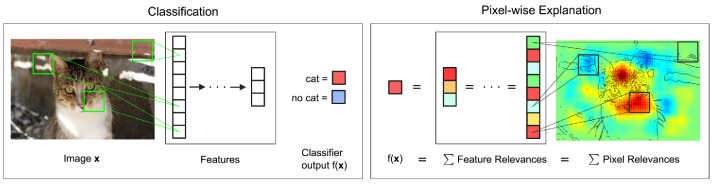


Figure 11: LRP explanation for a binary classifier

It is important to notice that in this report, we only deal with $LRP-0$. Nonetheless, different versions of that algorithm exist and are described in [MBL+19]. It is for example advised to use the $\epsilon-$rule in middle layers and the $\gamma-$rule for lower layers while using $LRP-0$ for upper layers. To briefly describe, the $\epsilon-$rule adds a small term ϵ in the denominator and the $\gamma-$rule adds a factor γ that is multiplied with the weights which are greater than zero to favor the positive contributing neurons.

2.4 Other post-hoc local methods

In this section I will discuss about some other approaches that are used to explain specific outputs. LIME[RSG16] is one of the most well known. It stands for Local Interpretable Model-agnostic Explanation mainly because the authors argue that it can give a local explanation for all kind of models. The idea behind LIME is that we will take an input that we want to explain and give around its neighborhood the behaviour of the model. Figure 12 illustrates the principle behind LIME. Let us say that we have a binary classifier and we want to see how the model is going to behave around a specific location or input x (in our example it is represented by the biggest red cross). To do this, LIME first creates fake samples $z_i \in \mathcal{Z}$ around this value. In

Figure 12 they are showed by either the red crosses or the blue filled circles. Creating these fake values will help us do a located regression around our input. However, the fake data might be near or far from the real input and thus we need to add a weighting function $\pi_x(z) = exp(-D(x,z)^2/\sigma^2)$ where D is a distance function that will be taken into account in our regression task. That is why we see that the nearer the created value is to the input the bigger it is represented. We can then compute the explanation using the loss function $\mathcal{L}$ in the following equation:

$$\mathcal{L}(f, g, \pi_x) = \sum_{z,z' \in \mathcal{Z}} \pi_x(z)(f(z) - g(z'))^2$$

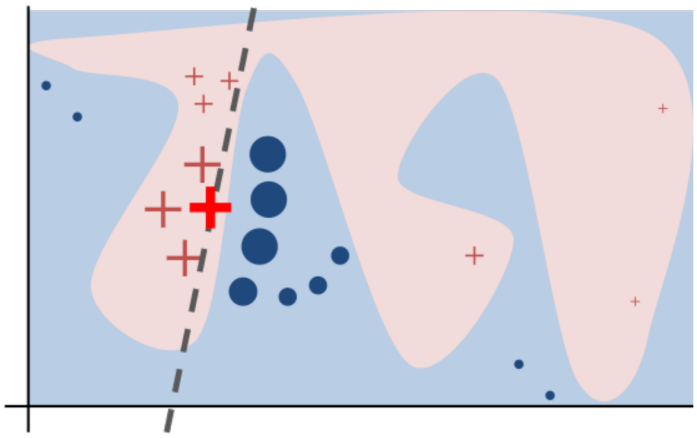


Figure 12: LIME providing the dash line as explanation

This method can be adapted to all kind of data. For example in images it is possible to do it by giving an input image x to the model, creating fake data will be done based on the real image by tearing the original input into superpixels and creating combination of these. We can afterwards compute which combination of superpixels would give us the representation of the targeted class. The idea is similar in SHAP[LL17] that comes from game theory. Indeed, SHAP tries to obtain what are called shapley values. These values correspond to the impact of each parameter in the final prediction score. They can for example be obtained by testing all the combinations and then attribute a score to each parameter.

Critical data routing paths or CDRP [WSZH18] is a different approach that presents a way to explain the importance of each neuron for a specific input during inference. In this method the authors added to the neural network control gates to turn on and off specific neurons. Control gates are simple scalars that will be multiplied by the neuron activation, that means that we can shut down the neuron by setting the control gate as 0 and in the other way we can increase a control gate if it is critical by setting it over 1. CDRP relies on an optimization algorithm. In the beginning the control gates are all set to 1 and using a specific loss function we are able to stabilize the prediction score of the input by shutting and increasing the scalars associated to the control gates. Figure 13 shows what those control gates are and what the notion of critical or negligible node is. Red nodes are not important for the prediction while the green ones are really important if we want to obtain the same score.

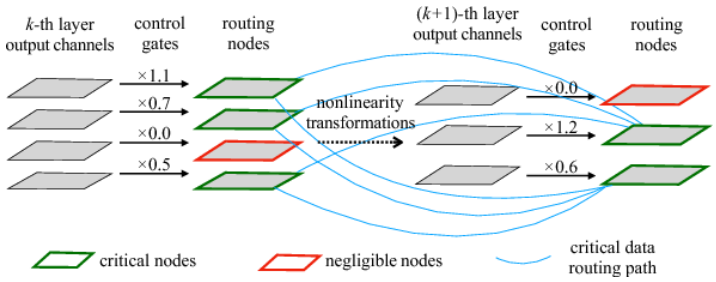


Figure 13: Control gates applied on output channels

To show that this optimization task highlights which neurons are critical for an input, the authors submitted their nodes to an experiment. This experiment's results are shown in Figure 14, the aim is to shut down all the nodes by starting either from the top or by the bottom. If the prediction score plummets faster with the top mode then it means that the identification of the critical neurons was successful. In the figure below we can clearly see that in the top mode, if we remove 5% of the most important nodes the top-1 accuracy drops from 70% to 30% whereas when we start with the least important nodes the top-1 accuracy drops only by 5%. We can conclude that this method identifies correctly which nodes are important for the prediction of a specific input.

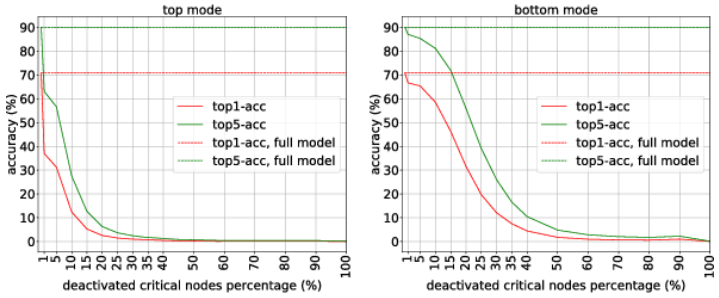


Figure 14: Drop of accuracy when we shut down critical nodes, on the left we start with the most important ones, on the right we start with the least important. Top-1 accuracy means that the prediction is correct while Top-5 accuracy means that the expected class is in the 5 classes that have the highest prediction score.

2.5 Decision trees

Saliency maps are great algorithms to detect bias. However they do not provide enough information about the input. For example, if we retrieve the saliency maps for two close classes, we may obtain similar information and thus we do not understand the decision the neural network did to choose the correct one. In Machine Learning, a decision tree is an interpretable algorithm used for simple tasks. Unlike neural networks they can not handle complex operations accurately because of their binary behavior (if and else conditions). However, these kind of algorithms can be used to give useful feedback on a classifier when combined to a neural network. [WDH+20] uses a Deep Learning algorithm backbone with a decision tree at the end to evaluate the path of inference of a neural network. To create Neural-Backed Decision Trees the authors retrieve a

specific pre-trained classifier first. When the backbone is selected, they have to create a decision tree at the end of the neural network. To do so, they remove the last layer corresponding to the n-classes and create a hierarchy from the weights of the last layer. Let us say we are in the case of VGG16. The last layer contains 1000 output neurons and the previous layer 4096. In this case, we remove the 1000 neurons and retrieve 1000 vectors of dimension 4096 corresponding to the weights for the corresponding neurons. To construct the hierarchy we first put the leaves at the end depending on the neighborhood. That is to say that for each vector, we compute its nearest neighbor and compute the parent node by averaging their values. We do this operation for all the vectors iteratively until the root is reached. Figure 15 illustrates how the hierarchy is built up.

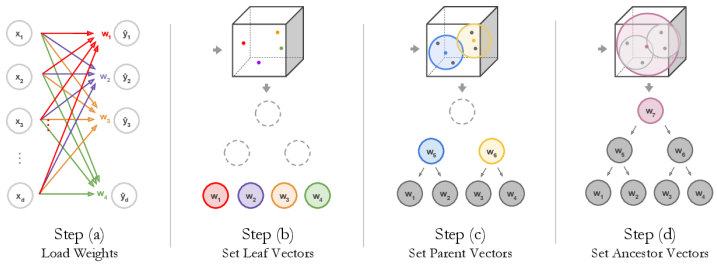


Figure 15: Construction of the decision tree

The paper proposes two inference methods, a soft and a hard one. In both cases we give the input to the neural network and process the data until the last layer. The values obtained are then given to the root of the decision tree (in our previous example we would have a vector of dimension 4096). Then, we can compute the softmax operation between the two nodes to compute the probabilities. The difference between the soft and hard inference modes is that in the hard inference mode we compute the probability at each layer of the decision tree and take the highest one as the correct path to follow until we meet a leaf. The soft inference mode allows low incertitude. That is to say that we do not follow the highest probability at a node to select the prediction but we compute all the intersected probabilities until the beginning to compute the final probability.

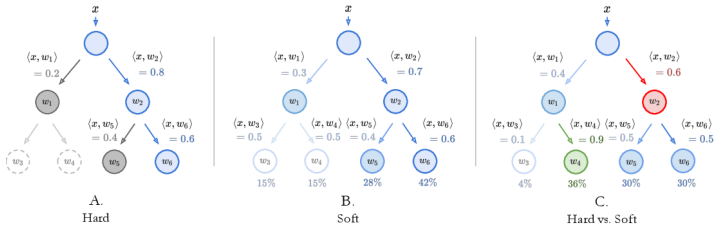


Figure 16: Inference modes in Neural-Backed Decision Trees

In Figure 16, the authors represent how the inference modes are computed. The weights of the Neural-Backed Decision Tree can be fine tuned with a specific loss function depending on the inference mode. Using that method we are able to see different properties of the model: the uncertainty for a specific input, the closest classes based on the hierarchy and we can even label the nodes for each layers using a lexical database

like WordNet. Figure 17 highlights the fact that we are given more information using NBDTs than normal neural networks like for example where the neural network is not sure about its prediction.

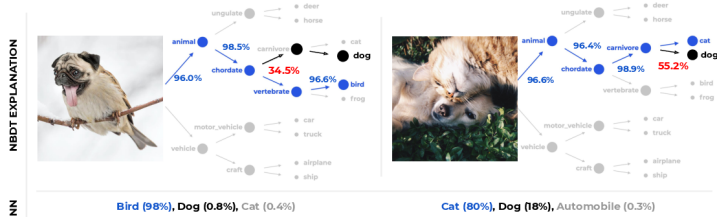


Figure 17: Prediction of two images, on the left a fake one with the body of a bird and face of a dog. The right one represents a cat and a dog lying on the floor

2.6 Adversarial attacks

Adversarial attacks are a well-known problem for neural networks as they can involve safety and security concerns like in DeepBillboard. In short, those attacks add a perturbation which is not visible to a human but will drastically change the prediction of the model. Backpropagation is a key concept when it comes to these attacks. I will describe a bit more what was explained during my last semester project: how they work and the known techniques that I have been able to see during my readings. Previously, I highlighted the importance of backpropagation for training the neural network. Indeed, during this phase, the gradient will be calculated and updated using what is called gradient descent of the loss function. In other words, we update the weights with the negative value of the gradient. This minimizes the loss function for the training dataset. The process is the same for adversarial attacks. However, instead of computing the gradient depending on the weights for the loss function, we compute the gradient for the predicted class in function of the features of the image. Consequently, this gradient will show the pixels which, if modified, will give the most variation to the output. Most of the techniques out-there use this property to create the smallest perturbation that will maximize the variation. Those attacks can be split into two fashions, the untargeted and targeted one. In an untargeted attack, the attacker creates the perturbation by doing a gradient descent with respect to the correct prediction, in other words, minimizing the prediction score for the real output. On another hand, the targeted attack creates the perturbation in order to maximise the prediction function for another output by doing a gradient ascent for this specific class. Four of the most well-known attacks are listed below:

- Fast gradient sign method[GSS15]
- Basic iterative method[KGB17]
- Projected gradient descent[MMS+18]
- The Carlini and Wagner attack[CW17]

Without going into the mathematical details, the fast gradient method is a method where the perturbation is calculated in one iteration. The gradient is multiplied by a

factor ϵ that will give the perturbation influence. The bigger ϵ is the more the adversarial image is different. The basic iterative method, is different in the way that in order to create the perturbation, the algorithm do small steps using FGSM iteratively. PGD and CW attacks are similar but apparently use projections to calculate the perturbation which are constrained in an ϵ-ball. If the step goes beyond that ball they are projected back.

Figure 18: Adversarial attacks that are performed using the Adversarial Robustness Toolbox or ART for the VGG16 model. In the top, the original picture representing a cat correctly predicted by the model. In the bottom, we see its corresponding untargeted attack with FGSM.

In Figure 18, we can see two pictures that look similar for a human but completely different from the neural network point of view. Originally, the model predicts correctly that the image is a tiger cat with a score of 96.2%. Nonetheless, after applying a perturbation using FGSM, the model thinks that the image represents an irish setter with a probability of 13.6%. Nowadays, researchers try to produce perturbations that would be inherent to a model. Those attacks also known as universal adversarial perturbation (UAP)[HMT21], will fool the models for all the inputs that are given. All the attacks above are well known but they imply the fact that the attackers have access to the gradient and thus the neural network. Blackbox attacks where the attack surfaces are smaller also exist. As far as I know, some methods can be used to perform these attacks such as estimating the gradient in order to be in a whitebox attack fashion. Before switching to explainability, I discuss some methods that can improve robustness for adversarial attacks: Feature denoising [XWvdM+19] where during the creation of a the model, denoising blocks are added. Those blocks are trained alongside the neural network using adversarial samples. They will be responsible to reduce the noise of adversarial attacks. The last method is called adversarial data augmentation where

during training you augment your dataset with adversarial samples. These methods are used to increase the robustness of the models and seem to provide good results.

2.7 Dimensionality reduction algorithm

Discovering where a neural network misclassification occurs is a tricky task as these algorithms are complex. This operation can become even more complicated when we discuss about images. Indeed, the input is processed in convolutional layers that produce high dimensional activation maps. The problem with high dimensional data is that we can not visualize them and compare them. Let us say that we want to compare an image prediction across the neural network with the same image in which we added a perturbation using FGSM. We do not have a straight forward tool that would tell us in which convolutional layer the activation maps are drastically different. That is why we need an unsupervised algorithm that is able to reduce the dimension of the data in 1, 2 or 3 dimensions. t-SNE [vdMH08] and more recently UMAP [MH18] are two of the most well-known dimensionality reduction algorithms. They both work on the idea that we need to have a loss function in which we try to penalize the original dimension distances which are high by a greater low dimensional distance. For example the loss function for t-SNE is the following:

$$C = \sum_i \sum_j p_{ij} \, log \left(\frac{p_{ij}}{q_{ij}} \right)$$

Where p_{ij} and q_{ij} represent respectively the pairwise similarities scores between two points i and j in the high and low dimensional space. We will not go into the details of t-SNE and UMAP but by using these algorithms we can clearly separate high-dimensional data into clusters (see Figure 19).

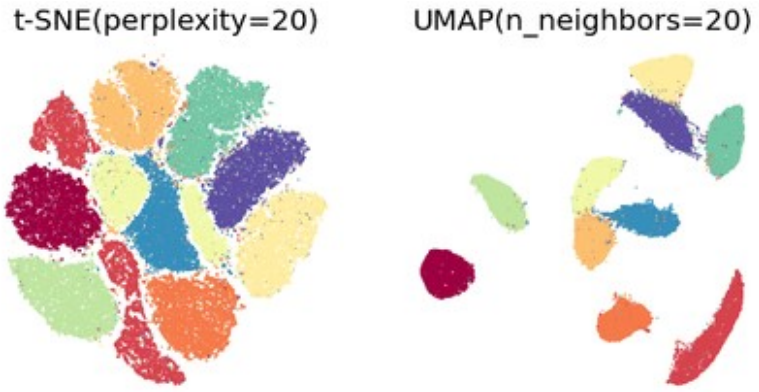


Figure 19: t-SNE and UMAP clustering over the MNIST dataset (28*28=784 dimensions)

Using those algorithms it becomes possible to visualize an approximation of the path the neural network is taking like in Figure 20 from [CdMP21]. Using UMAP parametric form, it can even become possible to do it in real time.

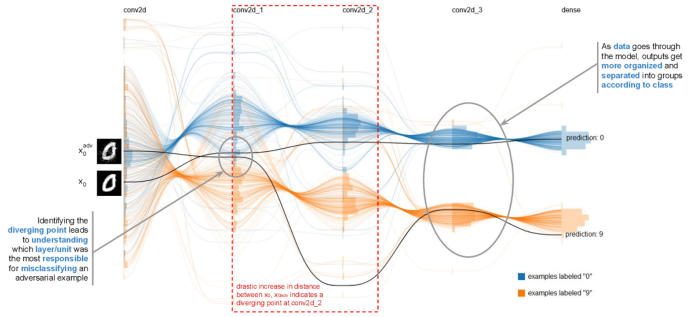


Figure 20: Neural network prediction path for an image and its perturbed version

2.8 Network Dissection

Network dissection [BZK+17] is a powerful global method that tries to give an understanding of a CNN as a whole. The objective of this method is to associate concepts to the convolution filters of the neural network. To do so, they use a dataset called Broden. This dataset contains segmented annotated ground truths for specific categories: object, part, scene, texture, color and material which are themselves divided into many different classes. The algorithm works as follows:

- Identify a convolutional layer of interest
- Input the Broden dataset to the pretrained neural network.
- Compute the activation maps $A_k(\boldsymbol{x})$ of the convolutional layer for all the input images
- Compute the distribution of the individual unit activations a_k for each convolution filter
- Compute the top quantile threshold T_k over every spacial location such that $P(a_k > T_k) = 0.005$
- Upsample the activation maps $A_k(\boldsymbol{x})$ for each input to $S_k(\boldsymbol{x})$ using bilinear interpolation
- Threshold $S_k(\boldsymbol{x})$ into a binary mask $M_k(\boldsymbol{x})$ where $M_k(\boldsymbol{x}) = 1$ if $S_k(\boldsymbol{x}) > T_k$, otherwise $M_k(\boldsymbol{x}) = 0$
- Compute the intersection over union score $IoU_{k,c}$ for a concept c and a convolution filter k:
$$IoU_{k,c} = \frac{\sum |M_k(\boldsymbol{x}) \bigcap L_c(\boldsymbol{x})|}{\sum |M_k(\boldsymbol{x}) \bigcup L_c(\boldsymbol{x})|}$$
- For each convolution filter k take the concept c that has the highest intersection over union score and if $IoU_{k,c} > 0.04$ then we can say that the convolution filter k is responsible for concept c

This method allows us to compare neural network architectures and the concepts they learn depending on the dataset. For example, the authors showed that residual neural networks have more concepts learnt than Alexnet, which makes sense since they are more accurate. To prove that this method is reliable the authors counted the numbers of detectors (concept bound to a convolution filter) during the training phase. Figure 21 highlights the fact that during the training phase, concepts are learnt by neural network.

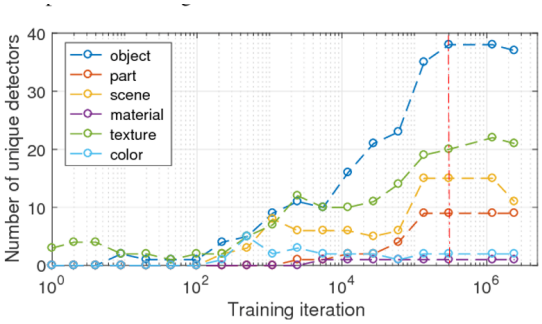


Figure 21: Number of unique detectors depending on the number of training iterations

2.9 Class visualisation

Another global method is class visualisation, that was proposed in [SVZ14]. The idea at a high level is to produce an input that would maximise the score of a specific output neuron. The method behind it is the same as a targeted adversarial attacks. First we need to create gaussian noise which is zero-centered (because in the paper the model was trained on zero-centered data to stabilize the gradient). Then compute a forward propagation. At the end of the forward propagation, we remove the activation function which in that case is a softmax function that gives a probability. The reason for that is to remove the impact of the other values. Indeed, in order to maximise the score, the gradient with a softmax activation function would also try to decrease the score of the other classes. Then we compute the gradient of the output neuron with respect to the input. Finally, we add to the noise the value of that gradient (multiplied by a learning rate). This method is done iteratively to obtain images like Figure 22. This process needs to use regularisation terms in order to avoid overfitting. In the mentioned paper for example, the authors use L2 regularisation with respect to the input pixels and not the weights to decrease high intensity values.

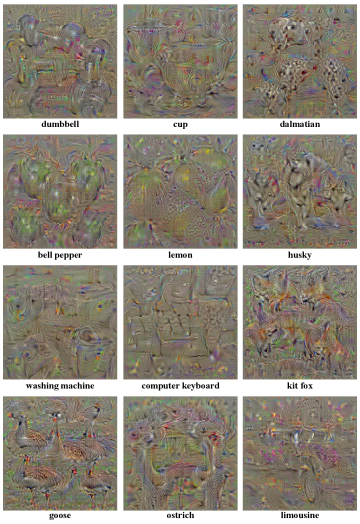


Figure 22: Class visualisation with weak regularisation term

In [YCN+15], the authors reproduced the same idea with different regularisation methods to have smoother representations. They introduced 4 methods that would help us having a better representation of what a neural network is actually looking for. They also argued that the combination of L2 regularisation and blurring was the best method to obtain clear images that would maximise the score of a neuron. The idea behind blurring the image is that we want to penalise high frequencies. The second advantage of their method is that they found out that they are also able to see entangled representations of concepts at earlier layers like convolutional layers. In the paper they do not explain how they are able to compute the images they obtain since convolution filters do not give scalars. But we can imagine that they obtained those values by first removing the ReLU activation functions for the last layers that they are currently looking at. The purpose of this would be to avoid dead gradients and since we are doing gradient ascent our goal would also be to remove the negative values. Secondly, they might also compute the sum of the activation maps associated to the filter. The greater that scalar is, the better it is. The authors also provide some hyperparameters for the neural network they used (AlexNet) to reproduce their work.

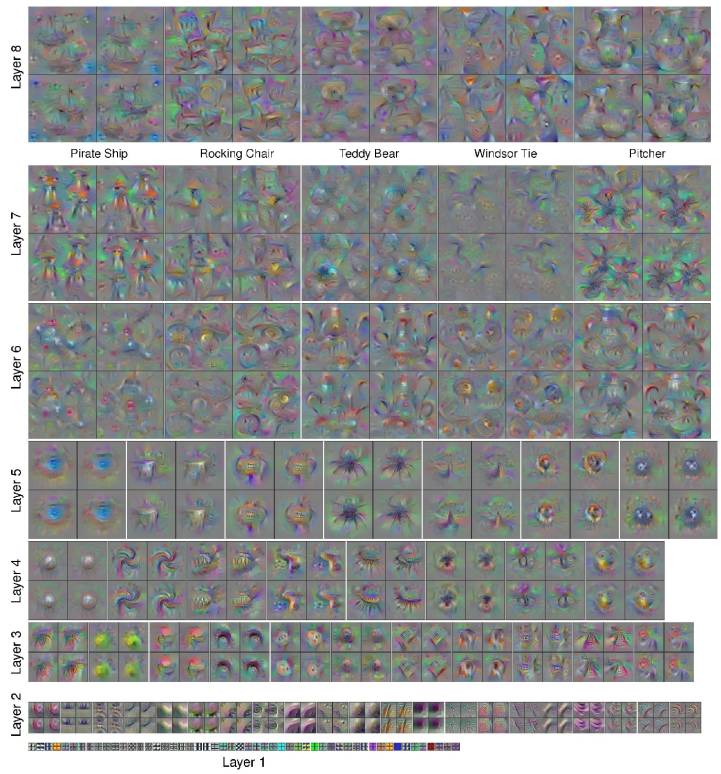


Figure 23: Class visualisation at intermediate layers with strong regularisation

3 Implementation

3.1 Global architecture

As I mentioned beforehand, we agreed with my two supervisors that we would like to build a web application in which a user can play around. To do so, I first needed to identify the technologies that we would use. It appeared clearly that the backend would need to be implemented with Python as Tensorflow's main library is in that language. As a result, we decided to go for Flask which is an open-source framework for web applications in Python. For the frontend, we decided to choose VueJS that is also an open-source Javascript framework. It is used for user interfaces in web application development like Angular or React. The toolbox would then be deployed on a specific computer running with RTX GPUs to make the code run faster. I also developed my web application using Gitlab in order to track the versions over time. In the beginning we also wanted to add the possibility for a user to upload his own model either if it was in PyTorch or in Tensorflow. As a consequence, I started to implement code that would translate a PyTorch model to Tensorflow using ONNX. Nevertheless, it appeared that it was quite complicated because the translation needed to have the input shape hardcoded or make the user input it, which might be cumbersome. Thus, for this internship we decided to only go for an implementation that would use some Tensorflow models and give explanation for these models. The different methods which were implemented are presented later.

3.2 Backend

The backend implementation consisted in two main steps, the first one was to implement the interesting methods which were described above in Python. Indeed, the objective was to implement classes and methods that would make the development of the web application faster. We have 7 classes that will be useful to the implementation:

- $NeuralNetwork$
- $ConvolutionalNeuralNetwork$
- $ActivationMaps$
- $DeconvolutionalNeuralNetwork$
- $MaxUnpool2D$
- LRP
- $PathFinder$

The $NeuralNetwork$ classes inherits from $tf.keras.Model$, its goal is to provide all the functions that will be related to gradient methods. One can for example load a model and create a $NeuralNetwork$ object to then use vanilla or integrated gradient. Inheriting from $NeuralNetwork$, $ConvolutionalNeuralNetwork$ represents a specialization of that class. The idea at the beginning of this project was to make the code as general as possible and thus we segmented two kinds of neural networks to make it usable for different type of architectures. $ActivationMaps$ class provides some methods to give information and display images from activation maps. This is in particular interesting for us as these images are important for DeconvNet representations. Following the activation map, the $DeconvolutionalNeuralNetwork$ class creates a mirror of a $ConvolutionalNeuralNetwork$ object. This is used in combination with the class $MaxUnpool2D$ to upsample and recreate the input picture starting from a specific feature map in the neural network. LRP via its method $get\_lrp\_zero()$ handles the backpropagation and gives an $LRP-0$ explanation. Finally, $PathFinder$ uses UMAP to reduce the dimension of the data to display it either in 1 or 2 dimensions. The second step was to use Flask to create and render HTML pages. Each web page corresponds to a different method. Flask is simple to use however some methods like class visualisation needed to retrieve and display some information in real time. Therefore, the use of web sockets, here Flask-SocketIO, was mandatory for our implementation. Web sockets instantiate a bidirectional connection between the client and the server and allow them to communicate asynchronously (without refreshing the web browser).

3.3 Frontend

After I finished the implementation of the methods in the backend, I started developing the VueJS application. VueJS works like Angular, that is to say that you can create dynamic components and display them easily in HTML documents. For designing reasons I decided to use Bootstrap which is a toolkit to design and customize responsive websites. I chose to use the CDN instead of the CLI version for all the dependencies

that will be discussed in the client side to make the code as simple and light as possible. The toolbox is relatively easy, the idea is that you have a homepage that describes all the methods that were implemented alongside with the papers they come from. Then the user can select in the navigation bar at the top which method he wants to study for the preloaded models: gradient saliency maps, Class visualisation, DeconvNet or LRP. For all the tabs, explanations can be easily computed, the user only needs to click on the green *AddExplanation* button to create a new container in which he will be able to compute the method that he is looking for and then play with the different parameters. To assist the user an information website is accessible by clicking on the information icon at the end of the description at the top of the web page. For shorter help, tooltips are available by hovering over the question mark icons. Figures 24, 25, 26 and 27 represent examples of the toolbox user interface. From a technical point of view, except for the homepage, every tab consists in two main components. A principal one in which you can add and remove different explanation. Every time a user adds a scenario or remove one by clicking on the cross at the top-right corner, the main component adds or removes accordingly a child component that will handle its own data. It is important to notice that the user data is saved on client side, that means that if a user leaves or changes the tab, their data is stored and they can come back without worrying of losing what they have computed. Regarding the requests to the server, I needed to do asynchronous calls that would allow us to keep the data across the pages and prevent reloading, as a result I used Axios which is a library to do asynchronous HTTP requests like AJAX with VueJS. Finally, we also needed to implement web sockets in the client side and decided to use SocketIO.

3.4 User Interface

3.4.1 Gradient method

To compute a gradient explanation, the user needs first to upload a file, the design is exactly the same as it is in Figure 27. The users can change 4 parameters depending on the use case. The model type which is the neural network architecture, the explanation which is in this case the method that will be computed. Currently there are 5 methods: Vanilla gradient, integrated gradient, smooth gradient, GradCAM and smooth GradCAM. The checkbox, if selected, applies a dot product between the explanation and the input image. Finally, the user can select the target neuron which is here associated to a specific label. A loading bar is present for all the methods since we need to make the user understand that the methods are currently running. Figure 24 represents what the user can expect for the gradient method tab.

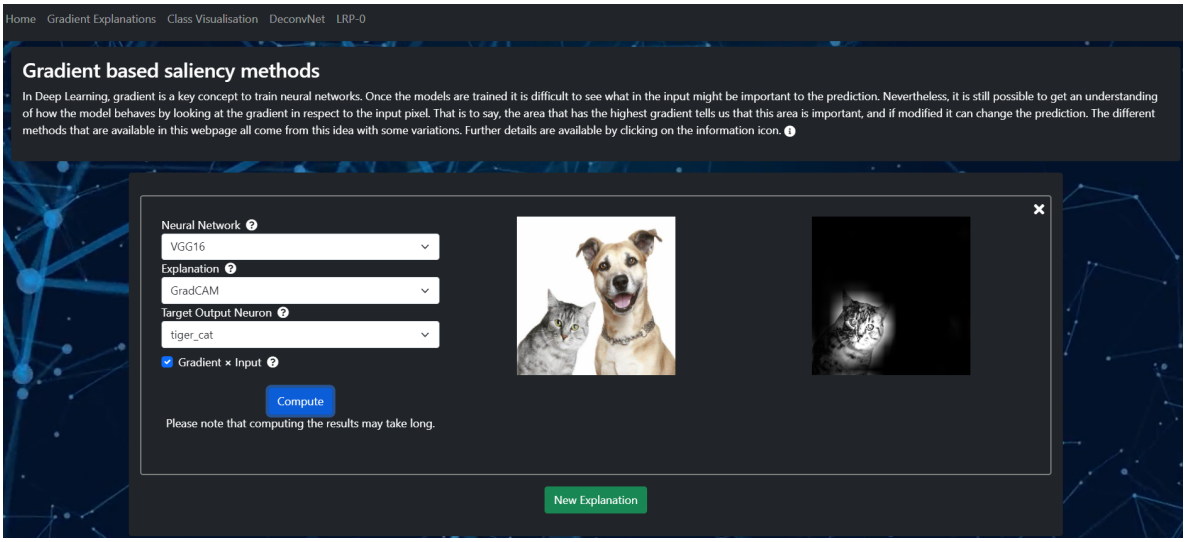


Figure 24: User interface for the gradient methods tab

3.4.2 Class visualisation

Class visualisation interface is a bit different as more parameters are involved. Furthermore, the user does not need to upload any image as the input image will be random gaussian noise. Figure 25 describes the interface for the web page. On the left, all the parameters that the user can modify are displayed. Like all the methods which are in this application, the user needs to select an architecture. Then they can choose a layer and a neuron to target, this neuron can either be a label, or it can represent convolution filters or intermediate neurons. To add regularisation terms, the user can increase or decrease the weight-decay or the standard deviation and occurrences of the blurring. The mean channels represent the mean value of the dataset for each channel (RGB) on which they were trained. Finally, the step and learning rate can be adapted to the scenario.

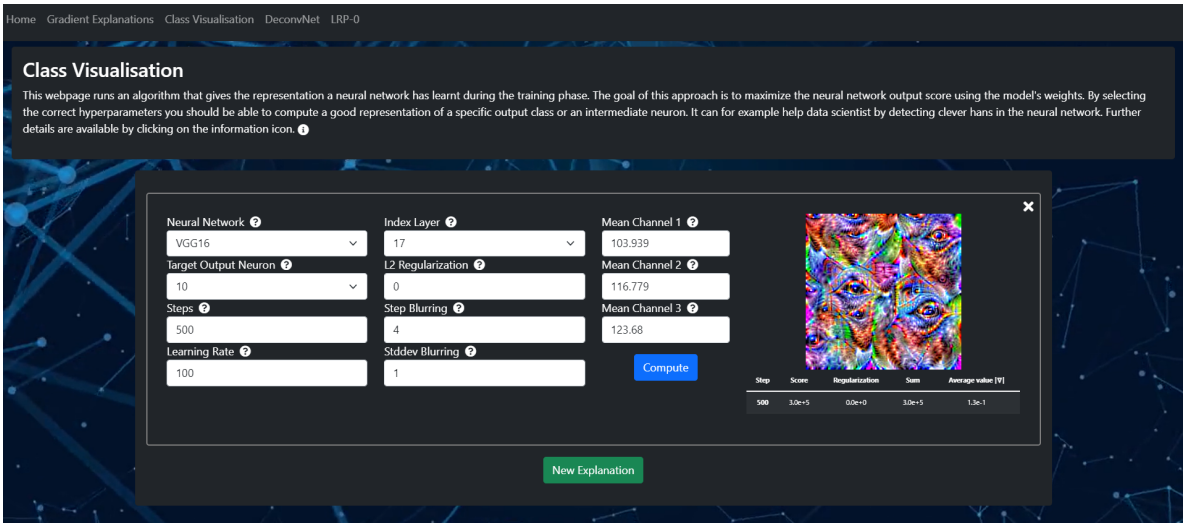


Figure 25: User interface for the class visualisation tab

On the bottom right corner, the progress of the algorithm can be monitored. Using

web sockets, the client is informed in real time about the score (that should increase) and some other important values. For example, it is advised for a user to compute a first iteration with 1 step to obtain the average value of $|\nabla|$ in order to set a correct learning rate. The learning rate should be approximately equivalent to $\frac{1}{average|\nabla|}$. Using this approach it enables the algorithm with 200 hundreds steps to converge to a correct representation.

3.4.3 DeconvNet

The DeconvNet interface works like gradient methods. The user provides an input image, uploads it and plays with the parameters. Alongside with the architecture to load, there are 2 options that the user can modify. The index of the convolutional layer specifies the convolutional layers of the neural network. A second parameter is given by the convolution filter or feature map to target. These values can be sorted by intensity when the enabled checkbox is selected. To calculate good representations, users can select the highly activated maps and display them. Figure 26 represents the described interface.

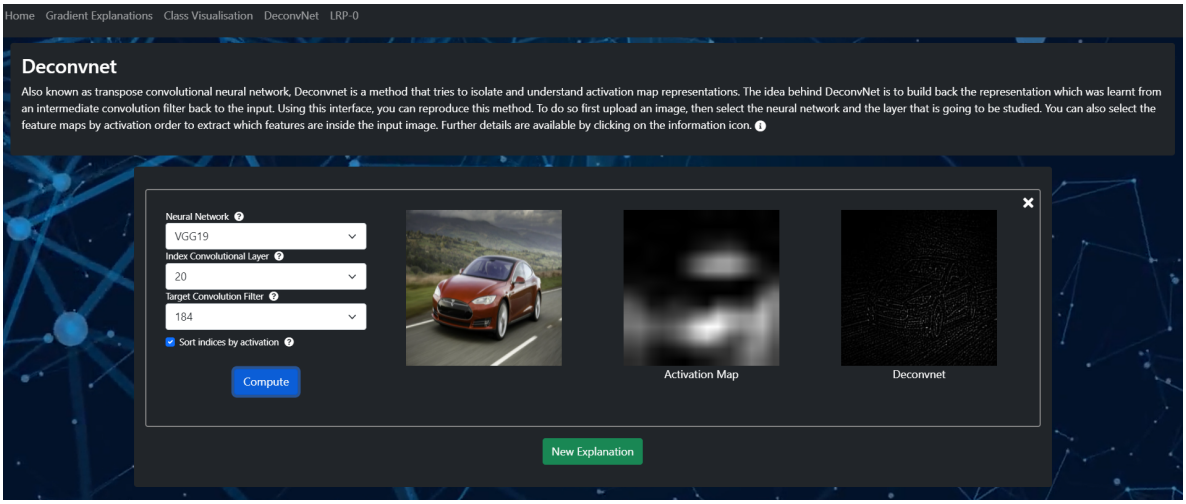


Figure 26: User interface for the DeconvNet tab

3.4.4 Layer-wise relevance propagation

The last user interface presented in this toolbox is used by LRP. Since this method is similar to gradient methods, there are only two parameters: the architecture and the neuron to target. However, since the computation may take longer for the current implementation, I decided to add another table indicating the current layer in the backpropagation. When the algorithm is running, the index of the current layer that computes relevance scores will be displayed on the bottom left corner of the container. It will stop and print an image when the index reaches the first layer.

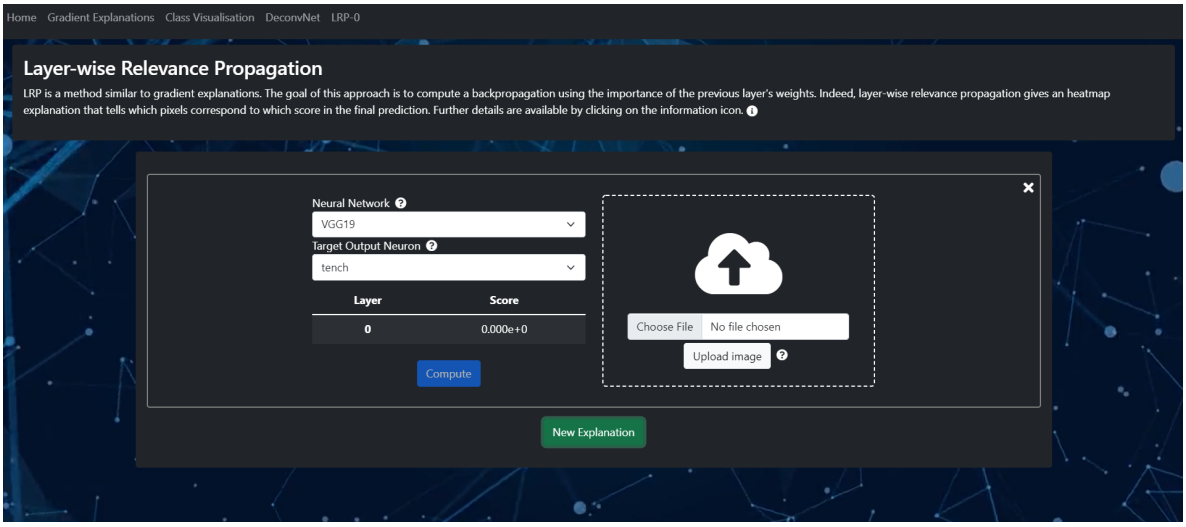


Figure 27: User interface for the layer-wise relevance propagation tab

3.5 Results

At the end of the internship, all the saliency map methods that were presented in the previous section were implemented, including vanilla gradient, integrated gradient, smooth gradient and gradCAM. This was easily done as these methods were already available using PAIR, an open-source code for gradient explanations in Tensorflow and PyTorch. The functions which are developed for Tensorflow use $GradientTape()$ which is a built in method available in Google's library to compute gradient using automatic differentiation. Other methods were trickier to implement, for example class visualisation. Indeed, this method was already implemented in Caffe but not in Tensorflow 2. As a result, we had to implement it from scratch. In that case we had to use automatic differentiation, which is normally used when we train models. Here, we needed to compute the gradient ascent for a specific intermediate or output neuron. It is important to say that intermediate neuron may also be activation map. I already described that concept earlier in the report.

Figure 28: Result of GradCAM; result multiplied by the input picture for the tiger cat class

The dimensionality reduction algorithm was almost finished, however there was a problem that was difficult to address regarding the data to load. Indeed, we need to

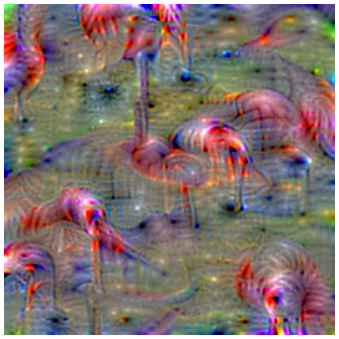
have data to be uploaded to create clusters and the question is how are we suppose to have this data as it requires the user to upload it and these kind of files are too huge. But apart from that UMAP is working well, and we can clearly see in Figure 29 that clusters are created after processing the data through the neural network.

(a) Raw data

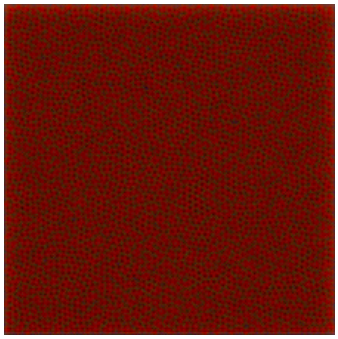
(b) Data after last convolutional layer

Figure 29: UMAP algorithm to reduce the dimension of the data for the same model at two different layers

Implementing DeconvNet was also difficult for different reasons. The first reason was that we were not sure if we could use the weights which were given in the corresponding convolutional layer for their transposed one, as it was not specified in the original paper. At the end, we decided to save the weights from the original convolutional layers give them back to the transposed layers and it appeared that the results were fine. Another problem that appeared was the Upsampling operation. In fact, the problem was due to the index of the MaxPooling layers which were not saved during the forward pass. It was difficult to find the solution as there are not that many resources online about this. We finally found the answer in the documentation of Tensorflow. We used the $max\_pool\_with\_argmax()$ function in Tensorflow. It allows the neural network to save the position of the highest value depending on the pool size of the pooling layer.



(a) Flamingo class output neuron



(b) 4^{th} convolution filter of the 4^{th} layer

Figure 30: Class visualisation of two different layers representing what the VGG16 neural network has learnt

Using DeconvNet, activation maps and class visualisation, we are able to identify which activation map corresponds to which feature detector. For example, Figure 31 shows that giving an input image of a tiger cat induces that the most activated activation map in the last convolutional layer corresponds to the 286^{th} convolution filter, using the deconvolutional neural network we can identify the fact that it corresponds to the cat face in the input image. Finally, we can try to see which image we can obtain by maximizing the activation map for that convolution filter and we can maybe identify a cat face pattern.

(a) DeconvNet top activated map

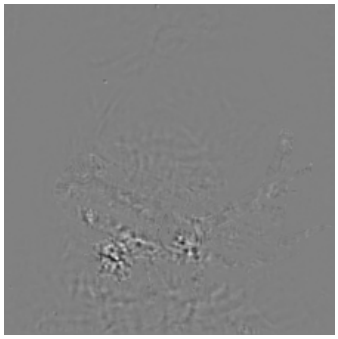
(b) Class visualisation for the same filter

Figure 31: Analyse of a convolution filter using DeconvNet and class visualisation of VGG16

The last method that we implemented is LRP-0. Like class visualisation and DeconvNet, we did not find an existing implementation for Tensorflow and had to do it ourselves. Fortunately, we could use the max unpooling operation that we have already done for DeconvNet. The problem for this algorithm was to do the backpropagation ourselves for Conv2D layers. Indeed, in LRP we have to compute the relevance score for each input neuron for all the next layers and for convolutional layers, which is quite time consuming. In the beginning for example I did a code that was running for several minutes to compute one backpropagation step (that is to say for one layer). After listening to Dr. Philip Trautman's advice I decided to use matrix multiplication using

Numpy. And after having optimized the algorithm several times we decreased the time by a huge amount. Now running on a simple laptop, it takes 1 minute to compute the backpropagation and it can be decreased to 24 seconds when we use a computer with a powerful CPU. This operation is even faster by running it on a GPU, it can for instance be on par with the implementation done by Fraunhofer (approximatively 5 seconds).

(a) Image representing a plane



(b) LRP explanation for the correct class

Figure 32: Layer-wise relevance propagation for a picture of a plane

3.6 Deployment

The implementation works using three models VGG16[SZ15], VGG19 and a variation of AlexNet[KSH12]. It appears that loading all these 3 models at the same time is quite memory consuming and as a result, we cannot use the GPU. Moreover, some implementations like class visualisation need to create a derivated model to remove activation functions for intermediate layers, indeed the problem is that removing the ReLU from the models can be a problem when we are handling multiple clients. As a result we decided to go for the CPU version of Tensorflow which uses the normal RAM of the computer (32 Gb in this case), and, as a consequence, it increases the computation time but allows several people to connect and test the application at the same time.

To make the deployment easier for IABG, in case someone wants to improve the code or add new features and also because the source code might be available on my GitHub soon, we decided to create a Dockerfile with all the requirements for the backend. Since the frontend uses CDN versions of VueJS and the other libraries which are used, building the application on new machines should be relatively simple. It was also important to have a real server and we decided to go for Gunicorn. It is a Python WSGI HTTP Server for UNIX that will deploy the Flask application which was implemented. We also created a Dockerfile for a reverse proxy called Nginx. Using those two images, new people should be able to build and serve the application on a local network without difficulty.

3.7 Related works

Many implementations have been done for explainability. As I previously mentioned, we were mainly influenced by the work done by Yosinski [YCN+15]. The code is avail-

able on GitHub and a presentation video is on Youtube. The implementation allows in particular to see in real time the activation maps to understand which convolution filters are active for which input images. It is also possible via their tool, using examples, to see which images in the dataset activate the convolution filters the most. Moreover, by using the class visualisation algorithm it is also possible for them to support the example explanations associated with these same filters. Finally, using the deconvolution algorithm, the authors can see which neurons are important in obtaining a certain feature map. The problem being that this implementation is compatible only for Caffe which is now old.

New libraries have emerged in recent years and to the best of my knowledge I am not aware of a similar implementation. Big companies like IBM presented their own toolbox that wraps already well-known methods. Google on the other side, provides its customers online debugging tools to improve their models. Nowadays, these new implementations are more intended for expert developers in artificial intelligence, because in their case, there is no simple and generalized application in which a user can modify certain parameters and obtain explanations for a pre-trained model. It is in fact important to know the different methods and how the tensors are made up to target the correct output (in the case of local methods). The new implementations online are often based on the most famous explanations such as LIME where the authors already published their code themselves. The library is accessible via *pip* in Python but it does not offer the possibility of giving explanations in real time unlike gradient methods from Saliency. SHAP is also one of the often used methods, it is also accessible on GitHub. From my personal point of view, a trend emerges which is that since the models are more and more complex and diversified (transformers, dense networks) it becomes difficult to have generalizable explanations and therefore a unique code for all the models. Nevertheless, even if our toolbox can not be adapted to every kind of model. It remains a fairly reliable and easy-to-use tool for the uninitiated in explainability. However, it is important to have a knowledge of certain technical terms in machine learning in order to fully understand certain explanations.

3.8 Possible enhancement

Before concluding, I would like to give possible ways of improving the application. It is indeed important for me to be able to tell what I have learned and the possible corrections that I would have made if time allowed it. First of all, the Layer-wise Relevance Propagation algorithm is problematic with respect to memory allocation. As a matter of fact, some tensor products are much too resource intensive, especially for the convolutional layers in the $back\_conv2d\_lrp()$ function. It would therefore be important to take a look at how to modify this function without slowing it down. Currently, VRAM is very expensive and with such a version of LRP it seems unreasonable to use a GPU. This unfortunately represents a major disadvantage for the application. In connection with this same idea of improving resource allocation, LRP and the class visualisation algorithm create a cloned model at each call in order not to corrupt the results of the other explanations which are calculated at the same time. This means that if several instances are created simultaneously, the memory can be emptied. In order to solve this problem it would be interesting to see if it is possible to lock the access to the model used during an iteration to avoid any race condition. Still concerning

class visualisation and LRP, if the calculation time remains substantial despite the modifications, it would be interesting to add the possibility for a user to interrupt the calculation of the explanation. If the parameters for instance in the first mentioned method are bad, it would be nice to be able to let the user stop the program, modify the parameters and then resume where he stopped it. Also, in the current version of our implementation, there is no real input data control, which means that a user can modify the input values in order to crash the application, for example if the user enters a negative number of steps for the class visualisation algorithm. It is also possible for a malicious user to craft his own HTTP requests in order to modify the behavior of the application. Therefore, in the eventual case that IABG would like to improve and deploy this application, it is necessary to impose more control on the forms that are sent. This can be done on the client side to improve the user interface, but it is mandatory on the server side to check the values entered. As I said before, three models are used in this application. VGG16 and VGG19 work perfectly well for all the algorithms, however for AlexNetFused, some algorithms seem to give strange values. In the context where a developer would like to resolve certain inconsistencies, I will give some leads in the next few sentences, however I would like to remind you that this is only an example, and that it would be more interesting to apply these advice on newer models. For class visualization, the current implementation gives dark values, this is due to the fact that the preprocessing phase linked to VGG16 or VGG19 is very different and therefore the algorithm should be adapted to the input values of the model which are not the same since they are normalized in this case. Furthermore, the implementation of DeconvNet seems to give erroneous values if one wishes to visualize deep layers. There are two explanations here: either the BatchNormalization layer is badly treated, or the strides and pool size in the convolutional layers do not work well for dimensions larger than VGG16. For UMAP, it is difficult for me at the moment to give an appropriate solution, except in the context where we only give an idea to CNNs trained on ImageNet, it seems to me difficult to give a solution unless the program has access to the dataset, and even with this access, I do not guarantee the computation time. Subsequently, if the program becomes more developed, I recommend using VueJS with its CLI version, which may be more suitable and more readable than the CDN version currently used. Finally, when the problems related to memory allocation will be solved, it would be more interesting to use graphic cards instead of the CPU to run the program. The benefit will be significant for the user experience and for the program itself because it will prevent it from running several tasks at the same time, given the fact that they will be processed more quickly.

4 Conclusion

4.1 Feedbacks on the toolbox

The realization of the application allowed me to reach several conclusions about the implementation of an explainability application as a service. First of all, not all methods are universal. Indeed, some methods can be used for all kind of models, i.e most gradient based methods. Nevertheless GradCAM, NBDT or Network Dissection clearly demonstrate the non-universality of all methods. Secondly, it can be clearly seen that

the methods are not bulletproof. For example, after implementing the different approaches, we can see that even if most of the time the results are consistent, some explanations remain confusing. Moreover the differences in practical terms of the methods are notable. For example, I would like to support this point by the fact that some have real scalability problems. UMAP for example, without going through the mathematical details, remains quite simple to understand. However, its implementation is quite complicated to set up. This is due to the fact that it requires the user to be able to transmit large amounts of data to train his neural network to associate the same concepts in a low dimension (without forgetting that this type of explanation remains doubtful because we add another model in the explanation loop). UMAP is not the only approach impacted by this drawback, NetDissec has the same major problem because in order to make the neural network dissection algorithm work, the user must have a suitable and segmented dataset. In a simplified way, we understand that the more the methods are global, the more they tend to be difficult to set up. Another observation that can be made is that each model is different and therefore no method can be implemented directly for all neural networks. If we take the example of AlexNet used in our web application, the input values are normalized while for the VGG16 and VGG19 implementations of Tensorflow the values are centered but not normalized. Moreover, in this case, the input channels are inverted which makes the task even more difficult. The preprocess or postprocess phases are not the only reasons why the algorithms are not generic. Indeed, the tensors are generally very different. In PyTorch's implementation of VGG16 the channels are reversed compared to Tensorflow and therefore this makes the developer to adapt each implementation. After having noted all these disadvantages, one can wonder if the realization of a library like Saliency should not be more adapted, because in this case each developer or user can adapt his code to his need and consequently avoids having the problems mentioned above. Indeed, it is always possible to create a system that explains a neural network like what was developed during this internship or by Fraunhofer, however as each model is unique it is necessary to adapt the code to the model studied and this remains a big drawback. Furthermore, even if explainable artificial intelligence is a growing field aimed at making users and scientists understand how neural networks work, we can very quickly see that this field requires good expertise and a significant mathematical background. I could actually see through my research that one of the objectives of explainability was to make models easier to understand. However, without real knowledge of the terminology associated with these models as well as those related to their underlying mechanisms, the explanation remains a result without real meaning. This argument has indeed been underlined in the paper called Sanity Check for Saliency maps [AGM+18] where we see that without real analytical capacity it is difficult to see if an explanation is correct. As I described previously it is not because an explanation seems correct that it actually is. Hence, it is important to understand how neural networks work.

4.2 Personal retrospective

From a personal point of view, this internship was incredible. Indeed, I had little knowledge about neural networks before. The whole internship was only possible because of my two supervisors; I would like to thank Dr. Philip Trautmann at first for all his assistance during research period. Actually, his guidance allowed me to better under-

stand scientific papers from a mathematical point of view. It was him who gave me the first directions of research. I am also grateful to Dr. Zardosht Hodaie who supported me on topics related to computer science. Thanks to them I learned so much and in so many fields. For instance, regarding theory, I acquired many skills in machine learning, convolutional neural networks, and adversarial attacks. Moreover, on the practical side, I was able to learn new things like the use of advanced Tensorflow. I was also able to learn new development tools like VueJS or Flask, and glimpsed the challenges and difficulties faced by DevOps when creating the Docker images. I would in addition like to thank my academic advisor Pr. Melek Önen who through her kindness, expectancy and feedbacks enabled me to give the best of me. I am also grateful to Dr. Martin Glas for his full support and attention to me, especially during a rather delicate health situation. It is important for me to thank Dr. Yasmine Israeli as well as she allowed me to realize this opportunity in Germany. This Opportunity represents a real plus for me because it acts as an important step, that is to say, going beyond my comfort zone that is France. Finally, I thank all my colleagues, whether masters' and doctoral students or full-time employees, they were of great help to me as they integrated me into the group very kindly. This internship represents a real shift in my future choices. Indeed, following this internship and without forgetting my different past experiences, I decided to continue my studies in scientific research. I was not sure before this internship about the idea of having to go into industry or academia and therefore to do a PhD or not. This choice seems obvious to me today after discussing it with my supervisors and some of my colleagues. Indeed, it is explained by different reasons: the first being the skills that are acquired as a result of research effort. I was able to observe a real increase in skills on my part when I tried to put into practice the different scientific papers that I was able to read. Indeed, when reading scientific papers certain questions may arise during the misunderstanding of certain passages and the resulting curiosity makes it possible to correct certain shortcomings and thus progress. In addition, this progression is even more accentuated during the implementation of these papers. I therefore deduce that when you do a PhD, advancing research also allows you to improve on certain aspects in a fairly advanced and rapid way. In addition, and still in connection with the previous argument. The people I have been able to learn the most from are people who have been beyond the master's degree. As Dr. Zardosht Hodaie says, to get things right, it is important to know exactly how they work and not repeat what others are doing. To do so one has to go in depth and thus it leads to a PhD. The last argument that is the most important for me and that really demonstrated itself over the course of this internship is the pleasure of doing technical things. I have already been able to work previously in some associations or companies in which I have been able to play different roles. It is without a shadow of a doubt during this internship that I was able to take the most pleasure during my working hours. To conclude, I thank Eurecom and IABG for guiding me through the various works, courses and internship that consolidated my professional project.

References

[AGM+18] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian J. Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In

NeurIPS, 2018.

[BBM+15] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 10, 2015.

[BZK+17] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3319–3327, 2017.

[CdMP21] Gabriel Dias Cantareira, Rodrigo Fernandes de Mello, and Fernando Vieira Paulovich. Explainable adversarial attacks in deep neural networks using activation profiles. *ArXiv*, abs/2103.10229, 2021.

[CSHB18] Aditya Chattopadhyay, Anirban Sarkar, Prantik Howlader, and Vineeth N. Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 839–847, 2018.

[CW17] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2017.

[GSS15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2015.

[HMT21] Hokuto Hirano, Akinori Minagi, and Kazuhiro Takemoto. Universal adversarial attacks on deep neural networks for medical image classification. *BMC Medical Imaging*, 21:9, 2021.

[KGB17] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *ArXiv*, abs/1607.02533, 2017.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84 – 90, 2012.

[LL17] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *ArXiv*, abs/1705.07874, 2017.

[MBL+19] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. Layer-wise relevance propagation: An overview. In *Explainable AI*, 2019.

[MH18] Leland McInnes and John Healy. Umap: Uniform manifold approximation and projection for dimension reduction. *ArXiv*, abs/1802.03426, 2018.

[MMS+18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *ArXiv*, abs/1706.06083, 2018.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[SDBR15] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2015.

[SDV+19] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128:336–359, 2019.

[STK+17] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda B. Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *ArXiv*, abs/1706.03825, 2017.

[STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *ArXiv*, abs/1703.01365, 2017.

[SVZ14] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2014.

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[vdMH08] Laurens van der Maaten and Geoffrey E. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[WDH+20] Alvin Wan, Lisa Dunlap, Daniel Ho, Jihan Yin, Scott Lee, Henry Jin, Suzanne Petryk, Sarah Adel Bargal, and Joseph Gonzalez. Nbdt: Neural-backed decision trees. *ArXiv*, abs/2004.00221, 2020.

[WSZH18] Yulong Wang, Hang Su, Bo Zhang, and Xiaolin Hu. Interpret neural networks by identifying critical data routing paths. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8906–8914, 2018.

[XWvdM+19] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan Loddon Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 501–509, 2019.

[YCN+15] Jason Yosinski, Jeff Clune, Anh M Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *ArXiv*, abs/1506.06579, 2015.

[ZF14] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

[ZKL+16] Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921–2929, 2016.

[ZLZ+20] Husheng Zhou, Wei Li, Yuankun Zhu, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. Deepbillboard: Systematic physical-world testing of autonomous driving systems. *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 347–358, 2020.

[ZTLT21] Yang Zhang, Peter Tiňo, Ales Leonardis, and Ke Tang. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5:726–742, 2021.