Open University of Cyprus

***Faculty* of Pure and Applied Sciences**

Postgraduate Programme of *Cognitive Systems*

Master's Dissertation

Creation and Analysis of an Explainable Deep Learning System

Vasileios Karvonidis

Supervisor
Ilianna Kollia

December 2022

Open University of Cyprus

Faculty of Pure and Applied Sciences

Postgraduate Programme of *Cognitive Systems*

Master Thesis

Creation and Analysis of an Explainable Deep Learning System

Vasileios Karvonidis

Supervisor
Ilianna Kollia

This Master's Dissertation was submitted in partial fulfillment of the requirements for the award of the postgraduate title
on Cognitive Systems
by the Faculty of Pure and Applied Sciences
of the Open University of Cyprus.

December 2022

BLANK PAGE

Summary

Deep learning models represent the cutting edge in Artificial Intelligence - AI. However, they lack one key element, explainability. Even though such models are able to perform so accurately, they act as black boxes, without letting us having any insight into how they made a prediction, and what drove them to reach it. This problem concerns the researchers for many years, as explainability is of utmost importance in order to use Deep Learning models in critical applications, such as those in medical, finance and automotive sectors. Explainable AI consists of a set of methods and processes that allow the users to understand and trust the results of the AI model, due to the fact that there is clarity in the decision making, and we can easily characterise the accuracy, the transparency and the fairness of the model, even in a complex situation. This will help AI to become more 'responsible' to its decisions, more trustworthy and able to help larger sectors and industries to adopt it. In recent years, researchers developed various techniques, which can identify the reasons behind the decision of a deep learning model. These techniques inspire and pave the way for new methods to be developed. For example, Class Activation Mapping produces heatmaps to highlight the region of the image which the model focused on in order to classify it. This visualisation of where the model is looking helps to identify whether the model is trustful or not. For example, a model could classify a train image by looking at the train tracks rather than the actual train. Despite the correct classification, the model might take into account wrong parts of the image, which could be a consequence of poor training. A more evolved technique which is based on Class Activation mapping, is Grad-CAM. This technique is considered class-specific, meaning that for the same image, it can produce a separate visualisation for each class which is present in it. Another interesting approach is Structured Attention Graphs (SAGs). This method is inspired from attention maps, which are popular tools for explaining the decision of deep learning models. The researchers argue that just one attention map is not enough. With SAGs, we can have a set of patches as attention maps, and we record the confidence level of the model on each one in order to evaluate how the model is impacted. This thesis will mainly focus on Grad-Class Activation mapping and Structured Attention Graphs. We will explain the procedures behind the image classification, we will benchmark the techniques and see how they apply in various datasets. We will also analyse their role in the general structure of explainable artificial intelligence.

BLANK PAGE

Acknowledgements

I would like to thank my supervisor Ilianna Kollia, for her guidance throughout this Master Thesis dissertation, as well as my family and friends for supporting me in this journey.

Contents

BLANK PAGE

Chapter 1
Introduction

Day by day, we get more attached with Artificial Intelligence in a similar way we got attached with the electricity and the internet. In its early days, AI was heavily affected by work made of psychologists and neuroscientists, as they were trying to understand and reproduce the full range of intelligence and behavior that is observed in humans in computational systems. But later on, this view was abandoned as more mathematicians and computer scientists started to work on this. Then, the emphasis moved to more quantitative results and metrics of performance. The goal was to make systems that are performing well in real life scenarios, to optimize and increase their performance. Nowadays, it is well established that AI solves and automates a vast number of problems in which, until some years ago, classic methods could face many challenges. This alone indicates that there is an utmost need to develop systems that could be found useful to each and every user in any major sector. Simultaneously, tasks become more challenging and complex, which naturally lead to the need of more sophisticated solving methods.

One significant argument for cognitive systems to be trusted in critical environments, is usually the lacking of adequate evidence of the system's conclusion, as more often than not, users need to know the origins of the system's answer. This led cognitive systems development research to be focused towards personalization, explainability, and transparency, concepts that are tightly connected and strongly affected by each other. All these, when they are combined properly, can lead to the construction of components that are necessary for cognitive processes to exist, such as sensing components (the input), memory, reasoning, learning and interacting components. With these components, we have built our way up to higher level ideas in order to define and construct cognitive architectures.

From this point the problem narrows down to the appropriate cognitive architecture implementation with regards to the application, and there are plenty that can be taken into consideration. This thesis will focus on the explainability techniques that are developed and used in deep learning systems, in order to determine whether or not the model performance is accurate and honest, in the sense that is able to support its results, based on valid information that it can find in an image. The domain of application will be the medical sector, and more specifically the medical recognition of Parkinson's disease in image scans such as DaTSCAN. In the following lines, we will briefly explain the aforementioned concepts, in order to provide clarity and argue why is necessary to enrich deep learning systems with explainability techniques.

Goal of AI is to provide personalized, comprehensible and transparent services. Deep learning models represent the cutting edge of artificial intelligence (AI). In Machine Learning, we teach the computer how to process and learn from the data. In deep learning, the computer trains and continuously reconfigures itself to process and learn from the data. Nevertheless, until recently, they lacked the key element of explainability. These models can perform so accurately, but they act as black boxes, without knowing how they made a classification and what drove them to reach a certain conclusion. This problem concerns the researchers for many years, as explainability is of utmost importance in order to use deep learning models in critical applications, such as those of medical, finance, military and automotive sectors. Explainable AI (XAI), consists of a set of methods and processes that allow the users to understand and trust the results of the AI model, due to the fact that there is clarity in the decision making, and we can easily characterize accuracy, transparency and fairness of the model, even in complex situations. This helps AI to become more responsible with regards to its decision, and will help larger sectors and industries to adopt it.

This thesis will mainly focus on grad-Class Activation Mapping (grad-CAM) and Structured Attention Graphs (SAG). We will try to understand and explain the procedures behind an image classification, and we will benchmark the techniques. The techniques will be applied in a dataset of images, which contains patients with signs of Parkinsonian syndrome. In this Thesis, we used datapoints from the dataset of the research "Machine Learning for Neurodegenerative Disorder Diagnosis — Survey of Practices and Launch of Benchmark Dataset" (Tagaris A. et al., 2018:1). Thus, a deep learning model will be created in order to be able to recognize healthy and non-healthy subjects. Afterwards, the deep learning model, will be evaluated not only based on its performance, meaning whether the prediction is correct or not, but also on the quality of its answer. The quality will be determined by the explainability algorithms which will show if the deep learning model is focusing to the right parts of the image to make a prediction. The high hopes we have with this thesis is to be able to contribute to the customization of deep learning models in order to be later implemented in completed cognitive systems that could help in any sector, which requires image processing, such as the one we focus here, the medical sector.

More specifically, in chapter 2 we review the related work and literature we based this thesis. We review the course of deep neural networks throughout the history, we make in comparison between the biological neural systems versus the artificial neural systems, and we analyze the various architectures of convolutional neural networks. Then we dive in the analysis and review of explainability techniques. First, we analyze the attention and perception mechanism in humans. This pins down the origins and inspiration of the explainability methods, how they are related to human nature, and how we can extend these cognitive capabilities when investigating a problem in a specific domain. Then, we analyze the main types of explainability, intrinsic and post hoc, their attributes, their differences, and their sub-categories. Our focus lies on post hoc techniques as we analyze in that section. Afterwards, we analyze and review the saliency map explainability techniques we focus on. Class activation mapping, which is a technique that produces heatmaps to an image, in order to highlight the class specific regions of it. Gradient-weighted Class Activation Mapping (Grad-CAM) is also a popular technique to visualize where the convolutional neural network draws its attention for the classification of an image (Selvaraju R. R., 2016: 4). It is also considered as a evolution of Class activation mapping. We briefly refer to High-resolution class activation mapping, also known as HiResCAM, is argued to solve the problem of erroneous highlighting in grad-CAM (Draelos, R. L., 2021: 2). And lastly, we refer to the work of Shitole et al (Shitole V, 2020: 6), which argue that a single saliency map that is produced by the aforementioned techniques is not enough, as it provides an incomplete understanding, since there are often many other maps that can explain a classification equally well, and this inspired him and his colleagues to develop Structured Attention Graphs. Also, in this chapter we analyse the Parkinson's disease and it detection methods.

In chapter 3, we analyse our application of the explainability techniques upon models which we developed and trained on a dataset with DaTSCAN images. We analyse the environment, the flowchart of the model creation algorithm, and every algorithmic step separately. We see in detail how we setup the initial parameters, how we proceed with the dataset pre-processing, the process of the creation of the model, the training and testing procedure, the saving of the model and how we keep its metrics. Then we review the various challenges we met during the development of the thesis and its application, and in the end, we review and analyse the performance and the results from the experiments and the models we developed with SAG algorithm and grad-CAM algorithm. In chapter 4, to conclude, summarize, commenting on results, and we propose some ideas for future work.

Chapter 2
Review on Deep Neural Networks and Explainability Techniques

In this chapter, we will analyse the concept of Artificial Neural Networks and the background of the explainability techniques that we experimented in thesis. More specifically, we will run through the challenges they faced throughout their history, analyse their origins of inspiration and how the pioneers of this discipline made deep learning methods thrive in various fields. This chapter mentions the different types of artificial neural systems, what they solve and how they solve it. Afterwards, we will focus on the convolutional neural networks, as they are used for image processing in the rest of the thesis.

2.1 Deep Neural Networks

'Deep neural networks' is one of the many references that describe the same concept. More often they can be referred as artificial neural networks or deep learning networks. History of deep neural networks start from the ideas of Frank Rosenblatt. In his first book "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms", in 1962 (Orbach J., 1962: 218), he structured and developed the basic concepts of which a deep neural network should be governed. In the late 60s, Alexey Ivakhnenko et al (Ivakhnenko A.G, 1967: 123) created the first eight-layered working system. The extreme amount of memory and process power these algorithms needed at that time due to their "nature", along with the application demands which got higher and the hardware technologies, were not ready to support them yet. This resulted to a massive disappointment and a great doubt towards their usability. Despite the fact these obstacles were eliminated in the early 00s, the majority abandoned the idea of solutions where deep neural networks took part.

All changed, thanks to Hinton and his colleagues (Hinton G.E, 2006: 3), with its "conspiracy" of the deep learning systems. Until then the term "Artificial Neural Network" was despised by many researchers, but Hinton and his colleagues, managed to rebrand the concept in an elegant way. In the aforementioned paper, Hinton et al. showed how multiple deep layer artificial networks can work efficiently if their training is efficient. Within the next years, various techniques were created, most with a specified use case. In order to analyze the explainability techniques, first we need to pin down the idea of the deep learning systems. In the following pages we will start simple by analyzing the basics of a deep neural networks, then we will underline the most common methodologies that are being used, by focusing on the most often used systems for image recognition, the Convolutional Neural Networks.

2.1.1 A Comparison Between Artificial & Biological Neural Networks

In order to have a better understanding of the design and the techniques that are used in deep neural networks, we need to start simple. As in many cases, here the technology observes and mimics the nature, so deep neural networks are inspired by the biological neurons and the functions of the brain. The simplest neural network, is constructed with one neuron. In reality, even a single neuron is an extremely complex mechanism. A biological neuron may vary on how it gets activated, as it depends on its structure the structure of its neighbors. The simple operational concept, dictates that once the neuron receives an electrical signal (or multiple signals), if the amount is enough to surpass a pre-determined threshold, it gets activated. The biological neuron in simple terms could be described as an electrically excitable cell, which consists of:

- The nucleus, the kernel of the neuron.
- The soma or the body of the neuron.
- The dendrites, cellular extensions with branches, the input system of the neuron.
- The axon, the output system of the nerve.

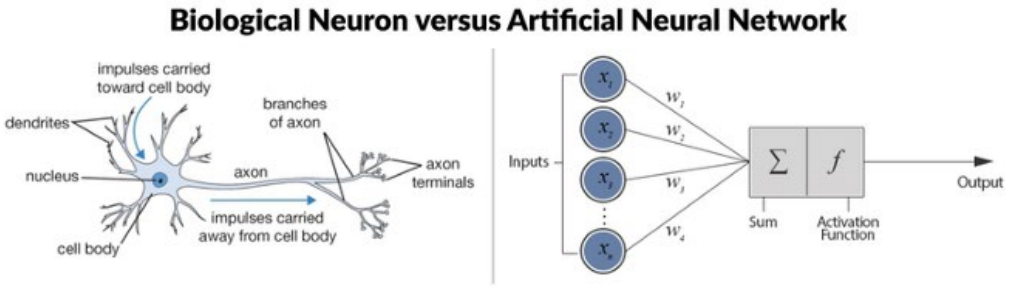

Figure 1. Biological Neuron versus Artificial Neuron (Moumene et al.,2020: 3)

By simulating the concept of neuron, we can construct an artificial neural network. An artificial neuron is consisted of the following.

- The input, which simulates the stimuli, the input system.
- The weights, which simulate the synapses.
- The sum, the product of the inputs and weights.
- The activation function, which simulates the threshold.
- The output, simulates the axon, the output of the neuron.

So, it is intuitive to comprehend that when multiple neurons are connected in a proper way, and they are able to produce useful outputs, the construct is a network of neurons. Neural networks consist of layers of neurons; we have the input-layer, the hidden layers, and the output-layer.

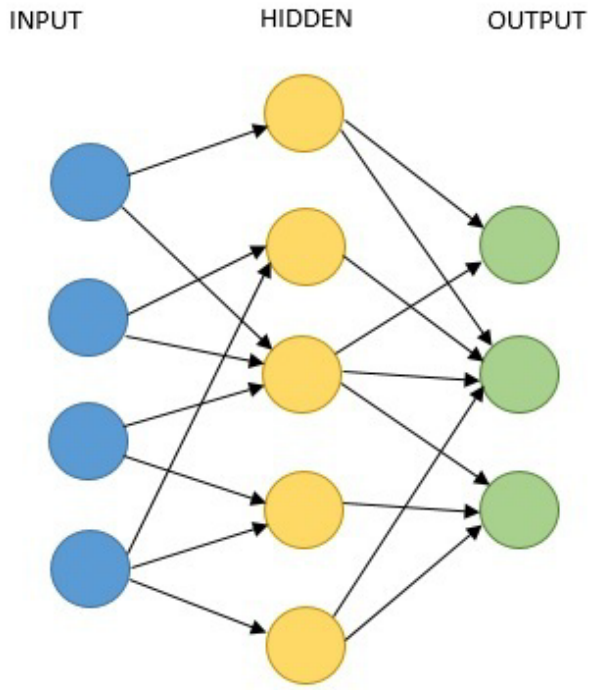

Figure 2. An abstract representation of an Artificial Neural Network. (Jadhav V.,2016: 7)

Abstractly, the information enters the network through an input layer, in a pre-determined format. For example, if the input is an image, it is needed to be re-sized in a certain pixel by pixel size, and each pixel should be placed in an exact input. The input layer passes the information to the next hidden layer, where depending on its weights,

certain neurons are activated and pass this information to the next layer, until the output layer is reached, which gives a result, also known as prediction.

Of course, real life applications need more complex structures than the one we just described, so this complexity requires a deep neural network. The usage of complementary priors helps to derive fast, greedy algorithms that can learn deep directed belief networks, one layer at a time, provided the top two layers form an undirected associative memory (Hinton G.E, 2006: 4). In general, deep learning algorithms give solution where classic Machine Learning cannot provide adequate results. There are two main reasons why a scientist may choose the first than the latter. First, is about the decision boundaries. When a classification problem takes place, an algorithm learns a function which separates two classes. This helps us determine if a given data point belongs to one class or the other. Such algorithm may be for example the linear regression. The second reason is about feature engineering, which is the most important step when it comes to the model building process. It consists of two stages, feature extraction, in which we extract all the required features for the problem, and feature selection, in which we select the important features in order to improve the algorithm performance. If we take as an example an image classification task in a machine learning algorithm, we have to handle manually the feature extraction, which needs a strong knowledge of the subject as well as the related domain (IBM Cloud Ed., 2020: 1). However, when it comes to deep learning algorithms, feature extraction is automated.

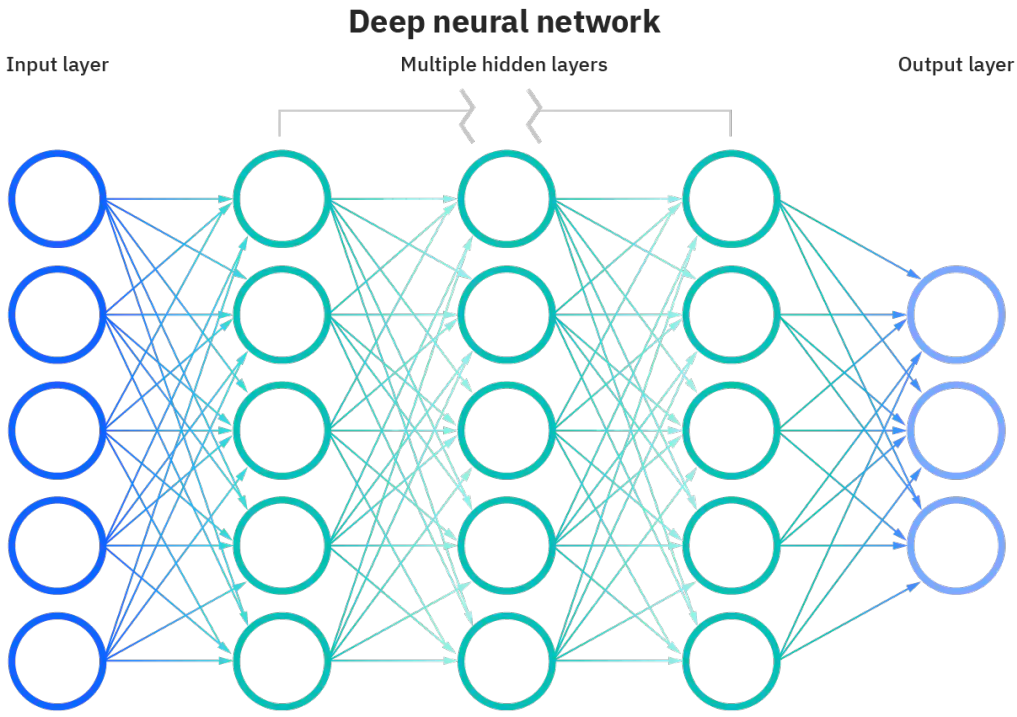

Figure 3. An Abstract Representation of a Deep Neural Network (Jayaweera P., 2021: 5).

Since the blooming of deep learning, many methodologies have been created in order to face tasks where other AI systems faced great challenges. One obstacle is the vanishing and exploding gradient due to backpropagation. In a case of an Artificial Neural Network with multiple hidden layers, the gradient vanishes or explodes as it propagates backwards. These limitations have been overcome by the various architectural structures that have been (and being) proposed. In the following lines of this chapter, we will briefly refer to the most notable techniques, and we will extend the analysis of the Convolutional Neural Networks.

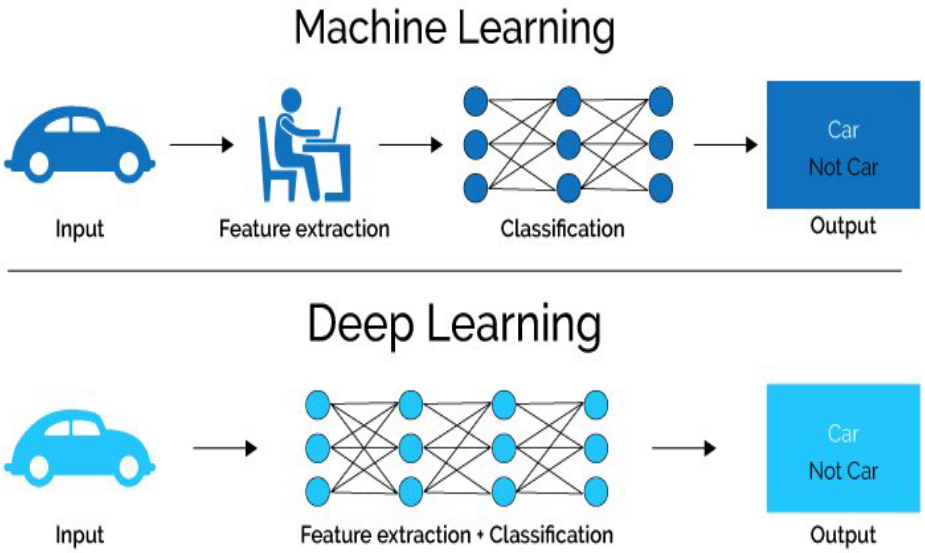

Figure 4. Machine Learning versus Deep Learning (Agrawal S., 2021: 149).

2.1.2 Convolutional Neural Networks

Convolutional neural networks, or CNN for short, are designed for processing grid-structured inputs. Images are the most common data that a CNN is used for, as it tends to exhibit spatial dependencies and detect adjacent spatial locations, which often have similar color values on the individual pixels (Aggarwal C.C.,2018: 315). Along with the extra dimension which specifies the color of the pixel, inputs often are represented as three-dimensional. Thus, all these features are depended based on spatial distances amongst one another. In simple terms, the mission of a CNN, is to detect patterns in an image, and classify the image accordingly.

Convolutional neural networks are consisted of neurons that have learnable weights and biases, similarly structured as the common Artificial neural networks. The basic structure of a CNN consists of various layers, which everyone has a different purpose. First, we have input layer which accepts the multi-dimensional image and feeds it to the convolutional layer. The convolutional layer filters the image with the help of kernels in order to highlight certain patterns. These kernels are smaller in size compared to the image. Then the convolved feature may be subjected to pooling in order to reduce its spatial size, and therefore decrease the computational demand of the data processing. Pooling also helps to highlight the dominant features of an image. Depending on the technique, there might be many successively module sets of convolutional and pooling layers. If the technique dictates that the layer sets are enough, we need to flatten the output and feed it to a fully connected layer in order to proceed with the classification.

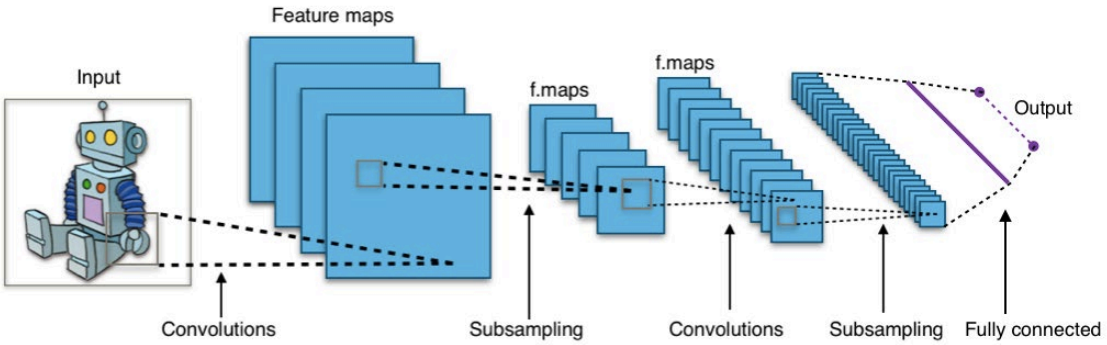

Figure 5. The framework of a Convolutional Neural Network (Kone C., 2019: 1).

2.1.2.1 LeNet-5

LeNet-5 affected massively the concept of convolutional neural networks. It was developed by Yann Lecun et al. in 1998. Its purpose was to identify images that are normalized and centered with regards to their size, such as handwritten digits. Each layer subsamples the product of the previous layer's feature maps. These feature maps, are essentially filters which detect and highlight certain patterns. For each filter, we reduce the computational energy by subsampling. When a feature of interest is detected, the LeNet-5 keeps track of the spatial distance between other features of interest as it plays the role in the classification process, once the data will pass to the fully connected layers and into the gaussian connected output layer. This approach offered a generic solution to a large number of problems encountered in pattern recognition (Lecun Y., 1998: 2283), although the main problem is that the depicted problem that it was need to be classified, needed also to be placed centered in the image and be clear, which is not the case for many classification problems.

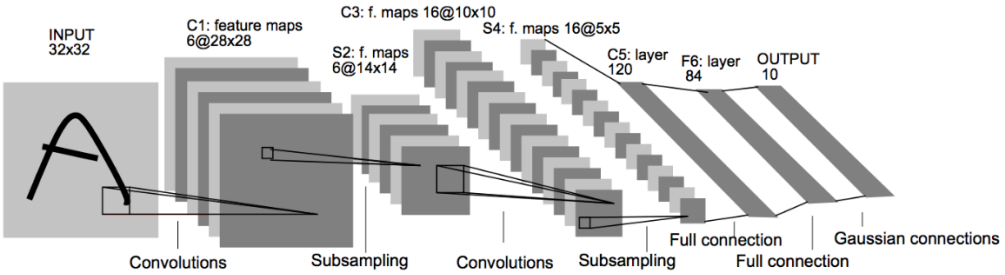

Figure 6. Lenet-5 Network (Lecun Y., 1998: 2283).

2.1.2.2 AlexNet

AlexNet was developed by Alex Krizhevsky et al. in 2012. It is similar to LeNet-5 but enriched with more convolutional layers, as it is shown in figure 25. The purpose of this enriched version of LeNet-5 was to deal with the ImageNet, which it consists of over 15 million images belonging to almost 22000 categories (Krizhevsky A., 2017: 85). With his work, he managed to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 competition.

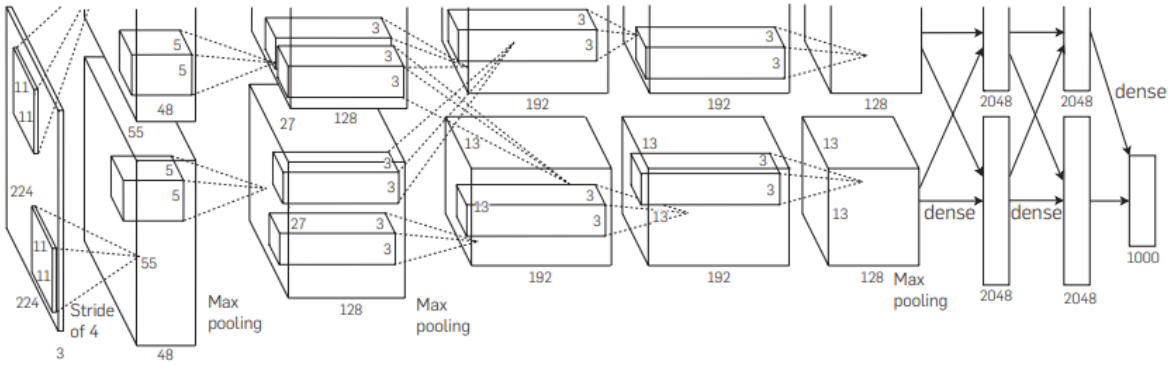

Figure 7. The AlexNet (Khan., 2020: 8).

2.1.2.3 Visual Geometry Group Networks (VGG)

Alex Krizhevsky's inspired many researchers to get involved with deep convolutional neural networks. Although these systems were suitable for large datasets, small datasets were a better fit with shallow networks. Shuying Liu et al (Liu S., 2015: 731), managed to develop the very deep network, also known as visual geometry group network (VGG). It was developed in 2014, consists of deeper and simpler variation of convolutional structures. The goal was to make a universal network that could handle easily small dataset as it can handle large. The adoption of a deep model to a small dataset was made through batch normalization and strong dropout setting.

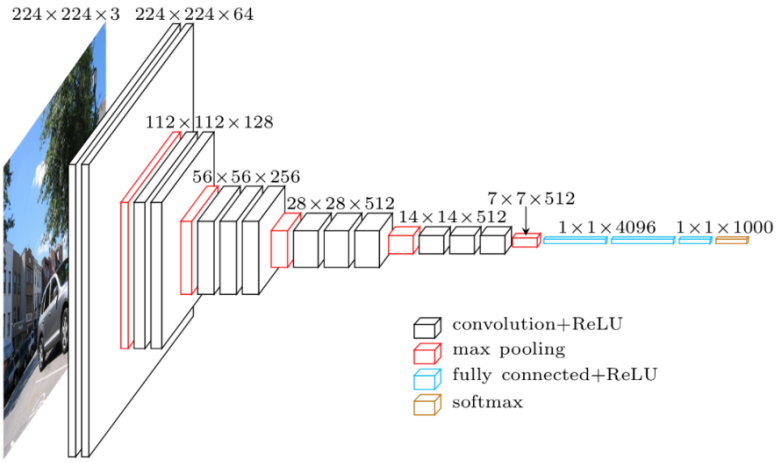

Figure 8. An abstract example of the VGG Network (van Staden., 2021: 3).

2.1.2.4 GoogLeNet

Created by Google researchers in 2014, GoogLeNet (also known as inception) is a 22-layer deep network with an architecture which is built to keep the computational burden constant while increasing the depth and width of the layers (Szegedy C., 2015: 4). It is the first model that introduces the inception module, which uses image distortion, batch normalization and RMSProp, a gradient-based optimization technique.

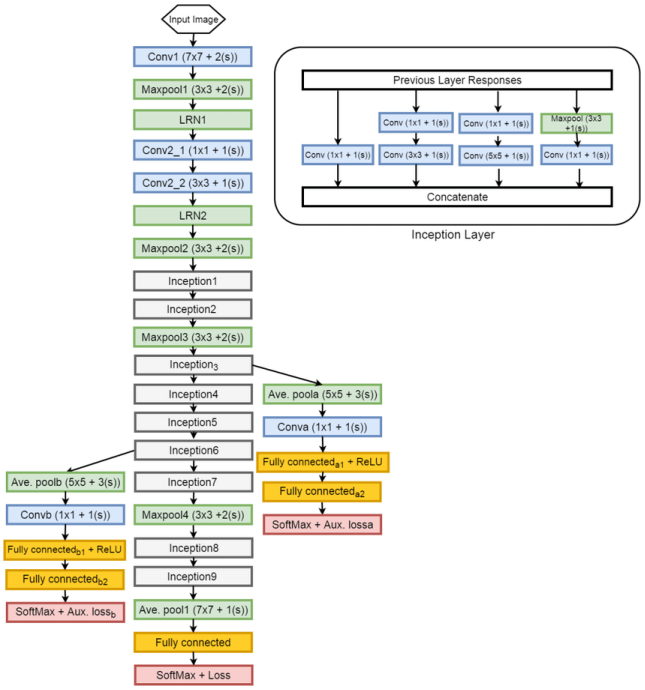

Figure 9. The GoogLeNet (Karri et al., 2017: 584).

2.1.2.5 ResNet

ResNet is a generally accepted principle of deep networks that are capable of learning more complex functions and representations of the input and generally lead to better performance. However, many researchers observed that adding more layers eventually had a negative effect on the final performance.

2.1.2.6 ResNeXt

ResNeXt is an extension of ResNet. It replaces the standard residual layer with one that leverages a "split-transform-merge" strategy used in the Inception Networks. In other words, instead of performing convolutions over

the full input feature map, the block's input is projected into a series of lower (channel) dimensional representations of which we separately apply a few convolutional filters before merging the results. The ResNeXt architecture simply mimics the ResNet models, replacing the ResNet blocks for the ResNeXt block.

2.1.2.7 DenseNet

In DenseNet, each layer's feature map is concatenated to the input of every successive layer within a dense block. Due to the network's capability of direct data usage from any previous feature map, we can work with very small output channel depths. This allows later layers within the network to directly leverage the features from earlier layers, encouraging feature reuse within the network. Generally, this also mimics the general ResNet model architecture, but when compared to each other, DenseNets are reported to achieve better performance with less complexity (Jordan J., 2018: 1).

2.2 Explainability Techniques

Explainability is a topic that challenges the field of deep neural networks for many years. Deep learning models are the most usual examples when we refer to black box models. As until recently, it was difficult to comprehend the inner workings and what drives the model to make a correct prediction. As Machine Learning techniques are becoming popular and essential for a variety of applications, it is reasonable for one to have trust issues against a black box algorithm, especially in critical applications where irreversible errors are not acceptable.

Notwithstanding, there is still much progress to be made as there are always going to be limitations such as energy consumption, small project time frames, and performance limited hardware. From the Tesla Autopilot system and general automotive applications, users should have zero tolerance for mistakes, and a robust behaviour from the whole autonomous system that its decisions should not lead to a social dilemma, as Jean-François Bonnefon and his colleagues issued in his article regarding Moral Machines problem (Bonnefon J.F., 2016: 2). Systems that diagnose diseases, such as Parkinson, should also be as robust as can be, as a wrong diagnosis could lead to great trouble. In order for black box algorithms to be allowed to assist us in everyday life and in critical decision making, we need to understand how they reach a conclusion, and how this conclusion can be explained in terms of logical reasoning.

In general, besides explainability, there are additional aspects that could make an artificial intelligent system behave more like an assistant, than a tool. We have to take into account human-like-ness, personalization and the decision making. The aforementioned aspects differ in importance, with regards to the application. For example, in image processing we demand from the system to point the parts of the image that helped it to reach a prediction, argumentation and human likeness are not relevant here. But before trying to solve this problem of explainability to deep neural networks, we should observe how our brain operates, in order to better understand the task ahead of us. Image processing with deep neural networks are the equivalent of attention and perception in humans.

2.2.1 Attention as a Function Mechanism in Humans

From the founding of Cognitive Psychology, many have tried to give attention a formal term. William James (1890) stated, "Attention is the taking possession by the mind, in clear and vivid form, of one out of what seem several simultaneously possible objects or trains of thought. Focalization, concentration, and consciousness are of its essence". On the contrary, they were many that stated the opposite, like Harold Pashler who assumed that "No one knows what attention is, and that there may even not be an "it" there to be known about" (Pashler H, 1998: 42). Attention remains an examination area of great importance in many scientific fields, such as cognitive psychology, cognitive neuroscience, and even artificial intelligence. The complexity of the relationship along with consciousness is high enough that, although it is subjected to investigation by psychology from the latter part of the 19th century, it is still affecting a vast majority of fields, such as those related to consciousness, mental health, and artificial intelligence. Philosophy was also involved and challenged to understand the concept and process of attention, as it is related with various philosophically perplexed phenomena. Thus, a vast number of discoveries are credited to philosophers (Stanford Encyclopaedia of Philosophy, 2017: 1).

There are various types of attention, but nowadays there is a debate in two categories; is the attention space-based or object-based? Which is more important and with regards to which task? Space-based attention could be interpreted as attention to a certain location in space, in order to process stimuli from this specific spatial location.

It is perhaps the most widely researched domain of attention as it is a crucial mechanism in order to deal with the environment. It is well established that the information eyes can perceive is so much more than the visual system can process, therefore, a bottleneck effect occurs (Soto D., 2004: 69). To surpass this, attention mechanisms have been developed in order to focus on relevant information, and ignore the rest.

For spatial attention they have been suggested various metaphors for how attention can be assigned to one or more locations in space, even without eye movement. Metaphors such as spotlight attention, gradient attention, attention gating and gradient filter. Spotlight attention suggests that attention moves around through space like a spotlight. It also can zoom in and out like a lens and can be narrowed or spread. This metaphor has been widely supported, as it has been criticized (Cheal M., 1994: 699). Gradient metaphor suggests that attention is distributed in a gradient around the cued location, or it may form an irregular field with multiple peaks of different heights. Attention Gating, which is based in rapid serial visual presentation paradigm but with two visual character streams, suggests that when eyes are fixated between two streams, attention will be distributed in serial mode (firstly the one stream, then the other). Gradient filter is a combination of the gradient metaphor, and attention gating which helps to understand spatial cueing tasks which may require simultaneous operations at different locations (Cheal M., 1994: 699).

Despite the fact that space has a predominant role in attention and a vast number of studies reviewed this concept, from the early 1980s, there was a certainty that this type of attention was not the only one in visual processing. Object-based attention was introduced, which, as the term states, supports the view that attention is focused on objects or perceptual groups of the visual scene. To define an object, visual perception is difficult, due to the fact it is not only characterized by its physical properties, but also how we perceive it. One useful definition of an object could be the elements in the visual scene organized by one or more Gestalt grouping principles and/or uniform connectedness (Chen Z., 2012: 784). Many theories have been built around object-based attention, such as distribution of attention, orienting, distractors, memory and inhibition of return. Many studies took place in order to compare these two mechanisms. In 2003 David Soto and Manuel J. Blanco, experimented whether space and object-based attention act in an interactive way or not. Results supported that the two procedures acted interactively, as they found the effect of object cueing only when spatial cues were invalid, but not when they were valid (Soto D., 2004: 69). When the target is presented at a cued location, an object-cueing effect may not be observed because there is sufficient activation to facilitate performance from space-based cueing alone. Alternatively, the interaction can be attributed to location cueing operating more rapidly than object cueing. When the target was presented at a valid location, spatial attention may facilitate a fast and accurate detection even before object features are processed sufficiently to facilitate the response.

On the contrary, Posner experiments showed the important role of spatial attention mechanisms (Posner MI, 1980: 143). The efficiency of detection depends whether the subject knows where in space a signal will occur. Despite the fact that the experiments focused on detection, results showed that in order to locate the signal, one needs to focus in a region of space. Posner suggested that attention can be likened to a spotlight that enhances the efficiency of detection of events within its beam. Another study provided evidence that spatial attention is used for perception and action. Input and output spatial attention are parts of Simon-like effects (Israel M., 2016: 255). It is very clear that spatial attention plays an important role in the way one perceives the environment, when at the same time needs meticulous investigation on its mechanics. As studies are being done whether attention is based on the one or the latter, it becomes clearer that these mechanisms both exist and cooperate in order to define the world around us. Spatial attention is responsible for the location of the region of space where the point of interest is located. Object based attention is responsible for procedures like orientation. Object based attention does exist, and object-based effects result from a combination of configural and probabilistic prioritizations, although it is not as prevalent and robust as it may be assumed (Pilz K., 2012: 1, Shomstein S., 2008: 132).

2.2.2 Perception as a Function Mechanism in Humans

Attention is only one part of the procedure of explaining a visual stimulus. Perception plays also an important role in visual comprehension, and in fact, this is what explainability algorithms want to achieve when they are integrated in deep learning models. The collaborative process between relation and attention helped human species to become what they are now. One is useless without the other; without the ability to attend to information we could not determine the way we should perceive it, and without sensation there would be nothing to attend to. Thus, the ability to perceive and attend to information is of utmost importance, as that allows us to interact with the environment. In humans, cognitive resources are centered in potential danger or mate prosperity automatically,

although how perceived stimuli must be examined in the context of its evolutionary function in order to determine how it affects the human cognitive system (Wilck A., 2019: 1).

The mechanism of perception identifies, interprets and organizes the sensory information in order to depict and understand the environment. This function involves the electrical signals from the nervous system, which are generated from the physical stimuli of the organs. There are various types of perception, such as sound, touch, taste, smell, social perception (e.g., recognition of a face or recognition of facial expressions), multi-modal perception (e.g., chronoception and familiarity), and senses like balance, acceleration, gravity and pain. The complexity of the various systems of perception is implied in the fact that perception is not a passive process, but an active process of receiving, analysing, and discriminating features of objects without direct contact with them (Coello Y., 2005: 39).

Attention and perception not only depend on each other, but also affect each other's performance. For example, an observer has the ability to detect something more quickly and accurately, if the information of its location is presented in the first place. This allows to ignore, or sometimes not to detect at all, irrelevant information, and focus the perceived mechanisms to the anticipated point of interest. This is called inattentional blindness and is presented when the observer carries out a demanding task which could be enough to ignore irrelevant information that is shown at the same time. Thus, the observer fails to attend to irrelevant information. If the irrelevant stimulus is not expected, this can last for several seconds. The experiment of Borojevic and Gvozdenovic investigated the possibility of perception without visual attention. Results showed that attention is necessary for the adequate integration of perceptual functions (Gvozdenovic V., 2013: 3). Furthermore, it has been reported that an observer can perceive a target much faster when an asymmetry exists. This has been explained by pooling of the signal in the spotlight. When the target has a specific property, it can be perceived more easily. Since accuracy is determined by the ratio of signal to noise, a larger spotlight—which collects more noise—can be used while keeping accuracy the same. Since a larger spotlight requires fewer shifts to cover an area, the result is faster search. (Treisman, A., 1998: 15, Reisberg D., 2013:1).

Various functions of attention affect perception, such as division of attention. Division of attention affects the temporal process of perception. Researchers in 2020 came to the conclusion that, by increasing the points of interest the detection rates are divided. These divisive effects were essentially constant over time and over the time-varying influence of the target signals and response tasks (Lappin, J., 2020: 3). Reports and references above show that attention and perception is related in various ways, especially in visual representation. It is important to mention that one affects the other. Inattentional blindness which takes place when demanding tasks are carried out, and change blindness which rejects perception changes during eye movement, are phenomena that essentially help the brain to deal with the environment in a more efficient way. Although the relation between attention and perception is complex, it is possible to clarify and detect the effects taking place from one to another. All the above show how important attention and perception are, in order to perceive, understand, and interact with the world.

2.2.3 Explainability Techniques in Deep Learning

We defined the importance of Attention and Perception and how they related to the deep learning models. But deep learning models lack one key element, explainability. In the following lines, we analyse some of the contributions that researchers have made in order to fill the gap of the core concepts between the deep learning methodologies and explainability.

2.2.3.1 A Brief Overview of Explainability Techniques in Deep Learning Systems

In general, there are two types of explainability, intrinsic and post hoc explainability. Intrinsic explainability focuses on the creation of explainable systems. Such systems are considered simple in structure, as complexity may cause issues in providing explainable results. Intrinsic explainability methods refer to short decision trees or sparse linear models (Molnar, 2022: 17). Intrinsic explainable models are considered as model-specific. Intrinsic models can be also developed with rule-based methods.

Rule-based systems represent a way for codifying the problem-solving ability of human specialists. Such specialists tend to precise most of their problem-solving techniques in terms of a group of situation-action rules, and this means that rule-based systems ought to be the tactic of alternative for building information-intensive knowledgeable systems. Though many various techniques have emerged for organizing collections of rules into automatic

specialists, all RBSs share some key properties. First, they are able to incorporate sensible human information in conditional if-then rules, and their talent will increase at a rate proportional to the enlargement of their information bases. Furthermore, they'll solve a good vary of presumably complicated issues by choosing relevant rules, and by combining the ends up in acceptable ways that, they adaptively verify the most effective sequence of rules to execute, and that they make a case for their conclusions by retracing their actual lines of reasoning and translating the logic of every rule utilized into tongue (Hayes et al., 1985: 1).

Post hoc explainability on machine learning models is an alternative solution for creating explainable systems. Instead on developing purely interpretable models, a second model is developed to interpret the outputs of the black box model. Post-hoc models are considered agnostic and they can be used in various AI systems. Need for such solutions may vary, and can be found in issues regarding regulatory and standard compliance, ethical principles or domain needs. XAI systems are a great solution on all that by providing answer which explain the outcome. There are two main categories of post-hoc explanations. Local explanations are used to provide individual explanations and trust upon datapoints. This type of post-hoc explanations, consist of various methods which we briefly mentioned below

Saliency maps are mostly utilized in image processing, used to show subregions that played the most important role in a classification. Examples are grad-weighted Class Activation Mapping (Selvaraju R., 2017: 621), and Structured Attention Graphs (Shitole V, 2020: 1). Rule-based explanations provide explanations based of logic and rules, usually expressed in the form of IF-THEN statements. It is considered as constructive explanation style method (Waa et al., 2021:3) Example-based explanations provide specific instances of a dataset in order to explain the outcome of the AI model (Molnar, 2022: 113). Such types of explanation are considered model-agnostic, as they are able to interpret any machine learning model.

Global explanations provide details regarding the process of decision for an outcome, meaning that it tries to explain the mechanism and the argumentation process of the system. It includes methods such as model distillation, summaries of counterfactuals and many more. Model distillation method refers to the transfer of knowledge between machine learning models (Hinton et al., 2015: 1). Counterfactual explanations refer to statements of a hypothetical reality that opposes the observed facts. An example of this statement could be: “If X had not occurred, Y would not have occurred” (Molnar, 2022: 182). In other words, the system provides contrastive explanations which form a hypothetical reality and outcomes.

According to Aniek F. Markus et al., ‘An AI system is explainable if the task model is intrinsically interpretable (here the AI system is the task model) or if the non- interpretable task model is complemented with an interpretable and faithful explanation (here the AI system also contains a post-hoc explanation)’ (Markus A., 2021: 9). In other words, explainability is the concept that an outcome or a conclusion can be explained in a manner that makes sense to the user, in order to be accepted as valid. We have already mentioned that, personalization tries to recognize the standards of user’s demand for a sound and acceptable outcome. Through explainability, personalization is utilized to build and format the outcome according to these standards. In order for the explainable outcome to be accepted, it should be transparent regarding its argumentation.

Explainability can break down into two categories, interpretability and fidelity. Interpretability is the possibility of something to be understandable and comprehensible to a human, and again, this has to do with the knowledge and experience of the user to a current subject. Interpretability is consisted of two generic underlying factors, clarity which refers to the provision of a rationale from the system that is similar for similar instances, and parsimony, for when the explanation is not too complex and can be presented in an interpretable form. Fidelity on the other hand, the property of describing the cognitive system in an accurate way, is tightly related to transparency. This category can be characterized by completeness and soundness. Completeness is about providing a sufficient amount of information in order to compute the output for a given input. Soundness describes the ability of the cognitive system to provide a sound and truthful explanation. These sub-properties can be depicted in detail in figure 5 and helps to grasp a better understanding regarding this property.

In recent years, researchers developed various techniques which can identify the reasons behind the decision of a deep learning model. These techniques inspire and pave the way for new methods to be developed. class activation mapping (Zhou B., 2016: 2922), or CAM for short, is designed to produce heatmaps to highlight the region of the image, which the deep learning model focused on in order to classify it. The visualization of where the model is looking, helps to identify whether the model is trustful or not. For example, a deep learning model could classify a

train image, not by looking at the actual train, but by looking at the train tracks. Despite the correct classification, the model might take into account wrong parts of the image, which could be a consequence of poor training, or other factors.

A more advanced technique which is based on class activation mapping, is Grad-CAM (Selvaraju R., 2017: 621). This technique is considered class-specific, meaning that for the same image, it can produce a separate visualization for each class which is present in an image. Another interesting approach is structured attention graphs (Shitole V, 2020: 6). This method is inspired from attention maps, which are popular tools for explaining the decision of deep learning models. The researchers argue that just one attention map is not enough, but with SAG, we can have a set of attention maps and we can capture the confidence level of the model on each one and how the classifier is impacted.

2.2.3.2 Class Activation Mapping

Class activation mapping (CAM), by Zhou et al. (Zhou B., 2016, 2921), is a technique that produces heatmaps to an image, in order to highlight the class specific regions of it. CAM provides accurate localization ability. Before, class activation mapping was used as localization function, it was proposed for training regulation. This technique helps us to understand if the correct classification of the image is honest. In their research, they have shown that convolutional neural networks behave as object detectors, despite the fact that there is no supervision on the location of the depicted object (Zhou, B, 2014: 7). Zhou and his team have found that the usage of global average pooling acts as a structural regularization function, which prevents overfitting.

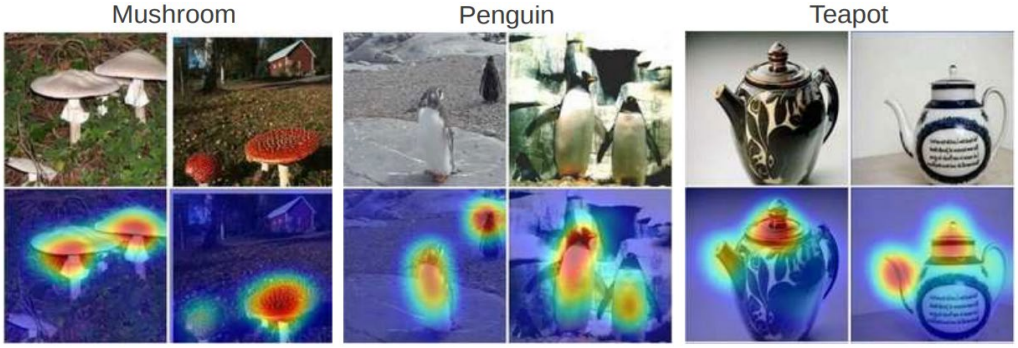

Figure 10. Class Activation Mapping highlights the class specific regions of the image (Zhou B., 2016, 2921).

The honesty of a cognitive system is critical, because one of the fundamental pillars of a cognitive system as an assistant is trust. For example, suppose the convolutional neural network of a cognitive system analyses X-ray images to detect an illness. If the patient is very weak, he/she may need to lie down on a metal bed. Statistics suggest that a weak human could be sick, thus the majority of people lying when they take X-rays are having the illness that is to be diagnosed. The problem here begins when the convolutional neural network starts to observe patterns of the background, such as a specific part of the bed, with the result of the correct prediction but with a false pattern taken into account (Zech J.R., 2018: 3). Class activation mapping helps to identify easily the discriminative regions of the image with a single forward pass.

The CAM procedure takes place in the second to the last convolutional neural network layer. In this point of the convolutional neural network, we are able to identify the important parts of the image by projecting back the weights of the output layer onto the convolutional feature maps. Then, with these weights which we derived from the global average pooling, we construct the spatial average of the feature map. We use global average pooling in order to construct the map. One could argue that we should use global max pooling instead. Zhou et al. have shown that this technique is capable of recognizing patterns and give sufficient explanation by pointing to the parts of the classification of nearly every convolutional neural network, such as VGG, GoogLeNet, and AlexNet. This adaptability to every network, allows us to evaluate the performance of every network, even if it is custom trained or not, with regards to the dataset that one may have to work on, and this allows us to choose the best performing model for every classification prediction problem. The authors of the paper state that there is a big difference between GAP (global average pooling) and GMP (global max pooling), as the GAP loss encourages to identify the extent of the object and helps to find all the discriminative parts of the object, while GMP helps to identify one discriminative part.

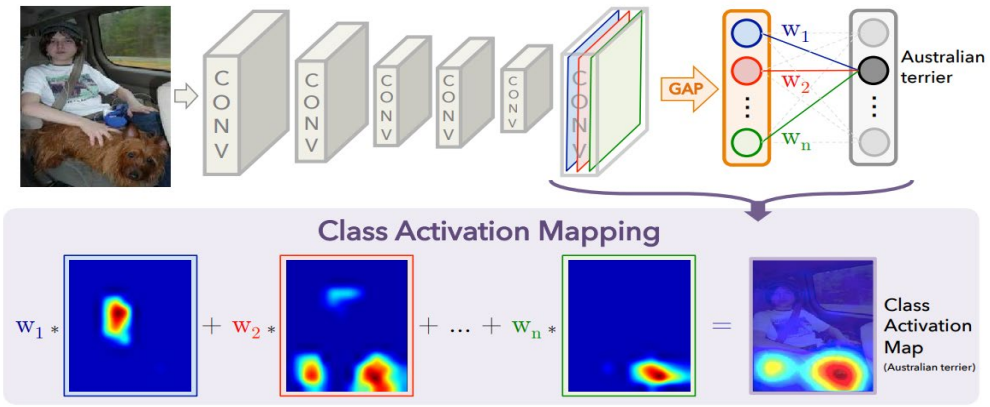

Figure 11. The Class Activation Mapping highlights the class-specific discriminative regions (Zhou B., 2016, 2922).

Results of the research have shown that the class activation mapping approach does not negatively affect the classification performance more than 1% to 2% when removing the additional layers from the various networks, but it is compensated by adding two convolutional layers just before GAP resulting in a new modified network. Overall, the classification performance is largely preserved for GAP networks.

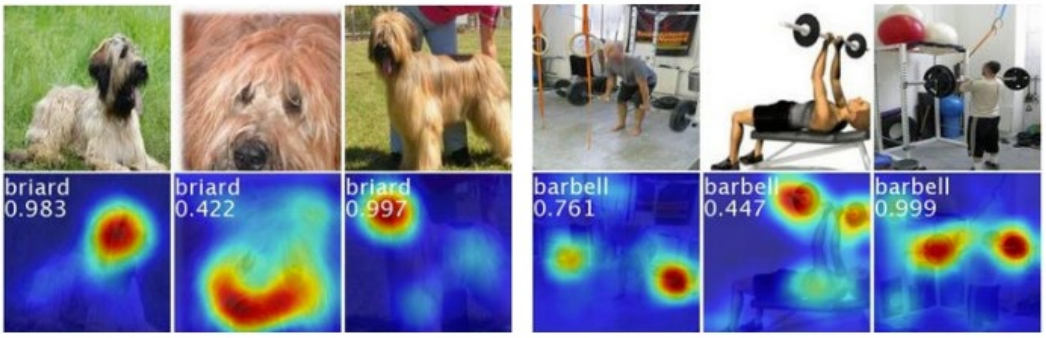

Figure 12. The CAMs of two classes from ILSVRC (Russakovsky O., 2015: 211). The maps highlight the discriminative image regions used for image classification, the head of the animal and the plates in barbell (Zhou B., 2016, 2921).

2.2.3.3 Gradient-weighted Class Activation Mapping (Grad-CAM)

Gradient-weighted Class Activation Mapping (Grad-CAM) is also a popular technique to visualize where the convolutional neural network draws its attention for the classification of an image (Selvaraju R. R., 2016: 4). Grad-CAM is known for its class specification capability, meaning that it can construct a different visualization for every class that exists in the image. It is considered weakly-supervised, as it is able determine the location of particular objects by using a model trained upon whole image labels instead of explicit location annotations. Also, it is considered and used for weakly-supervised segmentation, where the model predicts all the pixels that belong to the specific object without requiring pixel-level labels for training. Finally, grad-CAM is able to offer better understanding of a model, for example by providing insight into model failures.

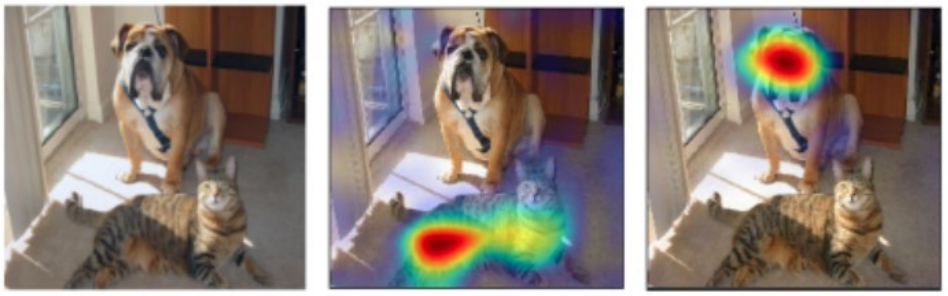

Figure 13. Example of the class-specific technique, grad-CAM (Selvaraju R. R., 2016: 4).

Grad-CAM does not require a specific convolutional neural network model. We ought to use the spatial information that is preserved through the convolutional layers, in order to identify the important parts of the image that determine the classification prediction. The difference between CAM and grad-CAM lies in the inner workings of the latter, where there we weight the feature maps using values that are calculated based on gradients. Although this is a very convenient technique for multi classification on an image, research has shown that there is a fundamental problem with grad-CAM as sometimes Grad-CAM highlights regions the model did not actually use. Thus, now we are going to analyse the successor of grad-CAM which solves that problem (Draelos, R. L/, 2021: 2).

2.2.3.4 High-Resolution Class Activation Mapping

High-resolution class activation mapping, also known as HiResCAM, is argued to solve the problem of erroneous highlighting in grad-CAM (Draelos, R. L., 2021: 2). It is often the case that deep learning models might rely on spurious correlations, for example the prediction of pneumonia based on a distinct metal piece of the bed (Zech J.R. 2018:3). As popular as gradient-based visual explanations might be, they often produce non-class-specific outcomes due to white-noise appearance. Due to the gradient averaging step, grad-CAM cannot guarantee the exact spatial information that is used for model prediction. HiResCAM, on the other hand, claims to solve this problem. HiResCAM can be calculated to any convolutional layer of the CNN, although, in order to guarantee the properties of explanation it must be applied to the last convolutional layer just like the previously mentioned techniques.

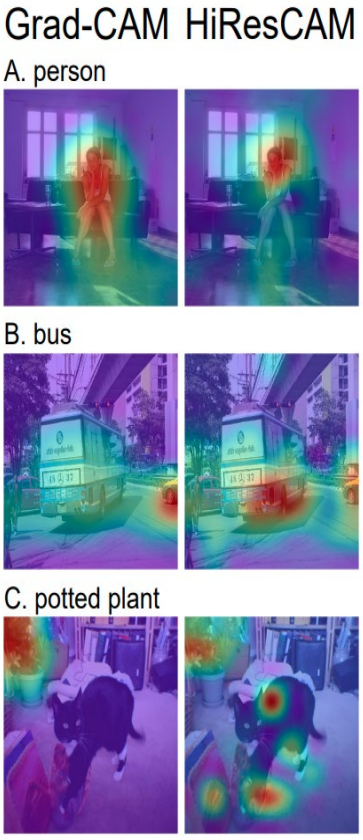

Figure 14. Grad-CAM vs HiResCAM (Draelos, R. L., 2021: 2).

2.2.3.5 Structured Attention Graphs

This technique was primarily chosen to be analysed for this thesis. Shitole et al (Shitole V, 2020: 6) argue that a single saliency map that is produced by the aforementioned techniques is not enough, as it provides an incomplete understanding, since there are often many other maps that can explain a classification equally well. Additionally, it is needed to know what part of the image played a bigger role in the classification process. This is done by using discrete search algorithm to find many high-confidence attention maps that are distinct in their coverage, instead of using the common grad-based optimization approaches. Multiple attention maps can show the CNN prediction more comprehensively with a logical structure in the form of a monotone disjunctive normal form (MDNF). However, this poses the challenge that these maps have to be shown with a proper visualization to help users gain a more comprehensive mental model of the CNN. This can be done with structured attention graphs, that basically are directed acyclic graphs in which each node shows an attention map of different image regions. This visualization technique allows the user to view what led to a specific prediction and how big was its role to the image classification in terms of percentage. This is done by decomposing local maps into subregions and making the common and distinct structures across maps explicitly.

(a) Original Image | (b) Grad-CAM heatmap | (c) I-GOS heatmap | (d) Region 1 (98% confidence) | (e) Region 2 (97% confidence) | (f) Region 3 (93% confidence)

Figure 15. Conventional explainability heatmaps compared to SAG diverse regions of the image (Shitole V, 2020:2).

The procedure starts by dividing the image into 49 sub images, 7x7 patches, due to the fact combinatorial search might be computationally expensive and unfeasible, if we treat every pixel as patch, thus we divide the image into a coarser set of non-overlapping patches. The number of explanations might be combinatorial, and instead of using a heatmap algorithm, search algorithms are utilized in order to check the convolutional neural network predictions on many combinations of patches and determine if they are able to explain the prediction of the CNN by being a sufficient minimum explanation. If the model is able to achieve the same confidence from the sub-image as well as the image, the rest of the image may not add anything useful to the image classification process. Thus, the objective is to find the minimal sufficient explanation (MSE) (1) that scores higher than a threshold where no proper sub-regions exceed the threshold, for some high probability threshold Ph. According to Shitole *"A central claim of the paper we purport to prove is that the MSEs are not unique, and can be found by systematic search in the space of subregions of the image. The search objective is to find the minimal sufficient explanations Ni that score higher than a threshold where no proper sub-regions exceed the threshold"* (Shitole V, 2020: 6).

$$f_c\,(N_i\,) \;\geq\; P_h\, f_c\,(x), \max_{n_j \subset N_i} f_c(n_j) <\; P_h f_c\,(x) \quad (1)$$

Where:

- c: a class
- $f_c\,(x)$: The output of class-conditional probability on class $c \in C$ for a specific input x
- N_i : The set of minimal sufficient explanations
- P_h: The probability threshold we set for the MSE against the entire image

So, we down-sample the resolution of the image into 7x7, where each pixel corresponds to a coarser patch of the image and the search is computationally less expensive, and then we use an attention map as a heuristic for searching the down-sampled images. Afterwards, we perform average pooling on M, with regards to each patch, in order to get an attention value for each patch, and produce a coarser attention map. Once the low-resolution attention map is produced, we use bilinear up-sampling to use it as a mask to the original image resolution. Bilinear up-sampling offers a slightly rounded region for each patch in order to avoid sharp corners that could be erroneously

picked up by the CNN as features. In the paper, two different search methods are analysed. Restricted combinatorial search, where it constrains the size of the MSE to k patches and finds the MSEs N_k by searching for all combinations (conjunctions) of k patches that satisfy the criterion for the equation (1). For this type of search, we need to prune the search space, in order to avoid an expensive combinatorial search from the computational point of view. We select the m most relevant patches, where the relevance of each patch p_j is given by an attention map as $\mathrm{M}(\mathrm{p}_j)$, and then carry out a combinatorial search. The second method, is referred as beam search, in which we search for a set of the top minimal sufficient explanations by maintaining a predefined set of distinct conjunctions of patches as states at the i_{th} iteration. This has a lot in common with the traditional beam search, but here the attention map is leveraged in order to generate the successor states. In each iteration, a new candidate states set is generated. We obtain the classification score for each candidate successor state and we select the w states with the highest score.

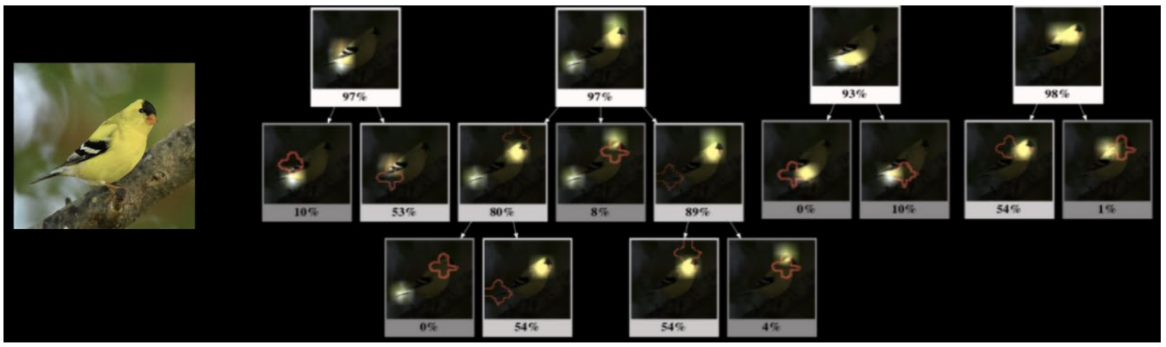

Figure 16. An example of a Structured Attention Graph (Shitole V, 2020: 2).

Now, it is needed to present that data in a readable way, in order to provide the user a clear explanation and build better mental models of the model's behaviour. This can be done with structured attention graphs - SAG. With SAG we are able to visualize how different combinations of the image parts affect the classifier. Structured attention graphs, are directed acyclic graphs in which each node shows an attention map of different image regions, and each node corresponds to the subset relationships between sets defined by the removal of a single patch. The root nodes are the set of the patches that show the minimal sufficient explanations we described before.

The procedure to create a SAG is the following. First, we need to find multiple candidates for minimal sufficient explanations. It is observed that these candidates might often have a large number of similar MSEs, so in order to minimize the cognitive burden on the user, we derive a small subset from this set, by heuristic prune. After we have created this subset, we build the SAG. Every element of the subset forms a root node. Child nodes are generated with the deletion of one patch at a time, from the parent node. At each node we calculate the confidence with a forward pass of the image. The nodes that have a probability lower than 40% are not expanded.

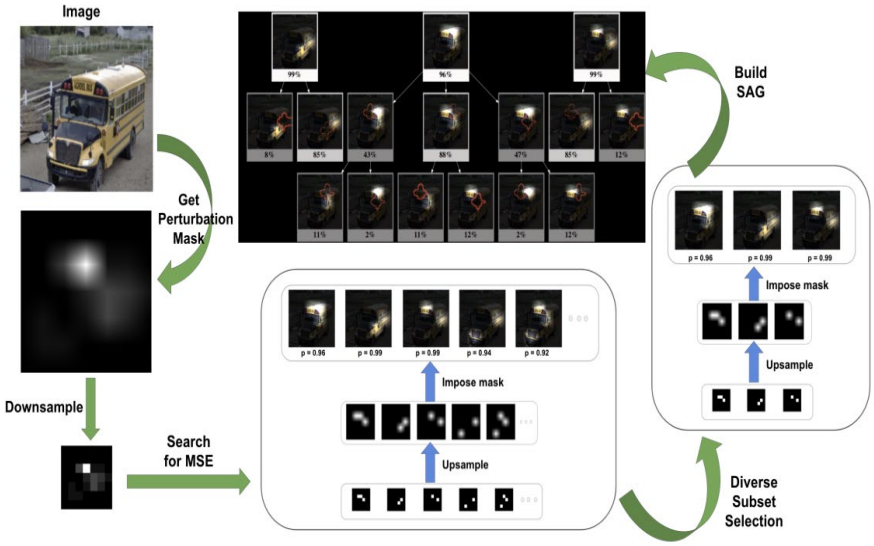

Figure 17. The procedure of generating Structure Attention Graphs (Shitole V, 2020: 2).

The work products that SAG creation algorithm offers are the following.

A folder which contains a comparison between an insertion and a deletion image, for every root, with regards to their confidence levels upon its prediction.

1. A folder with the patch images that construct the Structured Attention Graph.
2. A text file with the Monotone Disjunctive Normal Form of the inspected image.
3. A gif file which illustrates the MDNF.
4. The picture with its grid image that was determined in the algorithm.
5. The image perturbed.
6. An image with the perturbation heatmap.
7. The picture with the perturbation heatmap.
8. A dot file with the Structured Attention Graphs.
9. The structured Attention Graph.
10. The final product with the Structured Attention Graph and the Original Image.

The algorithm is accompanied with results that were derived from images of ImageNet (Krizhevsky A., 2017: 85). It is easy to reconstruct the same results by rerunning the algorithm with the same images. Some of them are also mentioned in the related paper (Shitole V, 2020: 6). In order to provide clarity, we have to rerun the algorithm and see what the results are and if they have any difference with the submitted ones. Our results in this case have no difference at all with the submitted and it was trivial to regenerate them. Here, we provide one example out of the many for the paper. The model architecture that is used is VGG19 (Simonyan K., 2015: 5), and for the results the provided by Pytorch pretrained models have been used. The monotone disjunctive normal form for this image is P_{18} & P_{25}| P_{15}& P_{23}| P_{18}& P_{24}. Here the vertical line is the symbol of or sign, and the ampersand symbols the and sign. P_{xy} is a specific patch on the image grid (see figure 42). Notice that the MDNF is depicted separately in the roots, and can be cross check along with the grid, also in image 42. Thus, the SAG is generated as it is shown in figure 43. The generated SAG is relatively shallow, which means that the children's nodes of the image, are weak to provide sufficient information percentage wise, as their children either have a confidence level lower than 40%, which is the lower acceptable threshold, either the root has left with only one patch.

Review of the researcher's results, helped us to have a better understanding of the algorithm from the point of view of a black box. We reran the results, created all the work products that we expected and mentioned before. The generated SAG, showed nodes that one could easily infer that the image label is a school bus. This means that the model is looking for information to the right parts of the image, which is what we expect from a pretrained model.

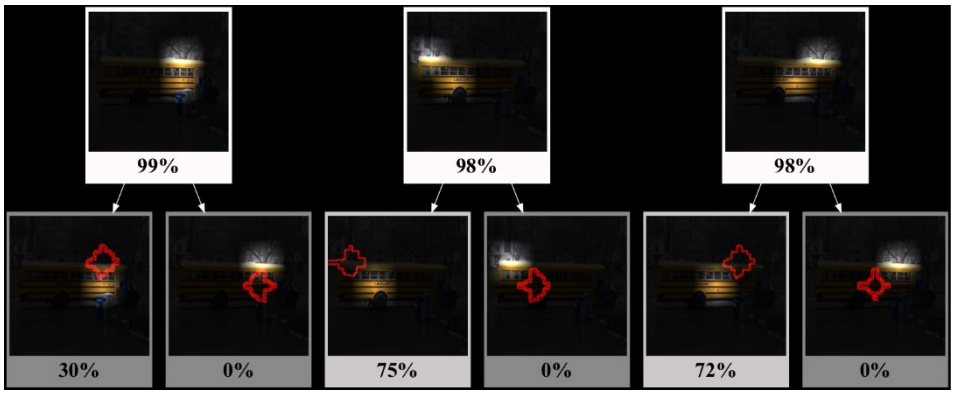

Figure 18. The Structured Attention Image of the prediction.

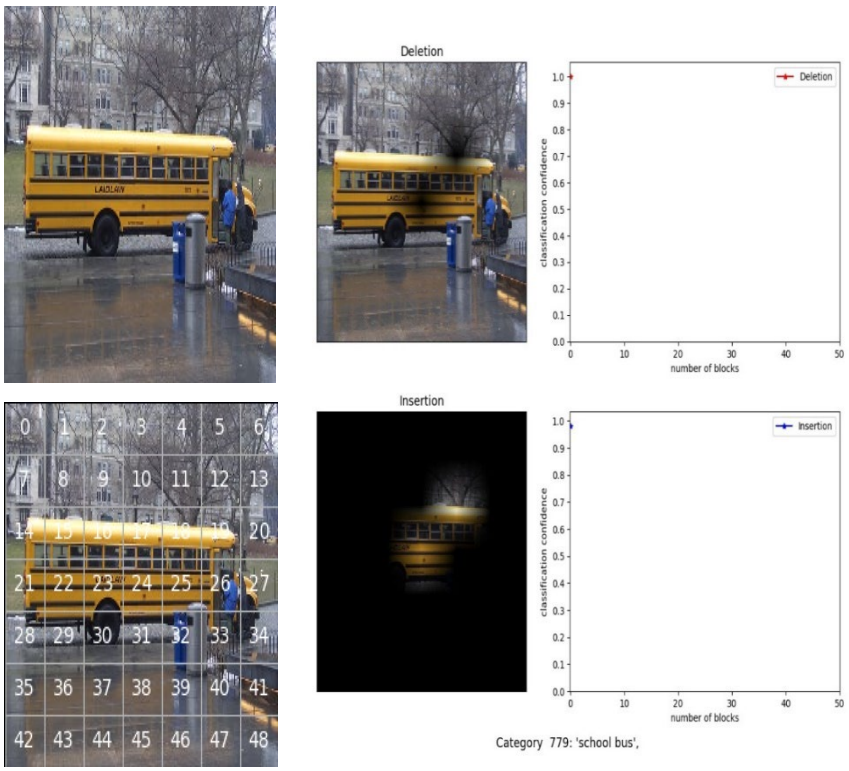

Figure 19. Top left, the perturbated image. Bottom left, the grid image. Right image, The prediction confidence of the deletion and the insertion image.

2.3 Analysis of Parkinson's Disease and its Detection Methods

Parkinson's disease (PD), refers to the chronic and progressive neurodegenerative disease, which can be characterized either by motor and non-motor features (Zesiewicz T., 2006: 1811). This disease was described for the first time in 1817 by Dr. James Parkinson. PD aggressive behavior causes a vast impact, not only to patients, but also to those who take care of them, as it affects and deteriorates the mobility and the control of muscles of the patient (DeMaagd G., 2015: 504). Symptoms of PD are caused due to the deterioration of the striatal dopaminergic neurons, as well as due to the neuronal loss in non-dopaminergic areas. Reports show that PD is the most common neurodegenerative disorder across the globe. Although PD is often related to elder people, patients can show symptoms from the age of 30. In the past, there have been many researches that have analyzed and reviewed the implications of this disease, and many have applied deep learning methodologies for its early detection (Zhao X., 2018: 1147, Kollia I., 2019: 1, Wingate J., 2019: 3). This thesis aims to enrich the collection of the work on this topic, by implementing explainability techniques in deep learning systems which have been trained to detect Parkinson's disease.

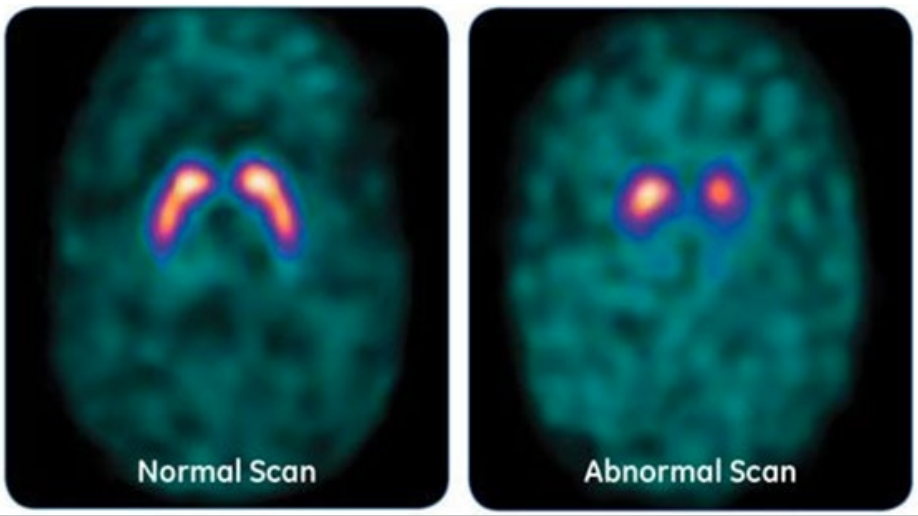

Figure 20. DaTSCAN of a healthy person (left), and a person with signs of Parkinsonian syndrome. (right).

Our problem has to deal with the clinical diagnosis of Parkinson's disease in patients via DaTSCAN images, which are images of the brain after the radioactive tracer Ioflupane $^{123}\mathrm{I}$ have been injected via an intravenous injection of 4–6 mCi scale (Calle et al., 2019: 11). Ioflupane $^{123}\mathrm{I}$ is able to find the way to the brain by attaching itself to the dopamine transporter. Experienced nuclear medicine trained physicians, are able to review DaTSCANs by evaluating the appearance of dopamine transporter concentration in the right anterior putamen, right posterior putamen, left anterior putamen, left posterior putamen, right caudate, and left caudate using a semi-quantitative scale (Calle et al., 2019: 11). According to Marek et al. (Marek K. et al., 2020: 18), abnormal DaTSCAN images fall into at least one of the following three categories (all are considered abnormal).

- Activity is asymmetric, e.g., activity in the region of the putamen of one hemisphere is absent or greatly reduced with respect to the other. Activity is still visible in the caudate nuclei of both hemispheres resulting in a comma or crescent shape in one and a circular or oval focus in the other. There may be reduced activity between at least one striatum and surrounding tissues.

- Activity is absent in the putamen of both hemispheres and confined to the caudate nuclei. Activity is relatively symmetric and forms two roughly circular or oval foci. Activity of one or both is generally reduced.

- Activity is absent in the putamen of both hemispheres and greatly reduced in one or both caudate nuclei. Activity of the striata with respect to the background is reduced.

Details for Parkinsonian disease signs can be examined in detail in Figure 22. In a healthy person we should expect a wider area of luminated striatum, in the shape of commas, which indicates a healthy dopamine system. In a DaTSCAN of a healthy person, we should expect two symmetric-comma (or crescent) shaped focal regions of activity, mirrored about the median plane (Marek K. et al., 2020: 18). Striatal activity is distinct and relative to the surrounding brain tissue. Thus, we expect a demonstration a normal "comma" configuration on the striata bilaterally (Calle et al., 2019: 12). The dataset, as we mentioned before, initially contained 289 of DaTSCAN images that are going to be fed to the network, 170 PD cases, and 119 NPD. The dataset we worked is based on datapoints from the dataset of the research "Machine Learning for Neurodegenerative Disorder Diagnosis — Survey of Practices and Launch of Benchmark Dataset" (Tagaris A. et al., 2018:1). Now that we have all the appropriate modules, we are able to apply the explainability techniques that Shitole et al, (Shitole V, 2020: 6) argued in their paper.

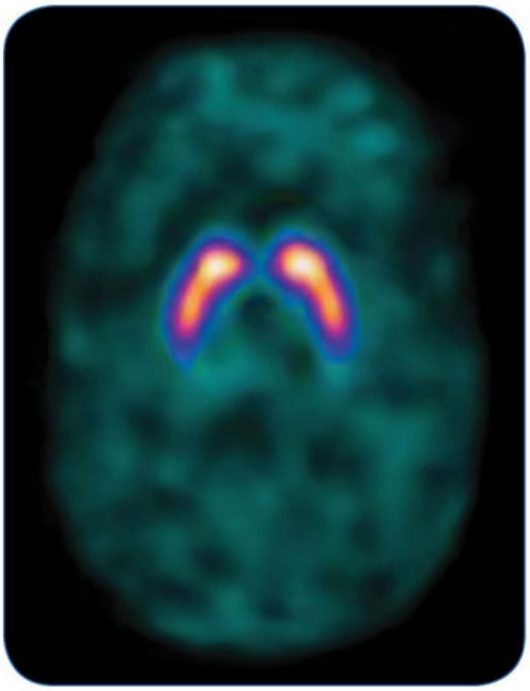

Figure 21. In transaxial images, normal images are characterized by two symmetric comma- or crescent-shaped focal regions of activity mirrored about the median plane. Striatal activity is distinct, relative to surrounding brain tissue (Marek K. et al., 2020: 18).

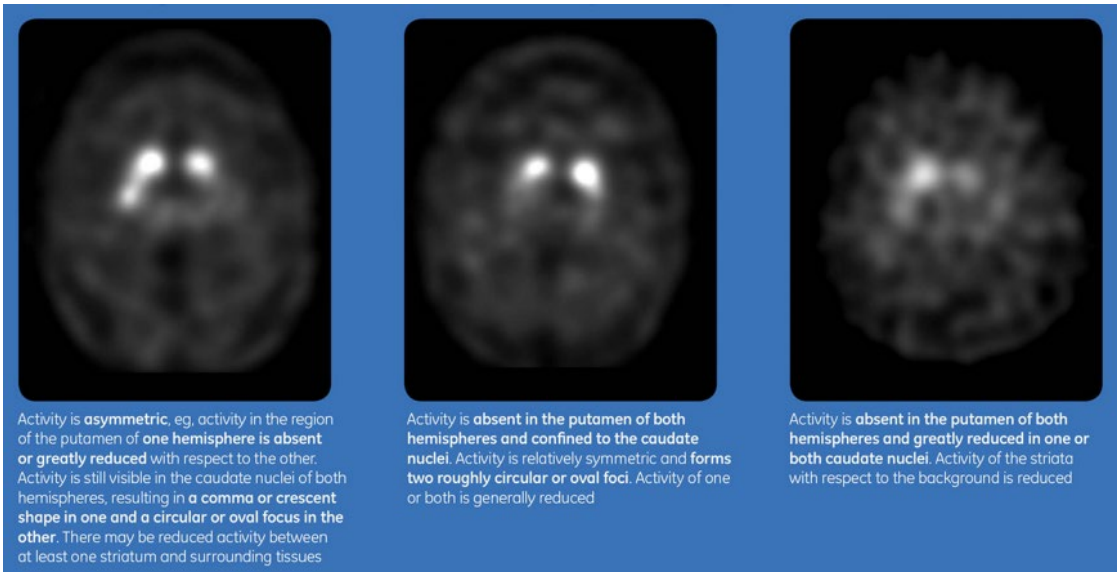

Figure 22. Examples of abnormal DaTSCAN images. Abnormal images fall in at least of one of the three depicted categories (all are considered abnormal) (Marek K. et al., 2020: 18).

2.4 Final Thoughts on Explainability

Structured attention graphs seem to be a powerful tool in order to explain the behaviour of a deep neural network. Explainability techniques could lead to new breakthroughs and commercial use applications that are using deep learning models. Cognitive systems are needed to be understandable when they make a decision, propose something or argue with the user. This could bring us closer to having such models more as assistants than tools. In the next chapter we will apply Structured Attention Graphs to a real-life problem. This thesis argues that structured attention graphs could help in the medical field, and especially on medical diagnosis. We will now create the model, and then we will integrate it in the SAG code. Afterwards, we will analyse and evaluate the results, in order to determine if the algorithm is capable of being implemented in such a critical field.

Chapter 3
Application of An Explainable Deep Learning System

In this chapter we will show the process of development of the various convolutional neural networks we used to experiment and will explain step by step the procedure. These models will be used in order to predict the datapoint of the DaTSCAN dataset, and also will be used in order to apply the explainability techniques. Then we will apply explainability techniques upon these models, and we will analyze and review the results.

3.1 Development of Experimental Convolutional Neural Networks

3.1.1 Environment Specifications

Deep learning models are "hungry" algorithms and require a generous amount of processing power. The necessary hardware environment may depend on the task. For our application the available hardware setup contains the following specifications.

- Operating Systems: Ubuntu 22.04 LTS
- Processor: Intel(R) Core (TM) i5-8600K CPU @ 3.60GHz 3.60 GHz
- Graphics card: NVIDIA GeForce GTX 1060 6GB
- RAM: 32GB

Regarding the software environment, the algorithm is written in Python 3, the machine learning framework we use for this application is Pytorch, an open-source machine learning framework that accelerates the path from research prototyping to production deployment. Since we have in the system a NVIDIA graphics card, the computation platform which is used in order to accelerate the computation of the training.

1. Language: 3.10.7
2. Pytorch Build: 1.12.1
3. Package: Conda
4. Computation Platform: CUDA 10.2

3.1.2 Algorithmic Steps to Create a Model

The steps of the algorithm for the creation of the models would be the following. The model is developed in Python 3.

1. Define the hyperparameters of the CNN (config.py).
2. Create and prepare the dataset in order to be model-compatible (dataset.py).
3. Create, configure and modify the model properly (model.py).
4. Train the model (train.py).
5. Save the model (save.py).
6. Create and save the model statistics in a report (metrics.py).
7. Upload model metrics to Tensorboard and create the appropriate URL (main.py).

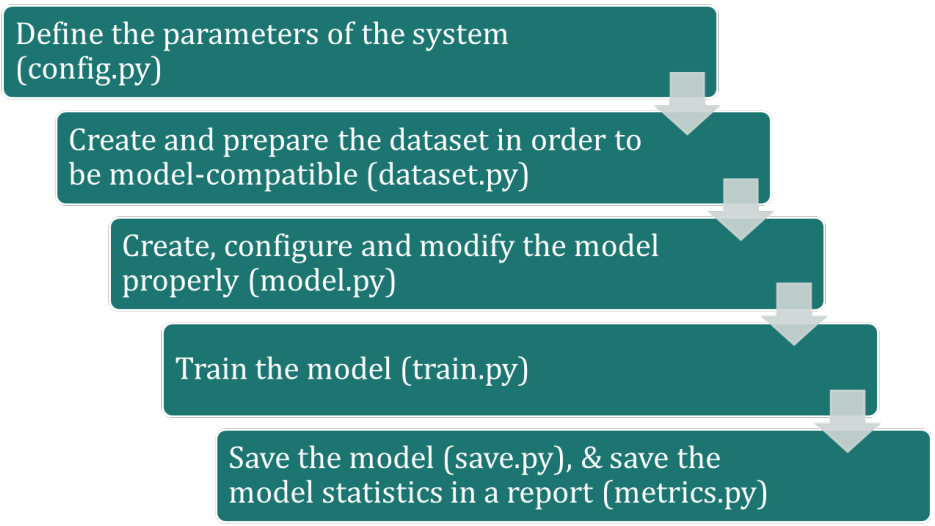

Figure 23. The flowchart of creating a neural network.

The steps are orchestrated in main.py. The code can be seen in detail in Appendix A, also it is available in the following link https://github.com/karvo/ouc-cogsys-cnn

3.1.3 Setup of Initial Parameters

The first thing that is needed to do, is to define the initial parameters of the model. These parameters play crucial role to the model's performance as they define the core aspects of the training procedure, and thus, the model itself. We pass them to the various class of the algorithm is through a dictionary. The following table describes accurately each and every parameter. You can see the code in detail in Appendix A.1

Parameters		
Variable name	**Description**	**Possible values**
model_architecture_lst	A list with the possible architecture to build a model	['vgg19','resnet50', 'custom']
model_architecture	The selected model architecture	One element from model_architecture_lst
pretrained_model	Select if you want to load a pretrained model	True/ False
unique_id	A unique ID for the newly created model	Integer created by the date and time
model_name	The model name	String that is created by the model architecture and the unique ID
saved_model_path	The path that the model will be saved	A string which describes the path
saved_model_filename	A complete path includes the model's name	A string with complete path of the model's name
root	The path of the dataset	A string which describes the path of the dataset
batch_size	The batch size that we will feed the model in every iteration	Any integer value
Train_dataset_shuffle	Select if the training dataset will be shuffled	True/ False
optimizer_function	Select the optimizer for the training	A list of strings which describes the selected optimizer
Learning_rate	The learning rate of the model	Any float value
Loss_function_type	The type of the loss function	A list of strings which describes the selected loss function
early_stopping_on	Select to perform early stopping	True/ False
steplr_on	Select to perform step learning scheduler	True/ False
lr_list	A list of possible learning	A list of strings which describes
Gamma	Parameter for the learning scheduler	Any float value
Learning_scheduler_step_size	The step of the learning scheduler	Any integer value
Epochs	The number of training epochs	Any integer value

Table 1. The list of hyperparameters.

3.1.4 Dataset processing methodology

The dataset we use contains 289 of DaTSCAN images that are going to be fed to the network, 170 PD cases, and 119 NPD. The dataset we worked is based on datapoints from the dataset of the research "Machine Learning for Neurodegenerative Disorder Diagnosis — Survey of Practices and Launch of Benchmark Dataset" (Tagaris A. et al., 2018:1). Now that we have all the appropriate modules, we are able to apply the explainability techniques that Vivswan Shitole et al, (Shitole V, 2020: 6) argued in their paper.

The goal here is to make a deep learning model to detect if a patient suffers from the disease or not. The creation of the dataset is dependent on the class Dataset which inherits the necessary features from the build-in Pytorch class Dataset. After we defined the path from the hyperparameters class, we specify the transformations. As data are not always in the appropriate processed form that need to be for optimal training, it is required to apply some transformations and perform some manipulations in order to make them suitable. We experimented on various transformation combinations. Afterwards, we create the training and testing datasets, and their respective data loaders. Dataset stores the samples and their corresponding labels, and data loader wraps an iterable around the dataset to enable easy access to the samples. Then, we define the length and the classes of the dataset, we store them in a dictionary and we pass them back to the main, where they will be used to initiate the training phase later on. You can see the code in detail in Appendix A.2.

3.1.5 Creation of the Model

This subroutine has been built with intent to be as dynamic as possible. In this part, we create the models. The idea here is simple, we need to create a model according to an architecture. In this subroutine we provide three options. First, we have the option to create a model. This means that we have total control of how the architecture will be. The Convolutional neural network might be shallow, but in some cases can still be adequately effective. You can see the code in detail in Appendix A.3 The second and third option is based on the idea of transfer learning. Here, we download a complete architecture which is provided by Pytorch. The architectures are ResNet-50 and VGG-19. We have the option to download the pretrained versions of them. This gives us the flexibility to apply transfer leaning, a concept in which we utilize the already stored knowledge of the model by tweaking the weights and biases of the model's last layers. In each case we have modified appropriately the last sequential layer in order to give us accurate prediction between the two aforementioned classes. Now is necessary to explain some necessary concepts such as transfer learning, learning rate, the optimizer and the loss function.

Transfer learning is a deep learning method in which we use a pretrained model for another task. This is considered a popular approach where pre-trained models are used as starting points to create image recognition applications. Without transfer learning, our capabilities to train and try various models would be limited, as proper training needs a huge amount of computation resources, a huge amount of time and huge amounts of training data. With transfer learning we are able to save resources, time without sacrificing better performance of the model.

Learning rate is considered as one of the most crucial hyperparameters for gradient descent, as it scales the magnitude of the model's weights with regards to the learning process. The selection of learning rate can be challenging as a small value could result to the need of a long training process, and on the other hand an extremely large value could speed the learning in such levels that could make the training process unstable. Additionally, learning rate schedulers, allow to change the value as the training process is ongoing, leading to better results. For the training purposes, we use the 'reduce learning rate on plateau', where it reduces the learning rate when the model has stopped improving.

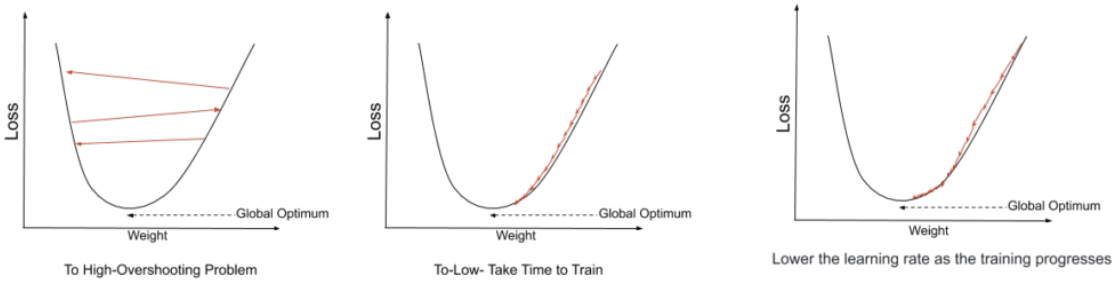

Figure 24. The importance of learning rate in training. The proper configuration can lead to global optimum (Pragati, 2019:1).

Once the model has been defined, downloaded and configured, we create the optimizer. Pytorch package contains optim. Most common optimization algorithms are contained in this package. Optimizer, receives the parameters that are going to be updated with regards to the learning rate that we have defined. The optimization takes place at the end of every training epoch. For the creation of the model, we use Adam optimizer. Adam optimization algorithm is an algorithm for first-order gradient-based optimization of stochastic objective functions and can be used instead of the classical stochastic gradient descend for network weights update. It was firstly introduced by Diederik P. Kingma and Jimmy Ba in 2015 (Kingma D.P., 2015: 2). Adam is considered effective, and it achieves that by using the second moments of the gradients. The above figure shows how Adam performs compared to other common optimizers.

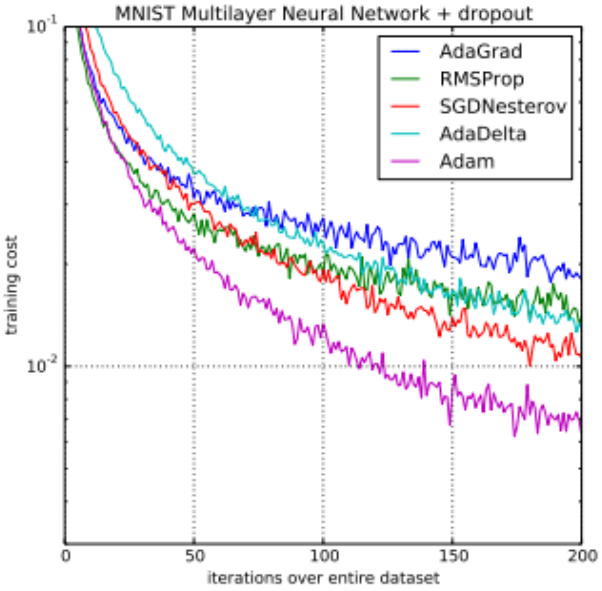

Figure 25. Comparison of Adam to Other Optimization Algorithms (Kingma D.P, 2015: 7).

Loss functions help the model to estimate the error between the target value and the prediction. Essentially, the loss function can help the model realize how far or close is the algorithm with regards the true class value. Further from the truth the model is, the greater the penalty is for it. Loss function determines the model's performance through a comparison between its predicted output with the expected output. In our training we use Cross entropy loss, which measures the performance of the model which its output is between 0 and 1. As the prediction declines further from the label, this function loss increases.

3.1.6 Model Training and Testing

The training procedure is pretty straightforward. Since we have defined the hyperparameters, the dataset, the model, the optimizer and the loss function, now we are ready to begin training the model. In the following lines we will describe some essential concepts that are necessary to be clarified in order to have a better understanding of the creation and training of the deep learning model procedure.

To keep track of the model performance we need to record its statistics. Most common performance statistics are considered accuracy and loss percentage, where accuracy describes how well the model can predict. Also, F1-Score, is a metric where we are able to combine the precision and recall of the model and take its harmonic mean value. Confusion matrices, also known as error matrices, are used in statistical classification and is a table layout in which we can depict in detail the performance of the algorithm by comparing its prediction against the target output. Every row in the table represents the target output and every column the prediction. With this way we can determine where our model faces difficulty with regards to the classification. Thus, a confusion matrix contains the following information.

- True positive (TP): A test result that correctly indicates the presence of a condition or characteristic
- True negative (TN): A test result that correctly indicates the absence of a condition or characteristic

- False positive (FP): A test result which wrongly indicates that a particular condition or attribute is present
- False negative (FN): A test result which wrongly indicates that a particular condition or attribute is absent

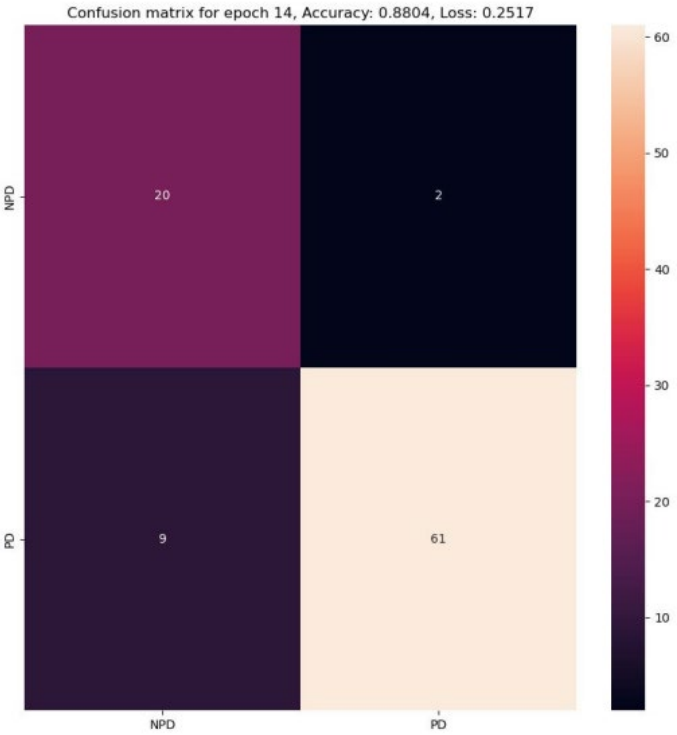

Figure 26. An example of a confusion matrix.

Additionally, Tensorboard is a great tool for sharing visualized measurements. In order to have a better view of our model's performance we need to record and push in a structure form its metrics, also during the training, we save the model's performance in a structured pdf form in which we can also use for sharing and analyzing the model's metrics. This offers a dynamic resolution around the model which we can share in any way is more convenient at the current time.

First, we train our network using the training set. After every epoch training is finished, we test our model using the test set. So, in every epoch we have two phases, the training phase and the testing phase. In every epoch, we need to switch the model's mode to training and evaluating respectively. First, we initialize by zeroing the running failures and the running corrects. Then we pass the first batch of the data loader containing the images. All predictions are stored in a variable and then they compared with the labels in order to determine, the accuracy and the loss. If we are in training phase, we perform a loss function to calculate how far a prediction was with regards to the target output, and we perform optimization. It is important before an optimization procedure, to zero the gradients in order to optimize correctly. Depending the phase, we store the metrics to the appropriate performance list in order to be used later on. In every epoch, we save the model, in order to keep the best performing for later, and of course to test different phase as well. You can see the code in detail in Appendix A.4

Before we review the code implementation, we should refer to early stopping. Due to the fact that too many epochs may drive our model to overfitting, early stopping allows us to specify when the training is enough and stop it. The metric that is used for the early stopping to take control is the decline of improvement amongst a certain tolerance threshold on the training phase of the model. In our case we set the tolerance to 5 epochs.

In order to have a broader selection of experiments and results, we create some models with different techniques. Cross-validation refers to the statistical method that is being used to give us an estimation of the machine learning model skills, i.e., how well the model will perform on unknown data. This method allows us to compare and select the most appropriate model by comparing it upon a prediction problem. In our case we used the 10-fold cross validation method. Essentially, it is a resampling method which breaks the training dataset into k subsets, and trains

the model using k-1 subsets at each iteration, while the last subset is used for testing. When the procedure has finished, we compare and choose the best, according to some criteria, model to run and generate the Structured attention graphs.

3.1.7 Model Saving

This function is called to save the model. With some simple lines of code with create a path, create a folder and then save the model. You can see the code in detail in Appendix A.5

3.1.8 Performance Report Creation

This class creates a model report which is used to store all the useful information which are related with the performance of the model. This piece of code, help us to store the hyperparameters, the model architecture, and the visualized performance of model regarding the accuracy, the loss, the F1-Score and the confusion matrices. You can see the code in detail in Appendix A.

3.1.9 Model Statistics

We created various model and ran various experiments. In the table below, we show the models' statistics we developed and worked with, as well as their related charts.

Name:	VGG1	VGG2	VGG3
Model architecture:	VGG-19	VGG-19	VGG-19
Optimizer:	Adam	Adam	Adam
Loss function:	Cross Entropy	Cross Entropy	Cross Entropy
Learning rate:	0.001	0.001	0.0001
Epochs:	15	24	12
Accuracy:	79.35%	81.52%	96.55%

Table 2. The models' statistics.

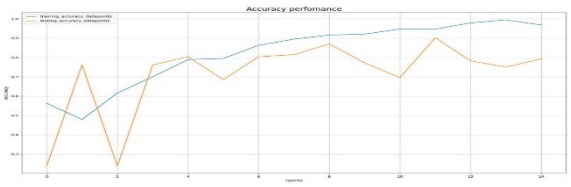

Chart 1. Accuracy performance of the model VGG1.

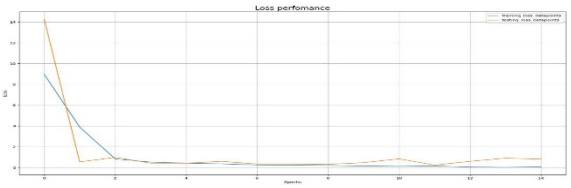

Chart 2. Loss performance of the model VGG1.

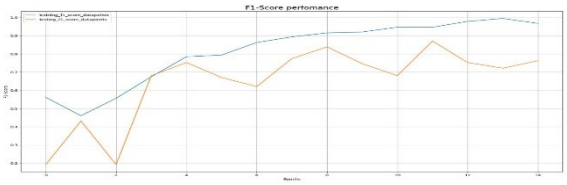

Chart 3. F1-Score performance of the model VGG1.

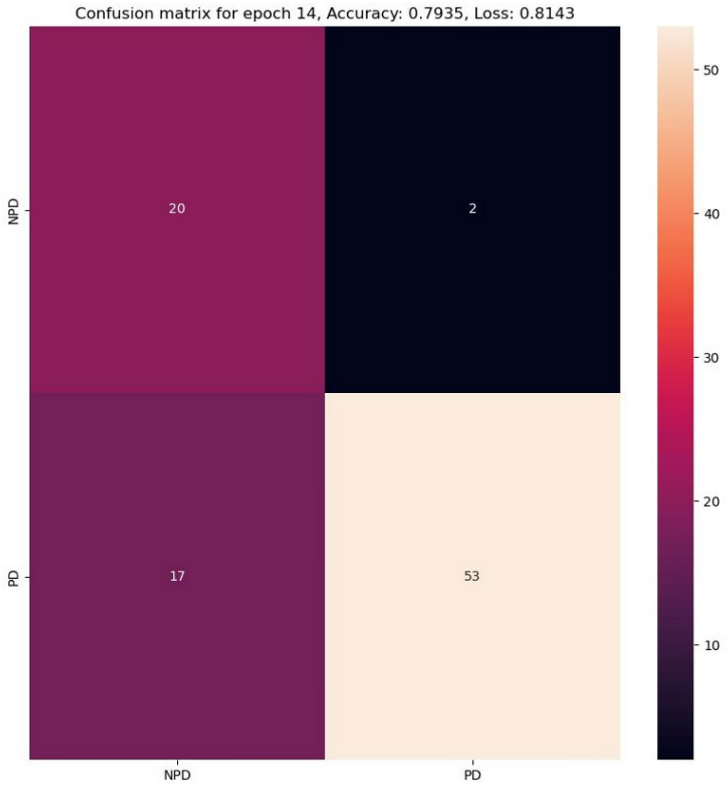

Chart 4. Confusion Matrix of the model VGG1.

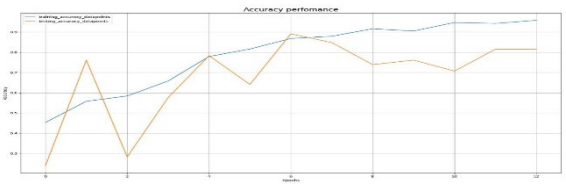

Chart 5. Accuracy performance of the model VGG2.

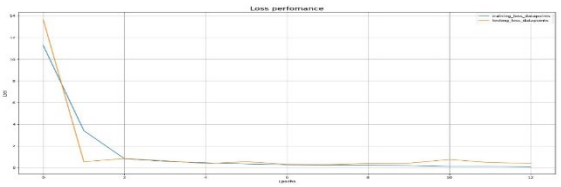

Chart 6. Loss performance of the model VGG2.

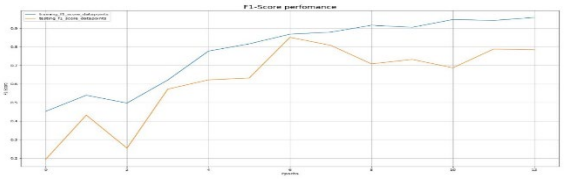

Chart 7. F1 – Score of the model VGG2.

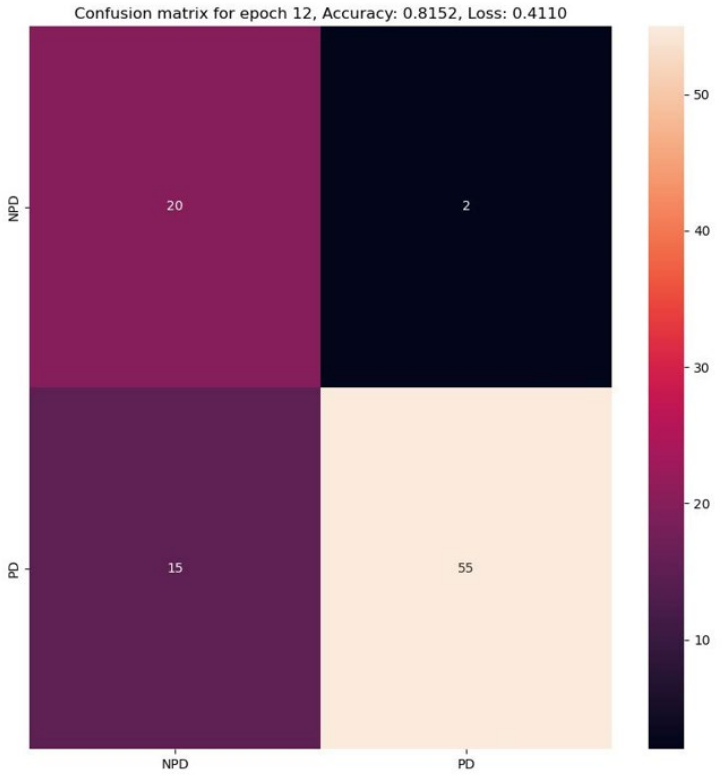

Chart 8. Confusion Matrix of the model VGG2.

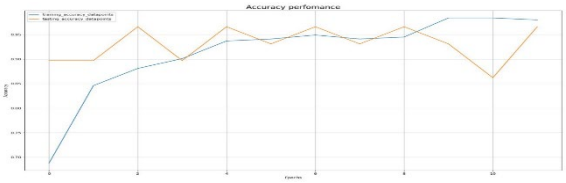

Chart 9. Accuracy performance of the model VGG3.

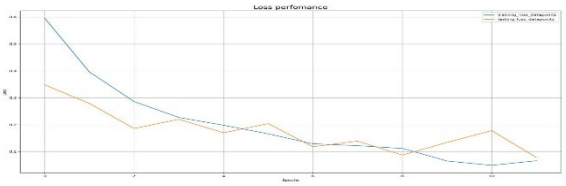

Chart 10. Loss performance of the model VGG3.

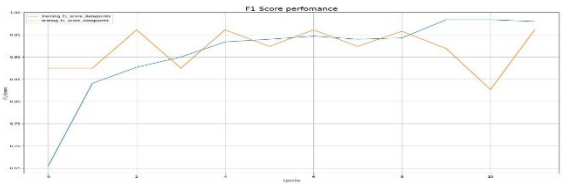

Chart 11. F1-Score performance of the model VGG3.

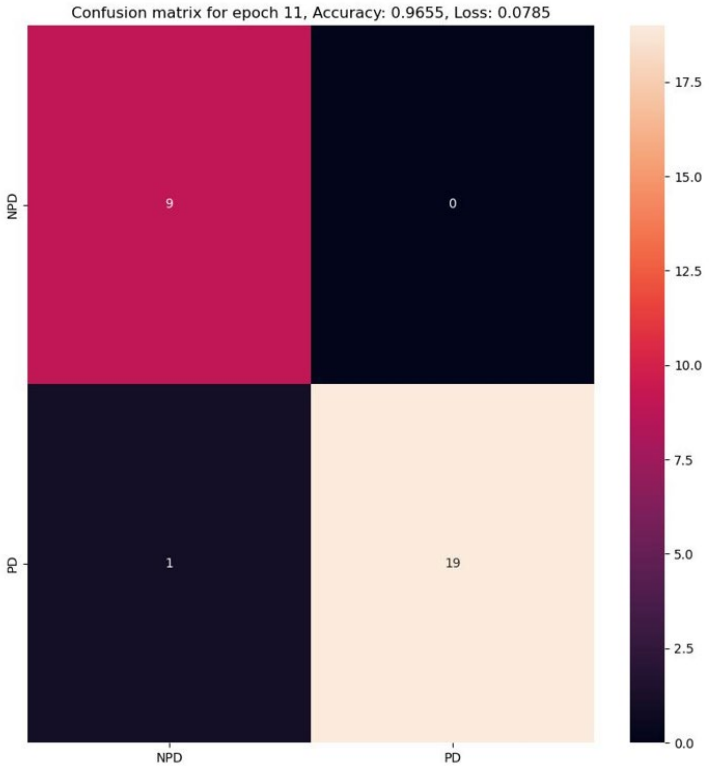

Chart 12. Confusion Matrix of the model VGG3.

3.1.10 Model Training with Augmented Data

Data augmentation refers to the creation of a large dataset by generating processed and carefully distorted images, based on a smaller dataset. We created the code which is shown in Appendix D, in order to generate augmented images which could help us investigate if this is the correct approach on proceeding with the SAG generation of PD images. As it was mentioned before the images are 282 in total, and from this dataset we created a larger set of 2820 datapoints. Despite data augmentation can help in multiple cases, the results were not adequate, both in training/testing phase and in SAG generation. The performance was very low and the models could not generate any results on SAG algorithm, even with a balanced dataset. One indicative example of the augmented data is depicted below. Also, the following graphs show the performance of each model in each phase, we have orange for the model architecture, blue for ResNet50, and red for VGG19. One reason could be on the structural information inside the datapoints. When we distort the colors, the useful information is distorted also, when we rotate the image too much, we noticed the confidence level of the model dropped. In general, results show that the architecture and ResNet-50 perform poorly against VGG-19, which the latter showed great performance in every experiment, even in situation like the current.

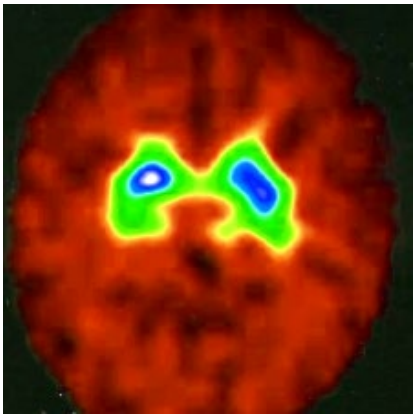

Figure 27. Example of an augmented image with label: PD

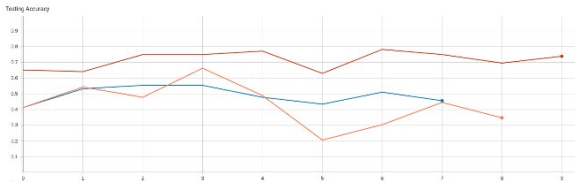

Chart 13. Training Accuracy.

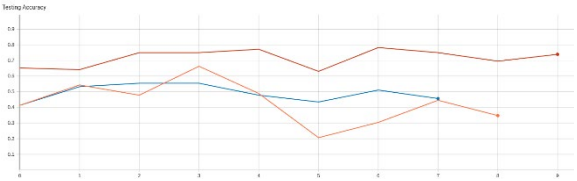

Chart 14. Testing Accuracy.

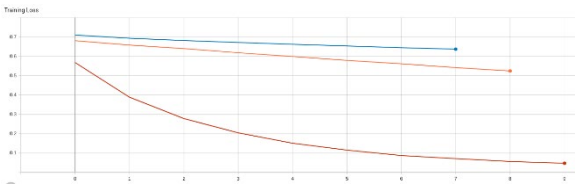

Chart 15. Training Loss.

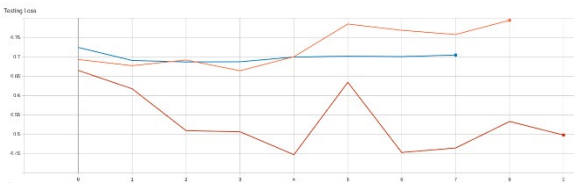

Chart 16. Testing Loss.

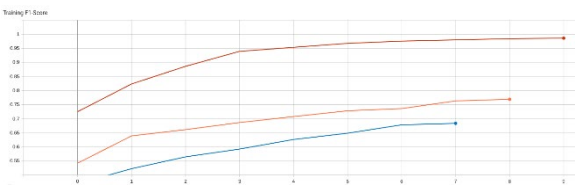

Chart 17. Training F1-Score.

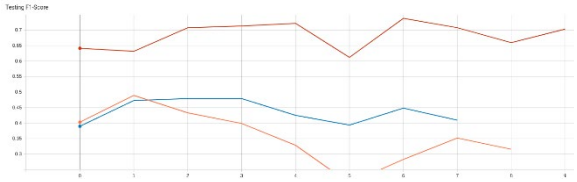

Chart 18. Testing F1-Score.

3.2 Application of Explainability Techniques upon the Developed CNNs

As explained in the paper (Shitole V, 2020: 6), there might be a plethora of explanations, which might make the investigation computationally expensive if we divide the picture into pixelated patches. That is why the algorithm subdivides each image into 7x7=49 patches. which later are down-sampled into pixels. This is inspired by the resolution grad-CAM uses (Deng J. 2009: 3). Then these patches, along with the utilization of search algorithms, provide combinatorial sub-images and feed the CNN, in order to determine if the CNN is able to explain the prediction. If the newly produced image is able to get a high score of confidence from the network, then it is characterized as a minimum sufficient explanation. Specifically, if the sub-image is able to score at least 90% of the original image's confidence, then the sub-image is a minimal sufficient explanation. In other words, if the sub-image can produce at least 90% of the image's confidence level, the most important information of the image is placed in these patches. There are two ways to search for MSEs, with restricted Combinatorial Search and Beam Search. The algorithm mainly focuses on beam search. The prevalent MSEs will be used to generate the SAG. Additionally, these MSEs create a monotone disjunctive normal form, which indicates the prevalent set of patches. Each conjunction is evaluated with regards to the confidence level by letting the model predict the image in which only the patches that form the conjunction are visible, also known as insertion image. Also, as a comparison the authors let the model predict an image, in which the patches forming the conjunction are deleted, also known as deletion image. This method creates a work product which allows us to inspect a node in detail. Furthermore, an attention map is generated with average pooling with regards to each patch. This map is down-sampled to lower resolution in order to be used as a mask upon the original image. Since this shows the confidence of the top MSEs, it is time to create the SAG tree with MSEs as root nodes. Their children are created through patch deletion procedure, which deletes one patch at a time to calculate the confidence of each child node. In order to not overload the SAG, nodes with confidence lower than 40% are deleted. The code of SAG can be found in Appendix B.

Now, is time to run the grad-CAM and SAG generation algorithm in order to investigate and analyze the results it will give upon the Parkinson's dataset (Tagaris A. et al., 2018:1). Up to this point we had to deal only with the model's prediction, without knowing if the model focused on the correct parts of the image. There were many variations, regarding the data that have been used in training, the model architecture and the hyperparameters. In every variation trial, we will investigate one picture from each class. In SAG generation algorithm, researchers load a Pytorch pretrained model. The usage of a pretrained model saves us a lot of training time, which can be found valuable to other research activities. In their paper, all the experiments and results were taken into place with VGGNet, specifically with VGG-19, nevertheless researchers briefly analyze the performance of a Resnet50 as a classifier. We re-trained an already pretrained VGG19 with transfer learning. This allows to take an already pretrained model, modify the last layers, and retrain the last layers in order to train it upon our dataset. The function that loads the pretrained model in SAG algorithm, was modified in order to host the developed models. To do this in Python, we define the directory where the model is saved, then we save the last sequential layer into a variable and we modify it according to our number classes. In our case the number is two. Then, we load each layer to its corresponding parameter tensor and with this way, we practically load the saved state of our model. Afterwards, we set the model to evaluation mode and we set all of the model's parameters gradients to non-computational mode. Now that the model is loaded and ready for use, we are ready to investigate the SAG algorithm. In the following lines, we will provide the results of each image.

3.2.1 Challenges Met During the Implementation

We need to adapt the algorithm and shape it according to our needs as it is not a 'plug and play' implementation. On the contrary, there are many features that we had to take into account in order to have a workable product which is able to collaborate with a convolutional neural network. The work of Shitole et al. (Shitole V, 2020: 6) has to deal with already pretrained networks such as VGG-19 and ResNet-50 which their performance is pretty much expected to be high. With that in mind, it was obvious that some tweaks inside the algorithm were more than necessary, and of course some problems that weren't predicted in the initial work had to be overcome.

3.2.1.1 Loading a Model in SAG Algorithm

The first challenge was on how to load a convolutional neural network. The concept of loading in general an already pretrained artificial neural network is simple in Pytorch. Although our model was an already pretrained model, we had to modify its architecture on its sequential layer, especially in its output, to reduce from 1000 to 2. Thus, the classifier is significantly reduced compared to the pretrained models Pytorch offers. In our code, the classes of the

dataset and the output of our model are basically the same, and with this way we are able to generalize and create models for any dataset we have available. The label information is drawn in the folder name that the images are stored, and in this way the model loading is flexible in order to experiment with any dataset and any trained network. The steps that are followed are the following. Step 1 to 4 are implemented by a function that is called from the function that is initially built to load the pretrained model. Steps 5 to 7 have been already included in the work of Shitole et al (Shitole V, 2020: 6).

1. We load the dataset in order to define the name and the number of classes. We receive a list with the class names as its elements, and the length of the list is the number of classes.

2. We load a pretrained model depending on the option that we have available. In our case we can load VGG-19, ResNet-50 or an entirely model architecture.

3. We modify the last layer. The number of outputs will be the length of the list we discussed in step 1. Various classifier architectures were tried, this will be discussed in a later stage.

4. Now, we load the state of the weights that we save during the training of the convolutional neural network. Practically, we need only the last layer's weights, but we can only load the entire weight state of the model.

5. We set the loaded model to evaluation mode. Evaluation mode, notifies to all layers that now we are in inference (evaluation) mode, and with this way batch norm and dropout layers will work in this mode.

6. If we have a GPU available, we transfer the model there.

7. We are freezing the entire model in order no changes to happen to its parameters with torch.no_grad(). This disables the backpropagation, which in this case we do not need. It helps to reduce memory usage and speeds up the computational power

3.2.1.2 Ground Truth Label

The generation of SAG algorithm takes one by one pictures and works with them individually. Developers of the code have created a text file which contains all categories of VGG-19 and ResNet network, in order to create the result products in a later stage of the algorithm. The text file is named GroundTruth1000.txt. This file has to be modified manually, thus the first two categories were changed as '0: No Parkinson's Disease, 1: Parkinson's Disease'. For simplicity punctuations have been avoided inside the code. When the model makes a prediction, the category is being drawn from this file and it is used to create result products.

3.2.1.3 Perturbation Mask

First, we need to create the low probability blurred image as a baseline. The blurred image could be a blackened image, a distorted by Gaussian parameters image, a distorted by Median values image, or by mixed values. The inputs of this function are:

1. input_img: the original input image.

2. img_label: the classification target

3. model: the model that you want to run.

4. resize_shape: the input size for the given model.

5. Gaussian_param: parameters for Gaussian blur.

6. Median_param: parameters for median blur.

7. blur_type: Gaussian blur or median blur or mixed blur.

8. use_cuda: use gpu (1) or not (0).

Then, we pass the image to get the prediction of the model. Since we predicted the label of the image, we obtain the perturbation mask by using integrated gradient descent, in order to find the smallest and smoothest area that maximally decrease the output of a deep model. As a result, we get the mask and its up-sampled version. The parameters that are used here are the following:

1. ups: upsampling factor
2. img: the original input image
3. blurred_img: the baseline for the input image
4. model: the model that you want to run
5. category: the classification target (category=-1 means the top 1 classification label)
6. max_iterations: the max iterations for the integrated gradient descent
7. integ_iter: how many points you want to use when computing the integrated gradients
8. tv_beta: which norm you want to use for the total variation term
9. l1_coeff: parameter for the L1 norm
10. tv_coeff: parameter for the total variation term
11. size_init: the resolution of the mask that you want to generate
12. use_cuda: use gpu (1) or not (0)

Now that mask is created, we need to get all distinct roots that can be found via beam search. By default, upper limit on number of roots obtained via search is ten. Afterwards we get this set of non-overlapping roots, and we prune it to get the top three based on their score. This is the selection phase of the roots that we want to show in the structured attention graph. The maximal number of roots are three, as the researchers did not wish to have an overwhelming number of roots in SAG. After this phase, we build the SAG. For each root, child nodes are being recursively generated by deleting one patch of the visible image on the root, at a time. Then the calculation of the confidence for each node being done by passing it to the model. If the confidence is less than 40%, there is no reason to add it to SAG as it is not adding any value and will cause visual cluttering.

3.2.1.4 Definition of the Last Layer

Another challenging part on the model creation was to define the last layer properly, in order to get accurate results, due to the fact that the architecture affects also the performance of the SAG algorithm. For the model, we chose a simple solution on the last layer, that is referred in the code implementation of the model creation in Appendix A.3.

```
# Configure the classifier
self.model.classifier = nn.Sequential(nn.Linear(25088,3136),
                        nn.ReLU(inplace=True),
                        nn.Dropout(0.5),
                        nn.Linear(3136,3136, bias=True),
                        nn.ReLU(inplace=True),
                        nn.Dropout(0.5),
                        nn.Linear(3136,len(self.class_names),bias=True))

# note: Newly constructed layer has requires_grad=True by default.
# You don't need to do it manually.
self.model_parameters = self.model.classifier.parameters()
```

Figure 28 . Code snapshot: The classifier, here we define the layers, its weights and its dropout rate.

3.2.1.5 Broken MDNF Boolean Expression

One major challenge was that in some cases the monotone disjunctive normal form Boolean expression was incomplete or nonexistent, for example dnf.txt file might have as MDNF the (2), which indicates an incomplete formula. As it stated before, a MDNF is basically an MSE. Here the vertical line is the symbol of or sign, and the ampersand symbols the and sign and each element P_x symbolizes a specific patch in the image. Thus, this can be interpreted that the MSE consist of P_{23} and P_{32}, or, P_{23} and P_{38}, and it would be a proper MSE if the blank value in the beginning did not exist. This implies that the root's children were at least equal to the confidence level of the root. This issue was unavoidable and a possible reconstruction of the whole implementation may be necessary, as many recursive procedures to the construction of roots are entangled into the code, and the complexity increases making a simple fix extremely difficult. Thus, the solution we propose is to nest the loop into a try-except method, since we consider the current as a "lost-cause", and we move to the next image.

$$\text{(Empty value)} \mid P_{23}, \& P_{32}, \mid P_{23} \& P_{38} \qquad (2)$$

3.2.1.6 Infinite Child Nodes

The accuracy level of the neural network, sometimes causes trouble to the creation of the child nodes as it seems that it can get a correct prediction from patches of the image where information is irrelevant. The probability threshold in the combination of these patches is close and above to the probability threshold of the image (in our case 0.9), and this causes the model to disregard the patches where true information lies. This results to many problems. One is the occupation of a huge amount of space in the hard disk drive, where the child nodes are saved. Second and most important, is the creation of SAG which is practically bloated with "infinite" number of roots; thus, it is not visible. This is another "lost cause" case. This issue has been treated with a timeout error. If the generation SAG exceeds the time limit of 120 seconds, a timeout error will be called and drive the algorithm to a timeout exception.

3.2.1.7 Multiple Models Evaluation

Especially with 10-fold cross validation, the evaluation of a vast number of models have raised the need to automate the evaluation procedure amongst the SAG and the models that have been created. This has been solved by nesting the main function of SAG generation into a loop, and by giving every time the path of the model, we call this function and we start the evaluation. The generation of SAG is run on every image as it has been aforementioned. When all of the images have passed through the SAG algorithm for the specific model, the results are moved from the Results to the model file with a tag which indicates the date and time of the run. This packet is served with a text file which gives detailed information of what happened to each picture during the run. This file contains the value of the hyperparameters that the algorithm run, and the name of model. Additionally, for each image, this file contains its folder, the label, the prediction and from this we can get some information if the run was successful or if it fell on one of the aforementioned issues. This loop can be run for a group of models, or for an

individual model. This method accelerated the procedure and the evaluation of the trial-and-error process upon the models.

All the above have been proved to make the adaptation of the SAG to neural models possible. These necessary changes made it possible to get results that are able to provide minimal sufficient explanations to our dataset. Despite the fact that these changes are able to generalize the SAG algorithm into other models and datasets, it should be noted that a possible reconstruction of the algorithm – and maybe of the concept – may be necessary to provide more robust results. Nevertheless, this approach restrains from the suggested methodology that researchers propose into their paper (Shitole V, 2020: 6).

3.2.1.8 Unbalanced Dataset

As it was aforementioned the dataset is a total of 289 datapoints, with 165 PD cases, and 124 NPD cases, which was already filtered and separated into classes. Thus, we have to deal with an unbalanced and small dataset. In the experiments it was observed that the class distribution and the dataset size caused problems in the training procedure of the model. This issue led to have models which were overfitted and biased towards PD cases, causing problems to correct predictions. After investigating the dataset, the following changes were made and in order to solve the overfitting problem, and have better results.

1. From the NPD cases, seven datapoints were deleted as their low resolution was causing problems with the prediction.

2. As a last step, we distributed equally the data between the training and testing dataset, and we end up with 190 training dataset size, with 95 PD cases, and 95 NPD cases. The testing dataset has now 92 datapoints, 70 PD cases and 22 NPD cases.

3.2.2 Results with Selected Image of the ImageNet

In order to further test the concept and the algorithm we selected random image from the ImageNet database. Here we test the algorithm with a different class, an image of a cat to see what the SAG output will be. Results shows that the algorithm is capable to perform on other images as well. We run the images with VGG19 and ResNet50 model architectures. The MDNF formula of this image is P10 & P16 & P27 & P30 & P31 & P38. As always is can be confirmed by the image grid and the root of the image. The following results are with VGG19 model architecture. We observe that we only have one root, but the SAG ends up bigger, as more patches of the root contain useful information about the prediction with confidence level greater than 40% in contrary to the previous image. As it is depicted in the chart, as the layers gets deeper, the confidence level gets lower.

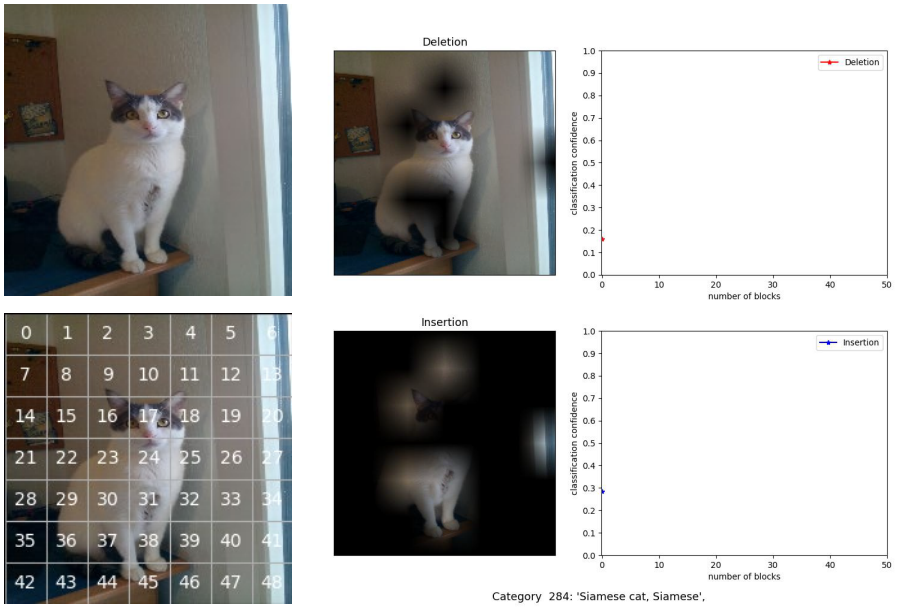

Figure 29. Top left, the original image with label: Cat. Bottom left, the grid image. Right image, the prediction confidence of the deletion and the insertion image, with VGG19.

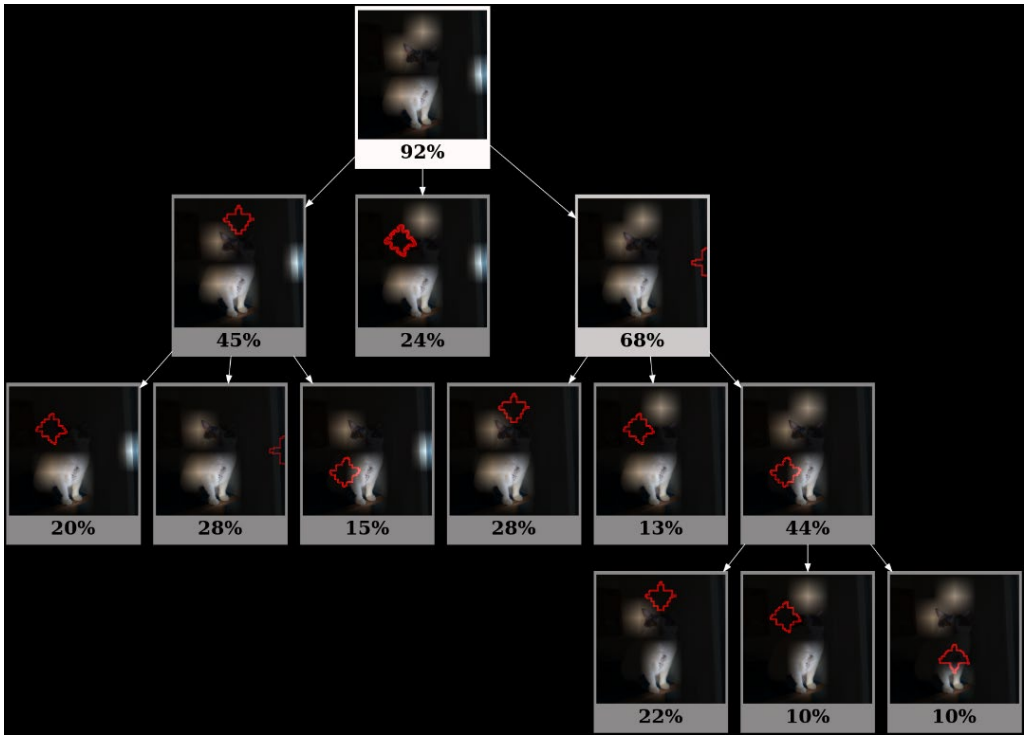

Figure 30. The Structured Attention Image of the prediction, with VGG19.

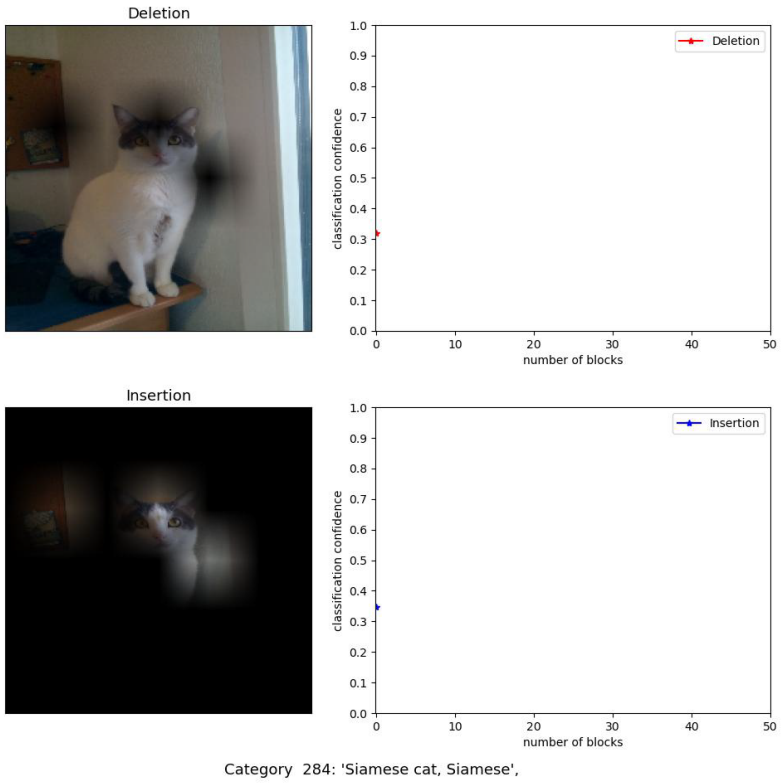

Figure 31. The prediction confidence of the deletion and the insertion image, with ResNet-50.

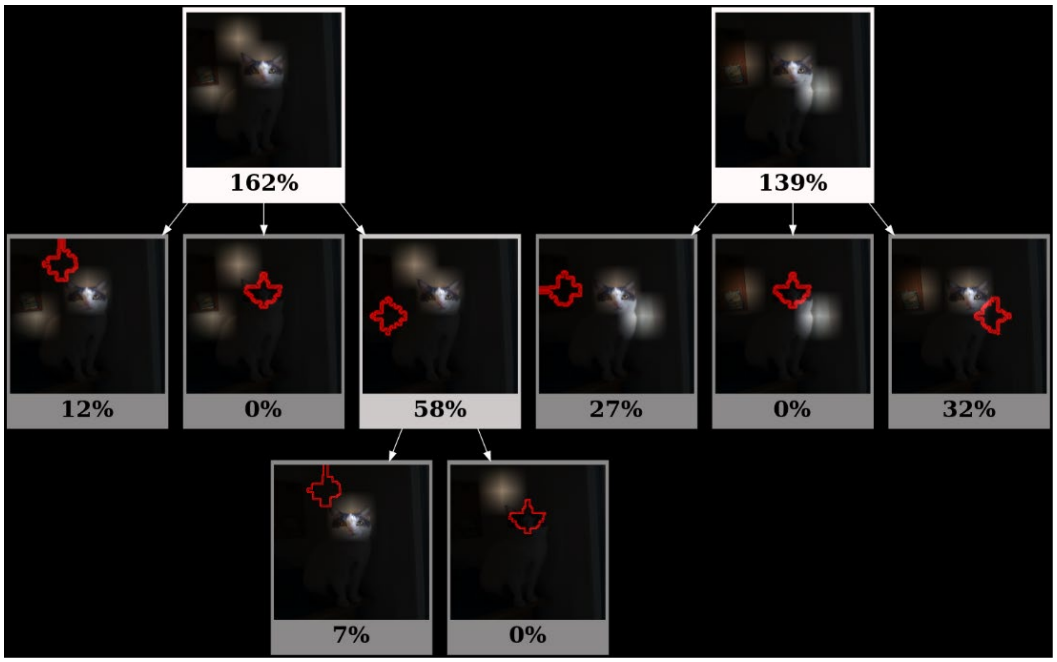

Figure 32. The Structured Attention Image of the prediction, with ResNet-50.

To have a better understanding of the algorithm's capabilities we ran it on a random ImageNet image. Again, the outputs were the expected work products, such as the grid image, the perturbated image and the insertion/deletion image, and the SAG. Prediction is not quite correct, this cat is definitely not Siamese, but still the animal prediction is on point. The SAG shows that great focus and confidence is placed on the facial characteristics of the cat, and this confidence is reduced as more patches are hidden in the deeper nodes. The fact the we get 0% in some of nodes, which show nothing useful, is the confirmation that the SAG not only predicts correct, but also in the right place.

3.2.3 Results with Developed Model Upon the Parkinson's Dataset

The code with which we generated the SAG results is available in appendix B. Also, it is available in the following link https://github.com/karvo/ouc-cogsys-custom-structured-attention-graphs. The code with which we generated the grad-CAM results is available in appendix C, and in the following link https://github.com/karvo/ouc-cogsys-pytorch-grad-cam. The data augmentation code is available is appendix D, and can be found also in the same link with the code of appendix B.

We will present the results for the two following datapoints. These datapoints were not present inside the dataset during the training period of the models. In order to understand and evaluate the degree of performance of each model with regards to the explainability techniques we applied, we stuck to the high-level rules for detection of signs of Parkinsonian syndrome (Marek K. et al., 2020: 18), which are mentioned in section 2.3. We worked on various methods, but mainly with VGG-19 architecture. All of the models were trained with transfer learning method. We proceed to different experiment, from which, we will show the results in the following sections. On this experiment, simple transformations are performed which are shown on Appendix A.2. Also, the experiments included different probability threshold. As a reminder, probability threshold (P_h), is the minimum score we expect for a minimal sufficient explanation to achieve in order to be included in the SAG. Furthermore, we review the performance of the models on grad-CAM algorithm. As we have already mentioned, after overcoming the challenges that were met and described in detail before, results show that with probability threshold, the models are performing well on NPD prediction. The SAG, shows that the models focus on the right parts of the image which is the area of striatum. In some cases, despite the model predicts the correct label, it usually struggles to find any meaningful roots to include in the SAG, which depict the damaged striatum. On the contrary, SAG points to irrelevant places of the image, such as the 'edges' of the skull, which means that in this case the classifier has not identified the relevant parts of the image that attribute to the classification accurately. We experimented with various training techniques and various hyperparameter configurations. The techniques have already listed above, and the results are shown below. We will see on the below results that the method of early stopping helped to increase the model performance, and building a model that did not lead the SAG algorithm to show irrelevant patches of the image.

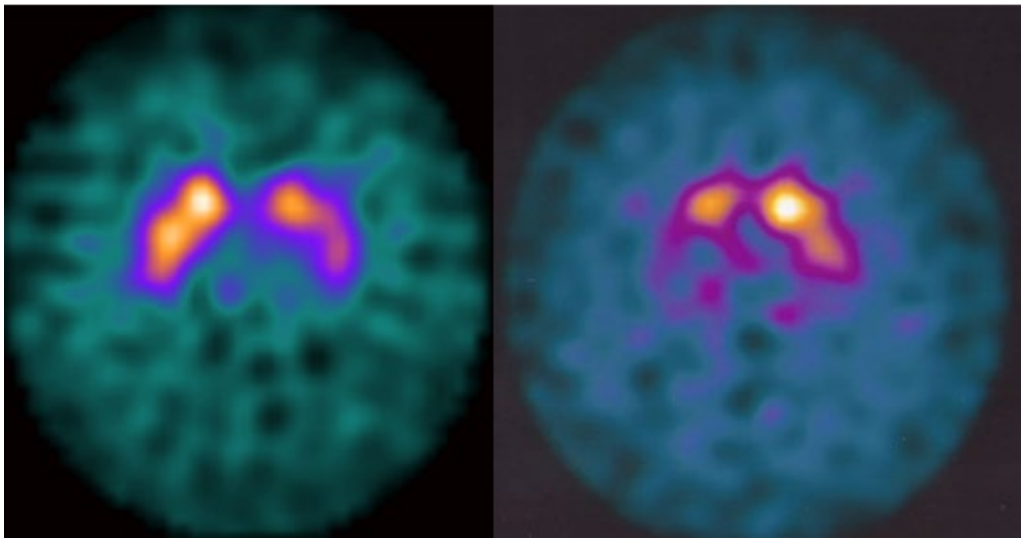

Figure 33. Left image: Non-Parkinson's Disease case, folder: NPD3, image number 8.

Our experiments included the following approaches.

- SAG generation with Model VGG1 and probability threshold at 90%
- SAG generation with Model VGG2 probability threshold at 90%
- SAG generation with Model VGG3 probability threshold at 90%
- SAG generation with Model VGG1 probability threshold at 97%
- SAG generation with Model VGG2 probability threshold at 97%

3.2.3.1 SAG Results with Model VGG1 and Probability Threshold at 90%

The model we used here is VGG1. It's accuracy on last epoch 14 with score was 79%. NPD case is on point, with a SAG sufficiently deep and pointing to the patches where the striatum is depicted. Although, we face problems on PD cases where the SAG is shallow and the SAG is not pointing to the proper parts of the image. Confusion matrix shows that false negatives and false positives are relatively low at 20%.

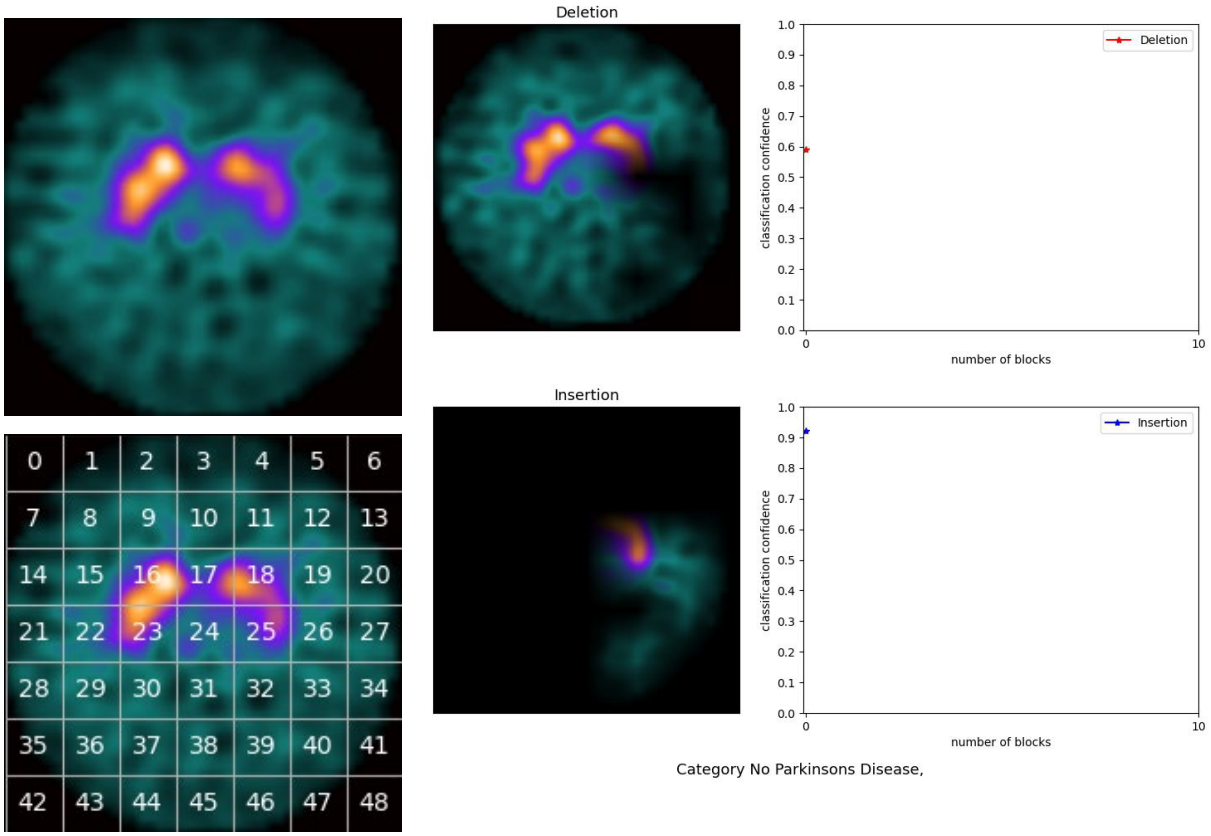

Figure 34. Top left, the original image of NPD case. Bottom left, the grid image. the insertion and deletion image for second root with model VGG1.

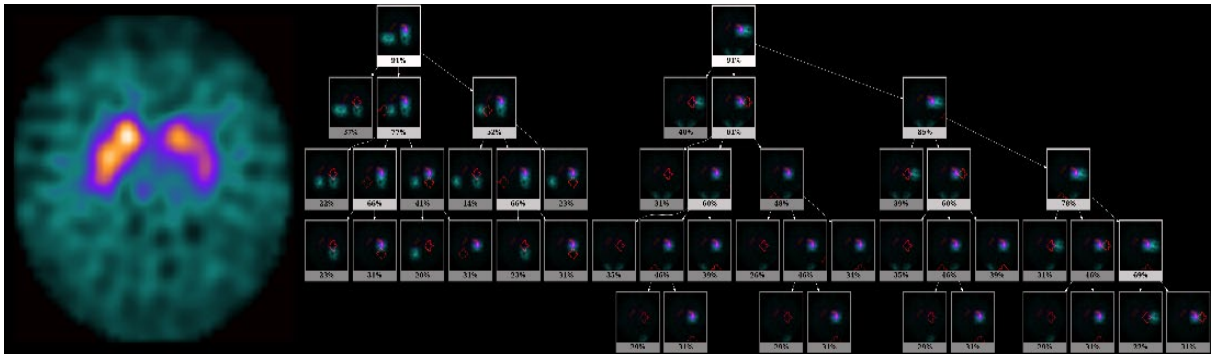

Figure 35. The Structured Attention Graph of NPD case with model VGG1.

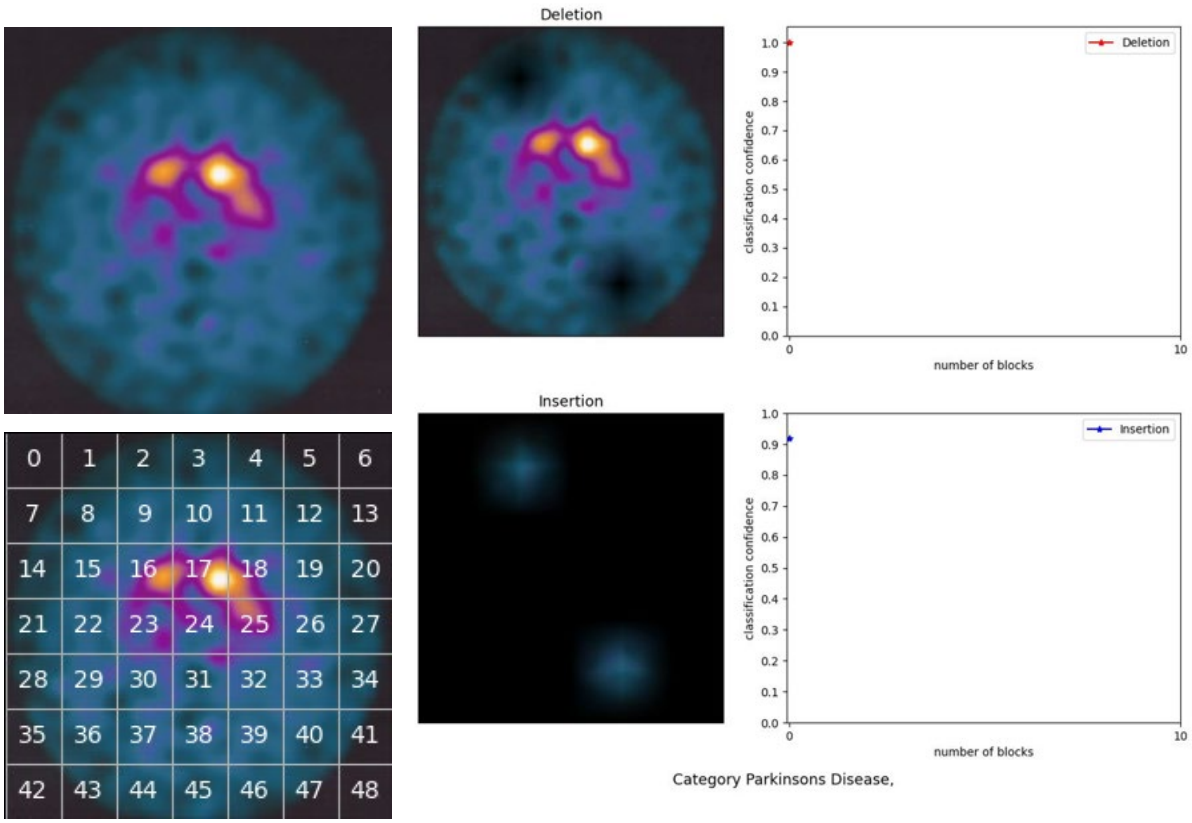

Figure 36. Top left, the original image of PD case. Bottom left, the grid image. the insertion and deletion image for the third root with model VGG1.

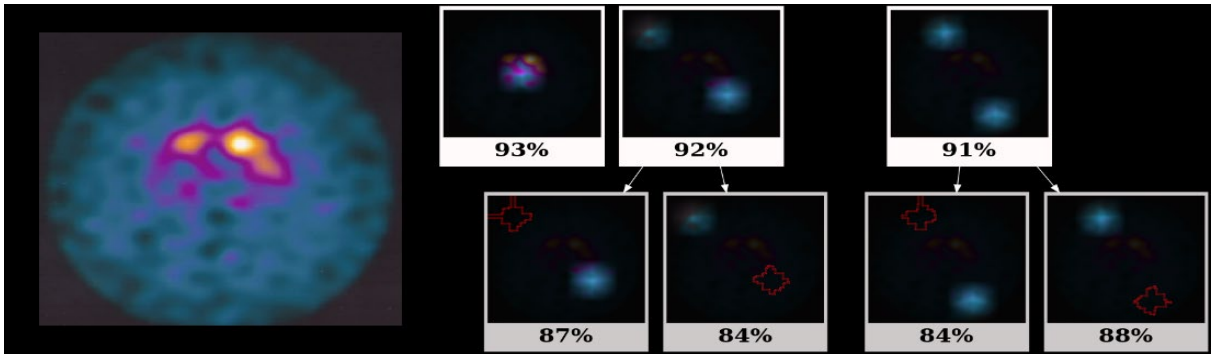

Figure 37. The Structured Attention Graph of PD case with model VGG1.

3.2.3.2 SAG Results with Model VGG2 and Probability Threshold at 90%

The model we used here is VGG2. It's accuracy on its last epoch 12 is 81%. Here, the quality of the SAG on NPD case drops as the first root is looking to meaningless patches with confidence 43%, this leads to divide further the roots. Additionally, the SAG of NPD is shallow, but still it can be considered as meaningful overall thanks to the second and third root. Also SAG quality gets better on PD case, which directs its attention further to the area of striatum. Here, we see now a rich SAG with a descent depth. Despite the fact that it points to some irrelevant parts of the image, overall, the quality of PD SAG is relatively better than the previous case without early stopping.

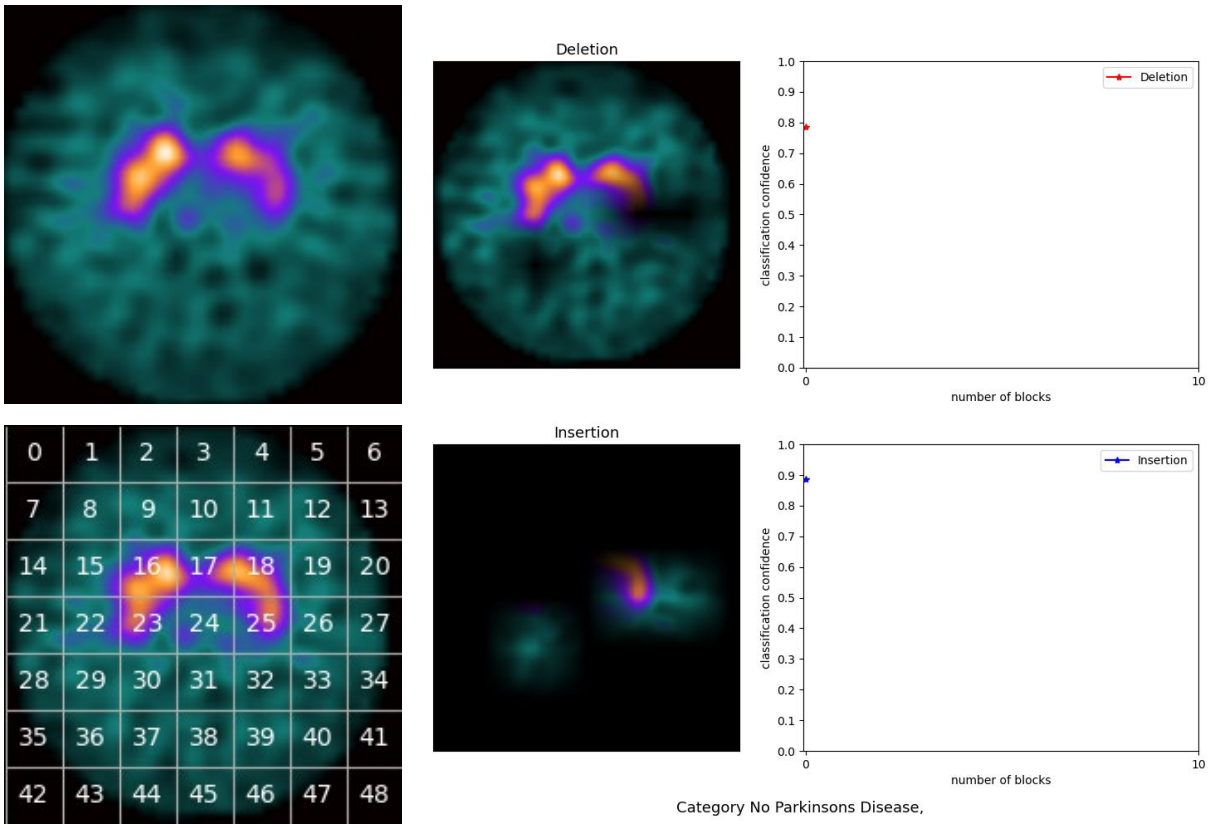

Figure 38. Top left, the original image of NPD case. Bottom left, the grid image. the insertion and deletion image for the third root with model VGG2.

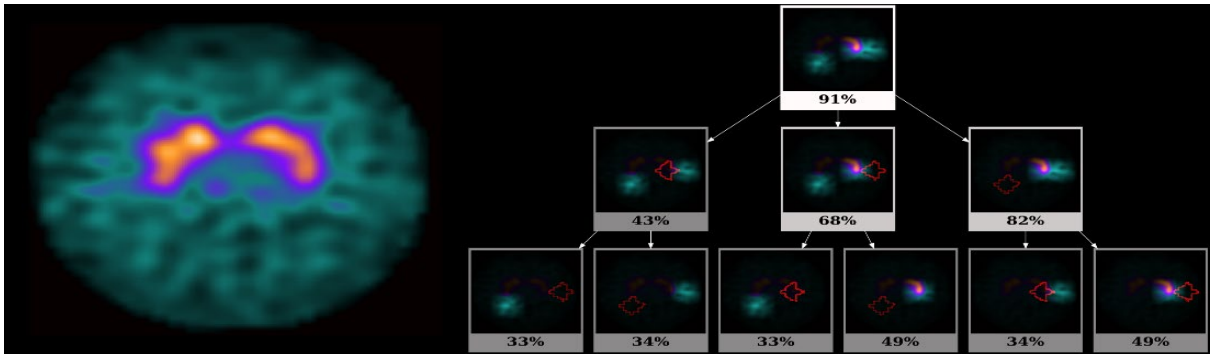

Figure 39. The Structured Attention Image of the prediction of NPD case, with model VGG2.

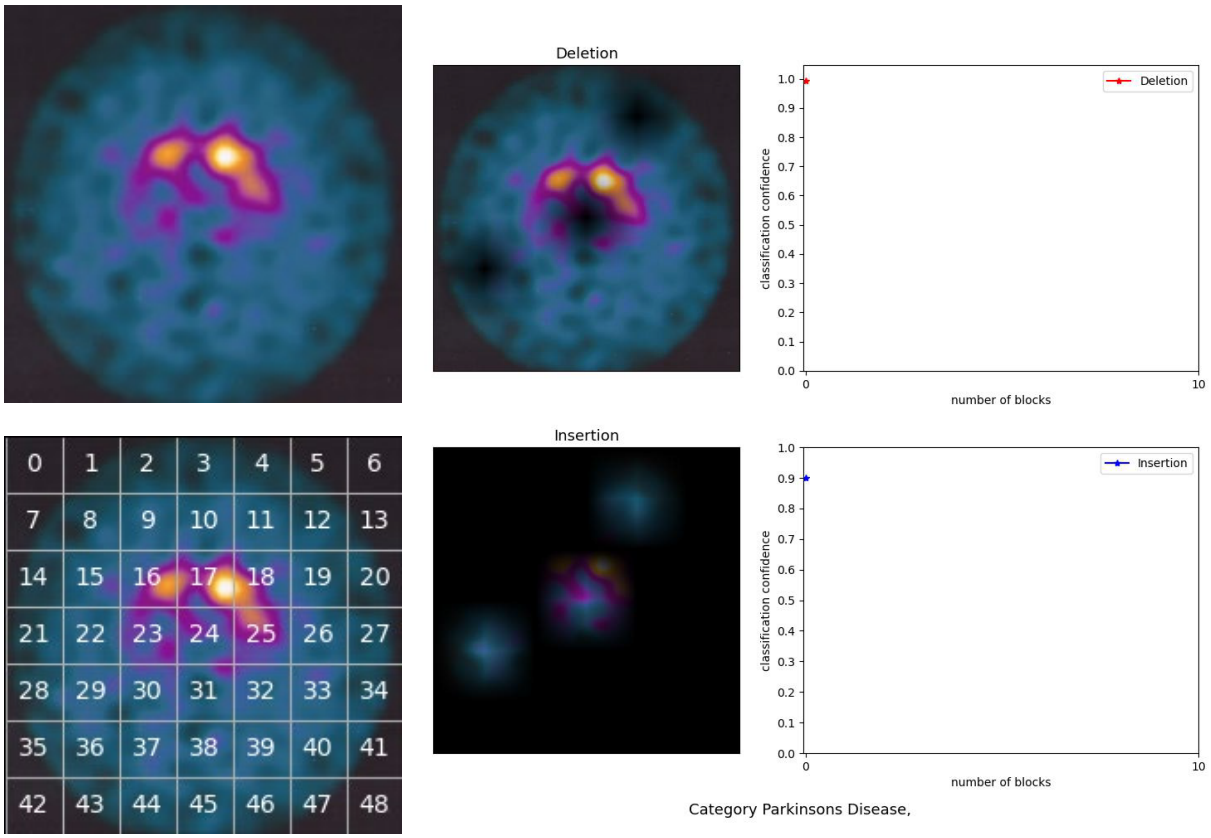

Figure 40. Top left, the original image of NPD case. Bottom left, the grid image. the insertion and deletion image for the third root with model VGG2.

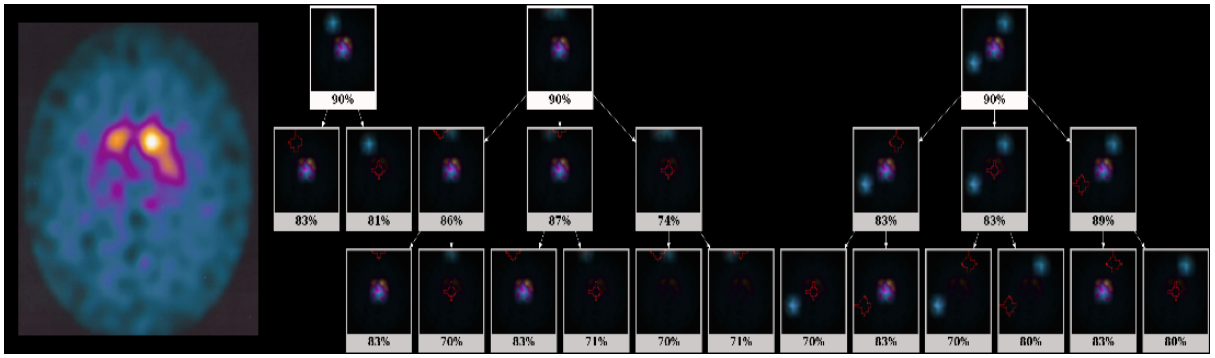

Figure 41. The Structured Attention Image of PD case with model VGG2.

3.2.3.3 SAG Results with Model VGG3 and Probability Threshold at 90%

We use 10-fold cross-validation to see how well the model trained on a set of data can generalize to data that remained unseen during the training. This is done by dividing the dataset into k sub-sections, training every time with a different sub-dataset, and evaluating with the rest of the data. Since in our code we have divided the dataset into 10 pieces, we end with 10 candidate models. The model is chosen over its overall performance and its mean value of the accuracy score for all epochs. Most of the models have accuracy over 85% and their score is close to one another. Here, we observe good quality on SAG of the NPD case, pointing specifically to the striatum, but on PD case, the SAG is showing on irrelevant parts of the image. Confusion matrix shows that there is almost no wrong prediction.

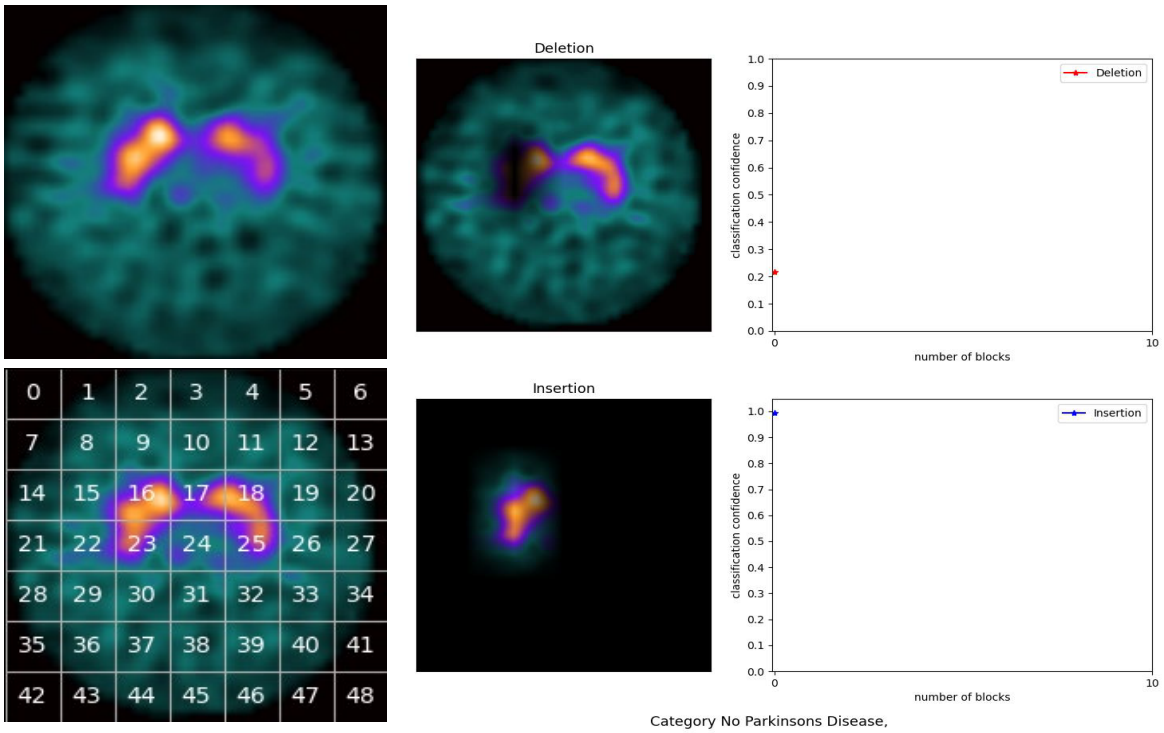

Figure 42. Top left, the original image of NPD case. Bottom left, the grid image. the insertion and deletion image for the third root with model VGG3.

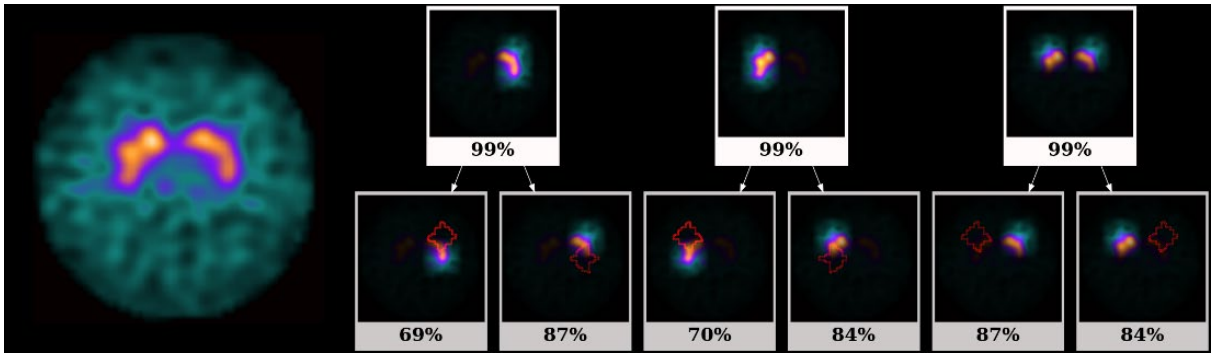

Figure 43. The Structured Attention Image of the NPD case with the model VGG3.

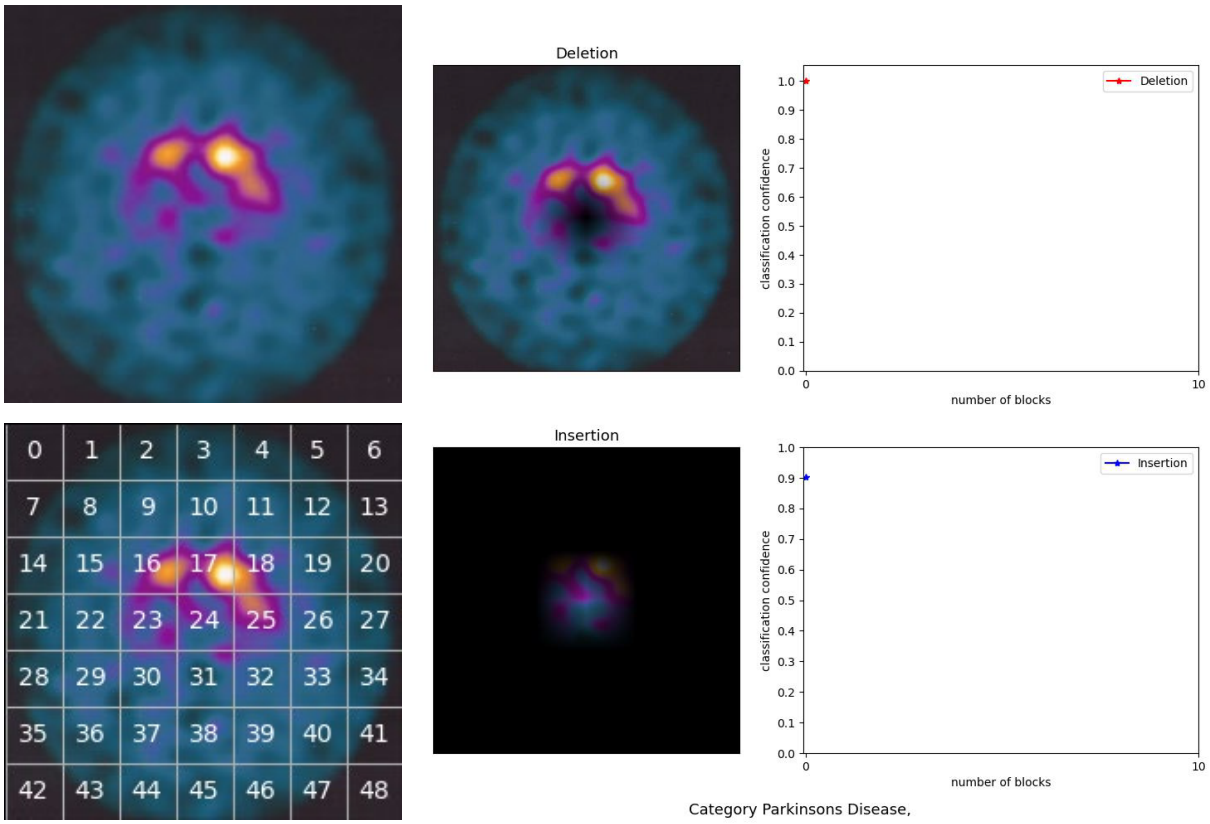

Figure 44. Top left, the original image of PD case. Bottom left, the grid image. the insertion and deletion image for the third root with model VGG3.

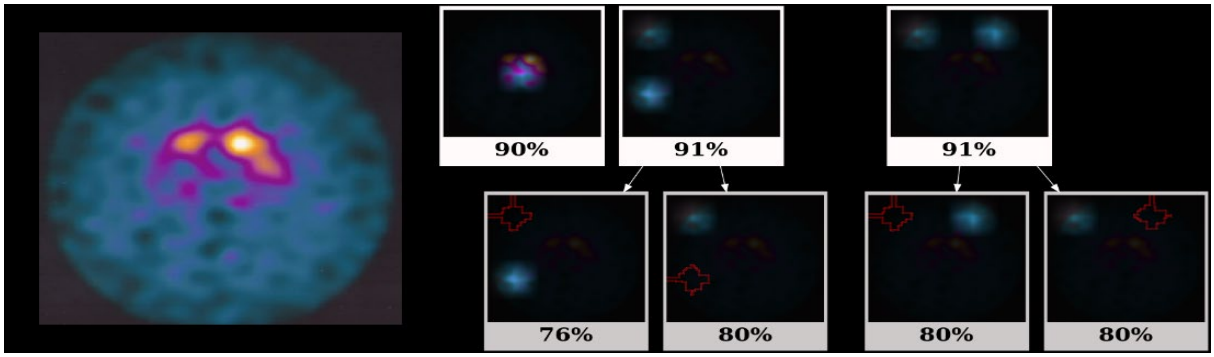

Figure 45. The Structured Attention Image of the PD case with the model VGG3.

3.2.3.4 SAG Results with Model VGG1 and Probability Threshold at 97%

By raising the probability threshold, we observe that there are cases in which the SAG in both classes, focuses directly on the parts of the image where the useful information is placed, meaning the area of striatum. Furthermore, when the nodes are depicting a meaningless patch, the confidence level is relatively reduced, and the SAG does not go deeper. This indicates that the model knows where the information is.

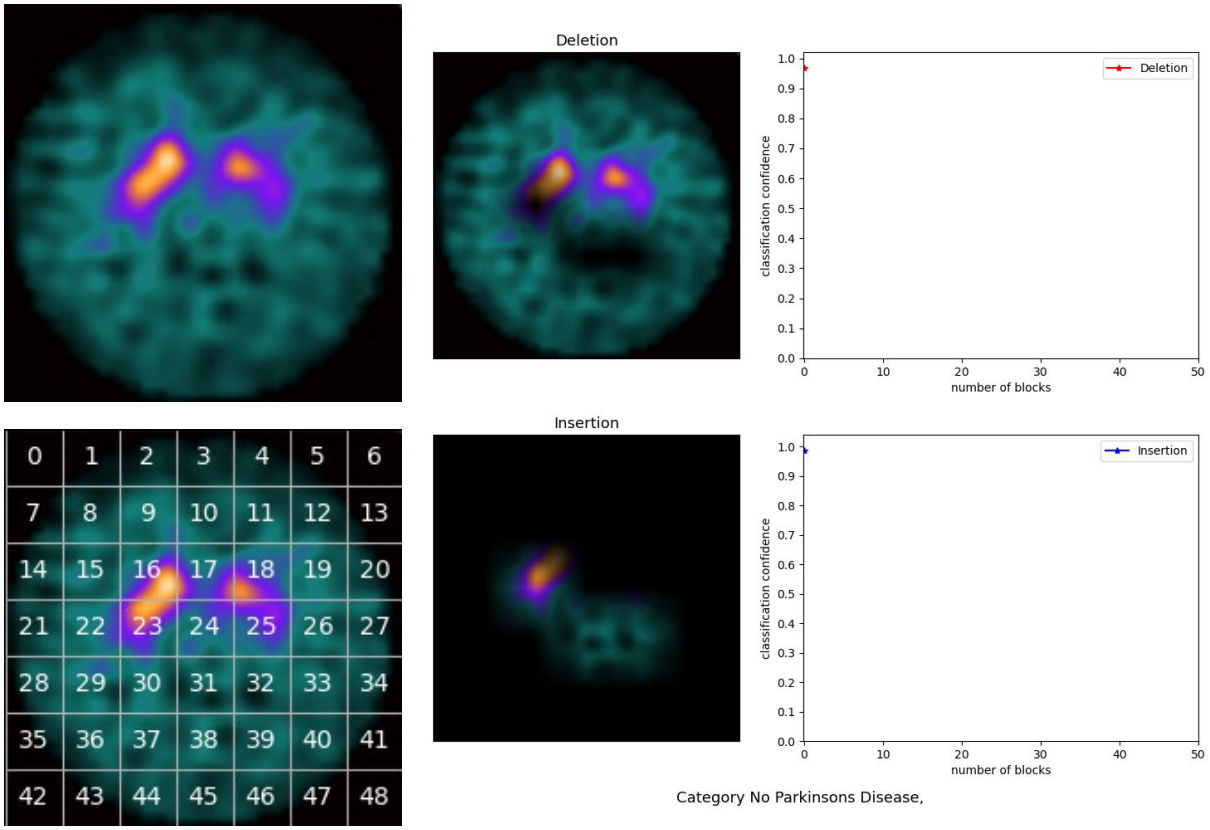

Figure 46. Top left, the original image of NPD case. Bottom left, the grid image. the insertion and deletion image for the third root with model VGG1.

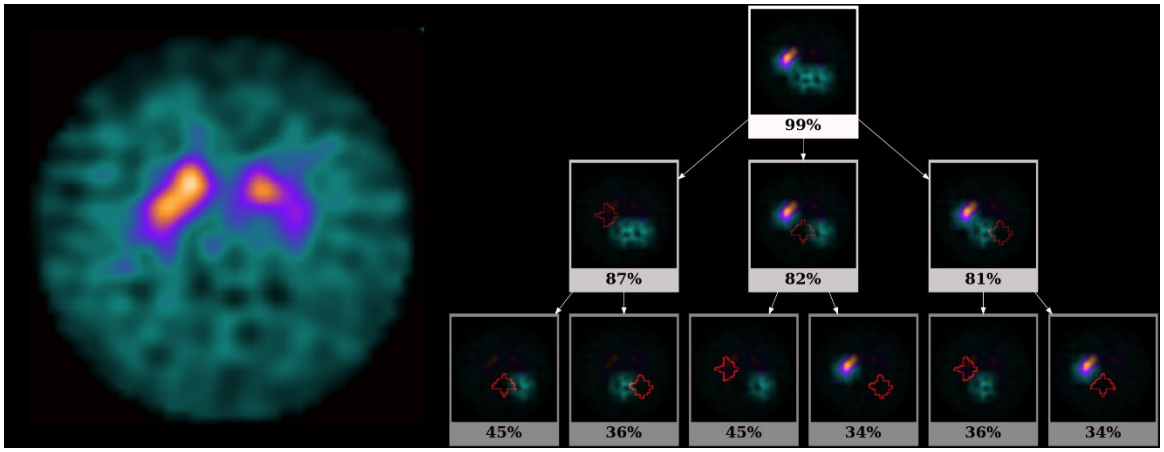

Figure 47. The Structured Attention Image of the PD case, with VGG1.

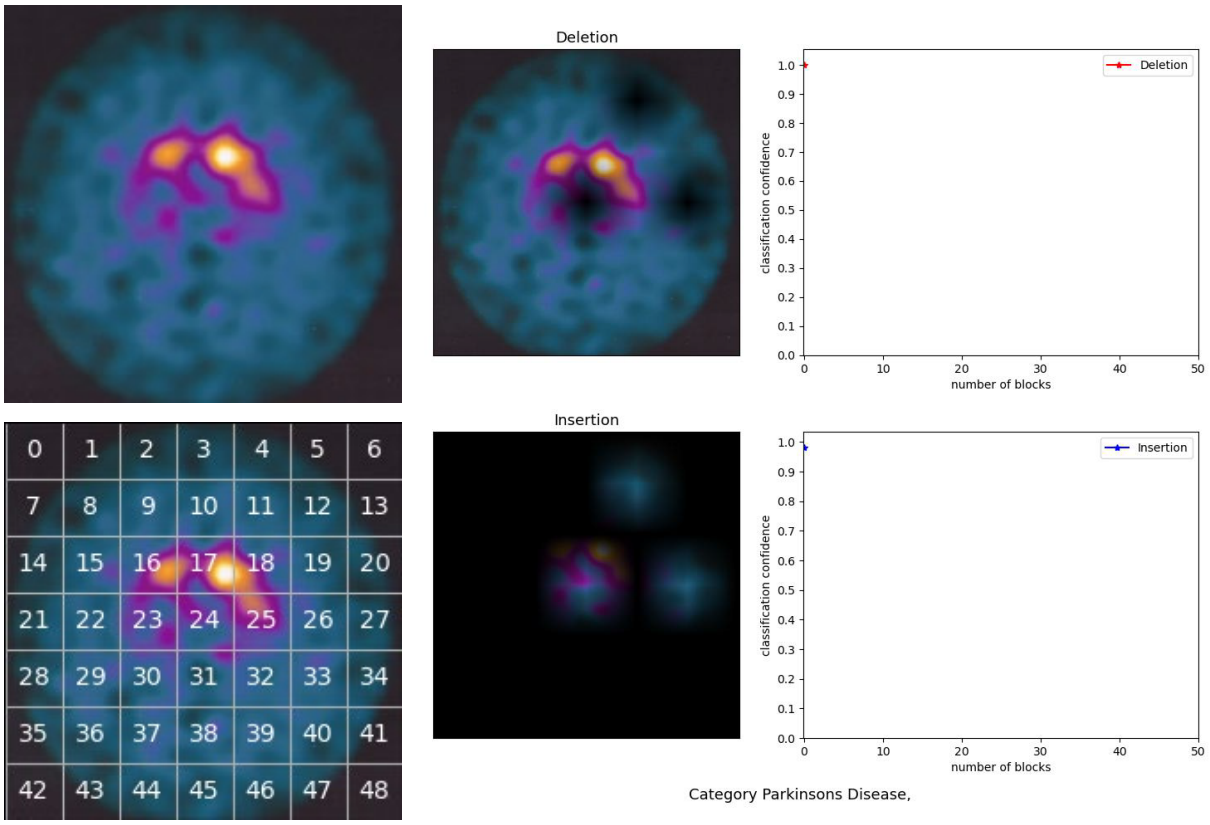

Figure 48. Top left, the original image of PD case. Bottom left, the grid image. the insertion and deletion image for the third root with model VGG1.

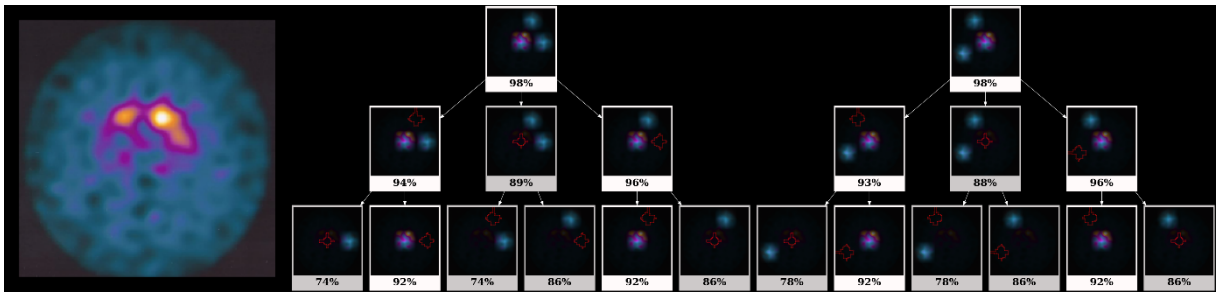

Figure 49. The Structured Attention Image of the PD case, with VGG1.

3.2.3.5 SAG Results with Model VGG2 and Probability Threshold at 97%

Here, as in the previous section the results we get are better in quality compared to the results with probability threshold at 90%. Specifically in PD case, the SAG algorithm, makes a better search. The downside here is that it has great confidence in irrelevant parts of the image, but again, it does not go any further in these patches, indicating that it again knows where is the information inside the image.

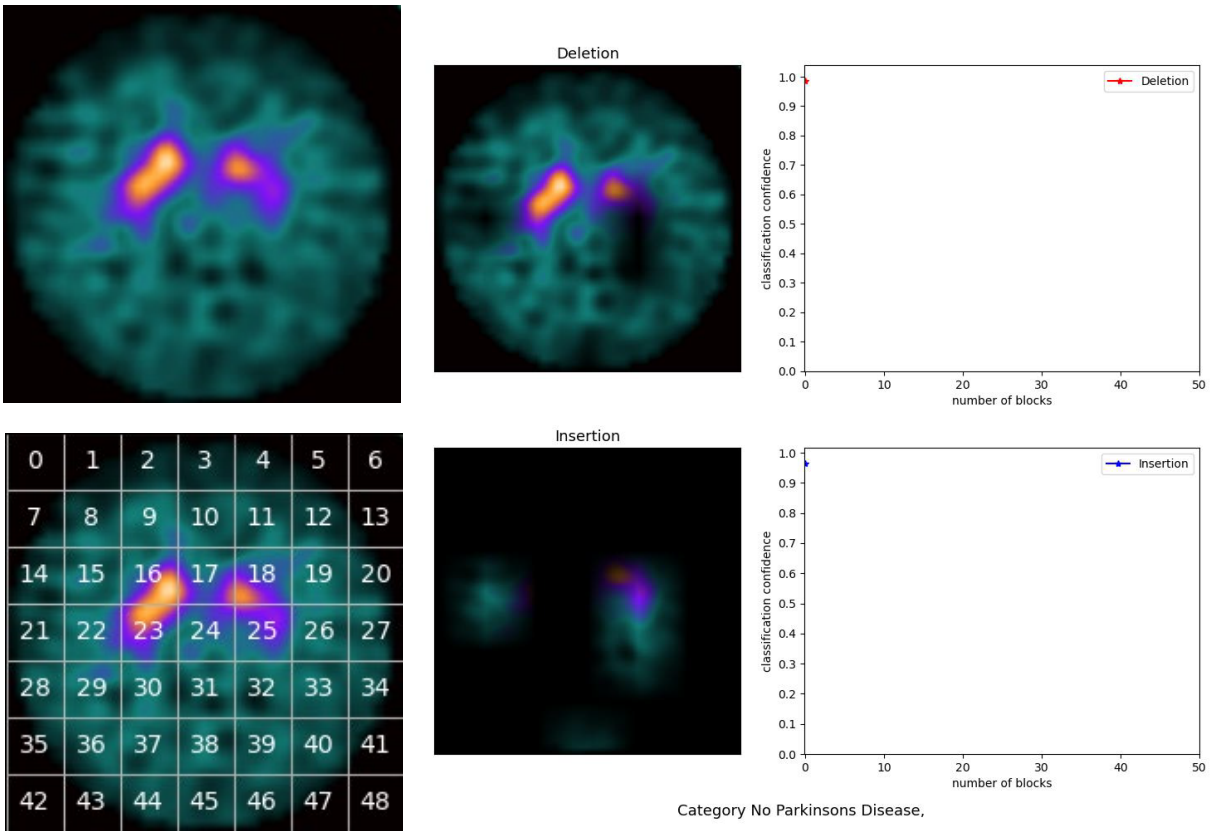

Figure 50. Top left, the original image of NPD case. Bottom left, the grid image. the insertion and deletion image for the third root with model VGG2.

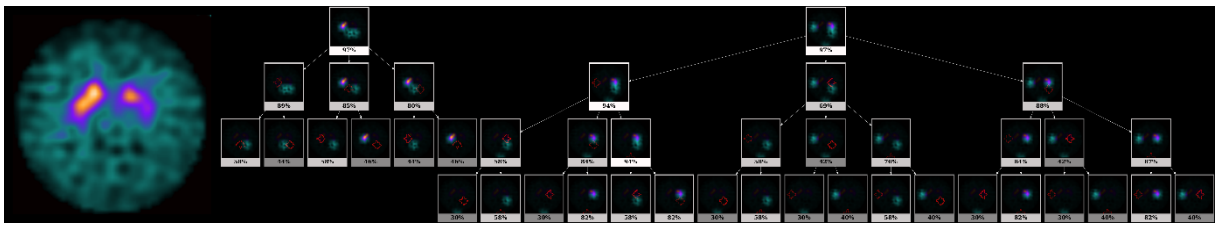

Figure 51. The Structured Attention Image of the NPD case, with VGG2.

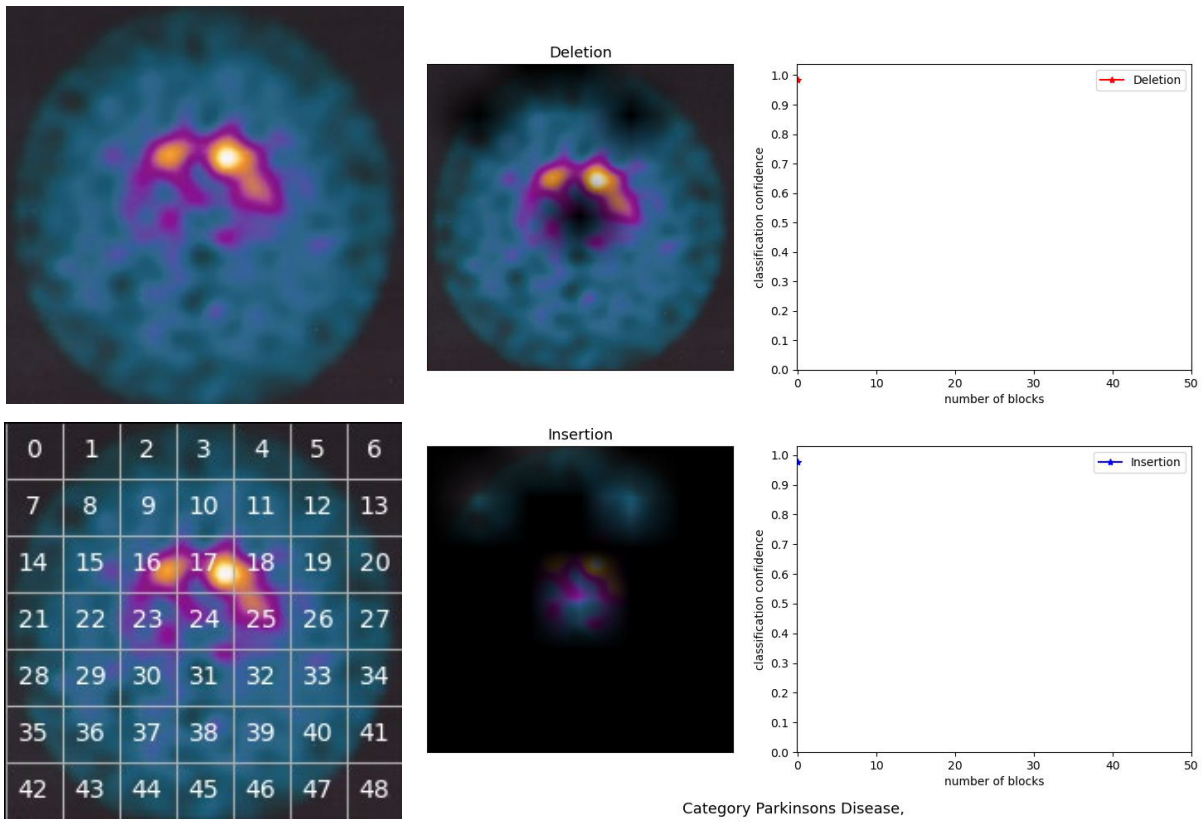

Figure 52. Top left, the original image of PD case. Bottom left, the grid image. the insertion and deletion image for the third root with model VGG2.

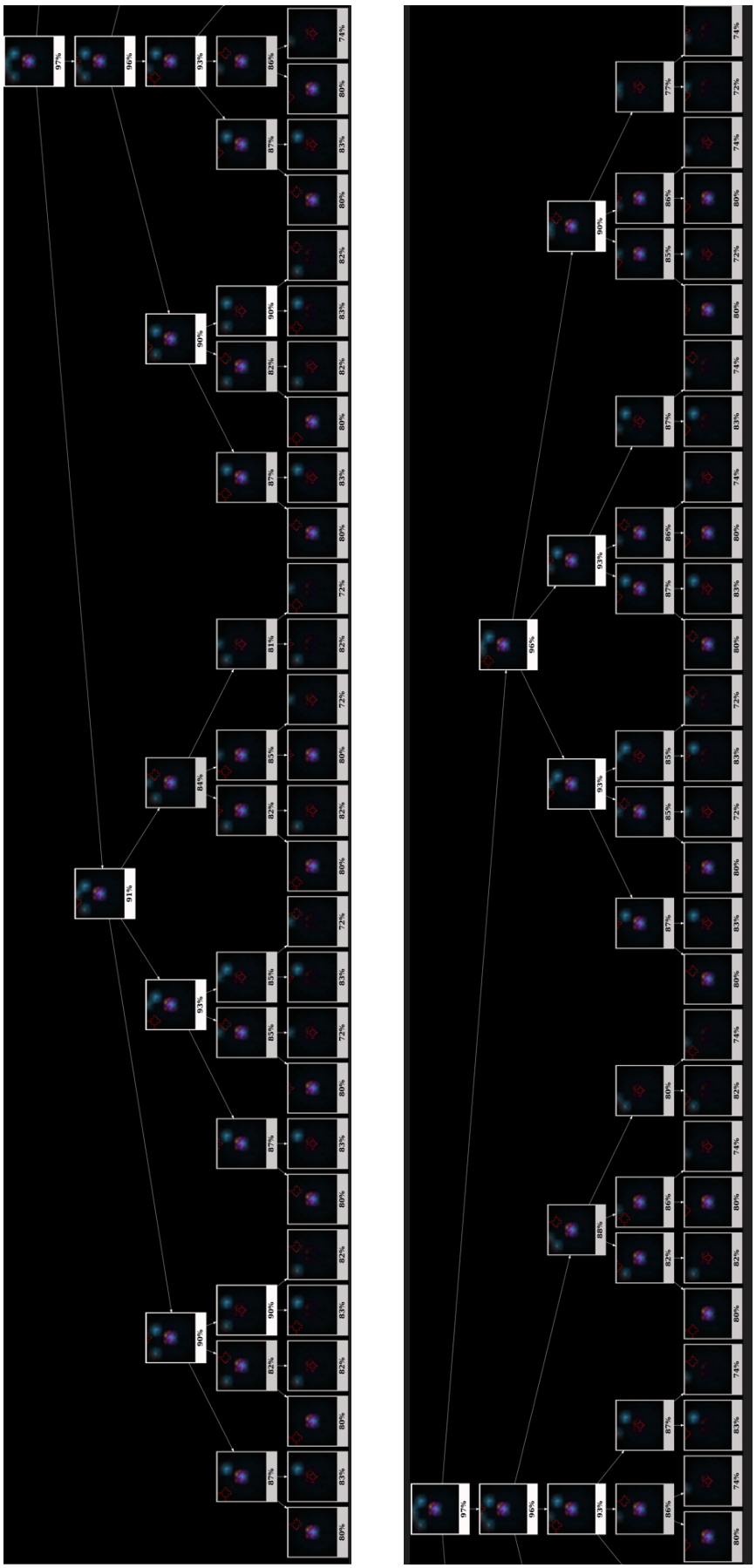

Figure 53. The Structured Attention Image of the PD case, with VGG2.

3.2.3.6 Results on the Original Dataset with grad-CAM Evaluation

In this section we show our research results with regards to grad-CAM. This is done in order to have a comparison against SAG and to figure if the results point to similar directions. This also helps to get an indication if the model is trained properly. Research on this algorithm helped us to understand and narrow down the field of search between the training and the explainability implementation regarding the possible root causes. In this case, all seem to point to the right direction, except model VGG2, and this indicates that the models are looking to the right parts of the image where the striatum is placed. Nevertheless, this experiment let us gain confidence with regards to the training model algorithm. Comparing the two algorithms we could say that we are equally confident on both results, which both, more or less, pointing towards the right direction.

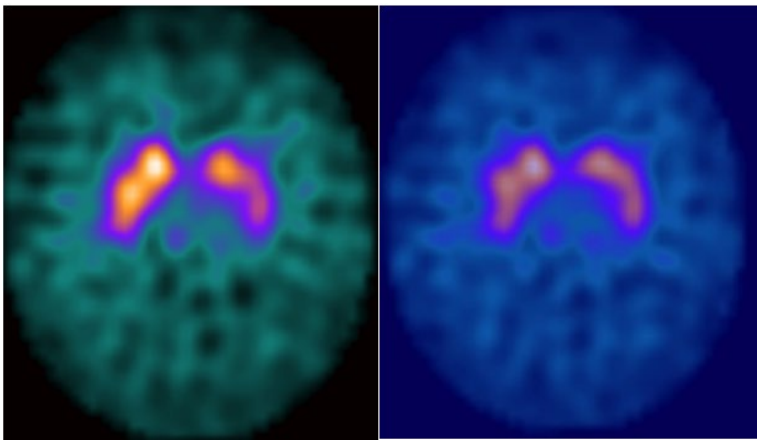

Figure 54. The original image of the NPD compared to the grad-CAM output with model VGG1.

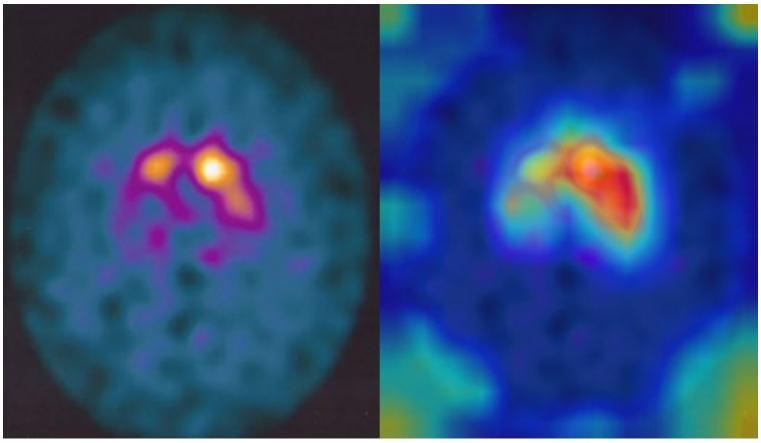

Figure 55. The original image of the PD case compared to the grad-CAM output with model VGG1.

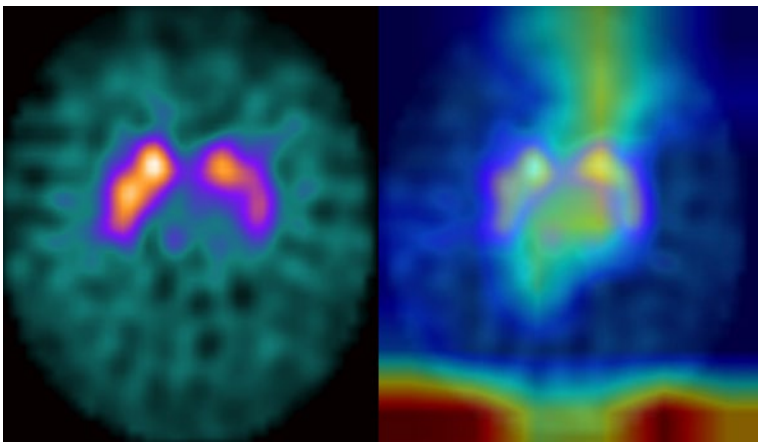

Figure 56. The original image of the NPD case compared to the grad-CAM output VGG2.

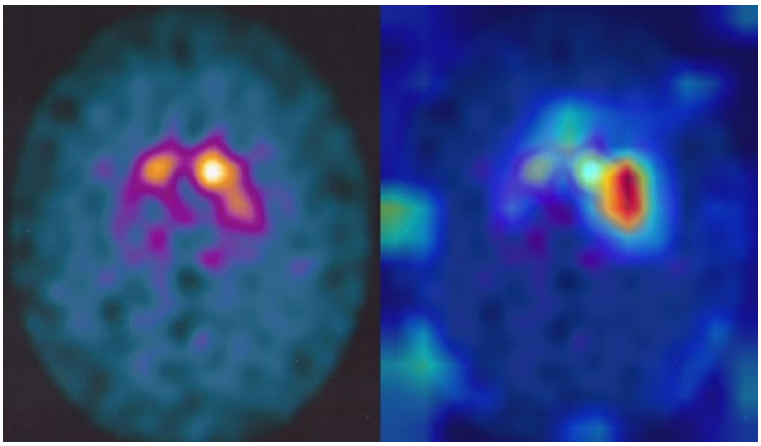

Figure 57. The original image of the PD compared to the grad-CAM output with model VGG2.

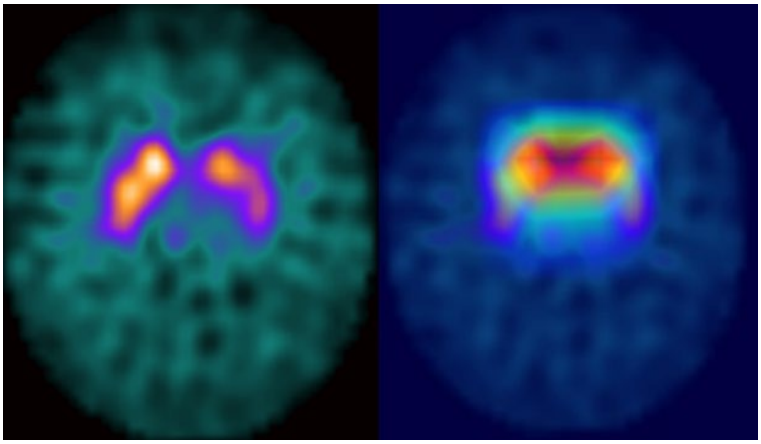

Figure 58. The original image of the NPD compared to the grad-CAM output with model VGG3.

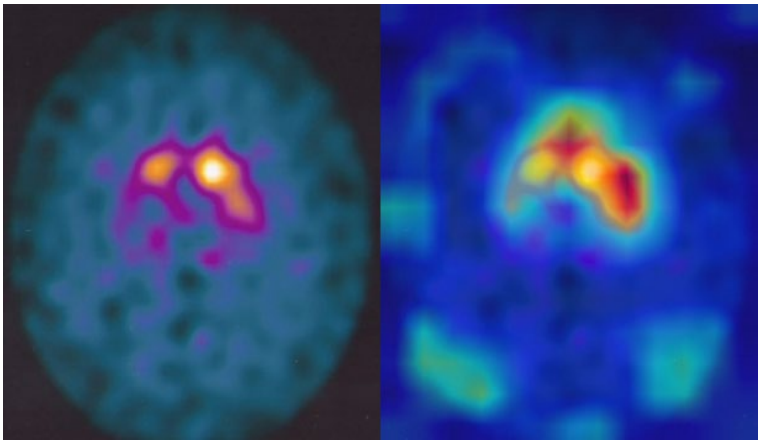

Figure 59. The original image of the PD case compared to the grad-CAM output with model VGG3.

Chapter 4
Epilogue

4.1 Summary

The aim of this thesis is to analyse the explainability techniques that can be applied in order to aid artificial intelligent methods, and as an extent, cognitive systems, to be more transparent with regards to their decision of an outcome. We discussed the importance of Explainable AI systems, we made a brief overview of the explainability techniques in deep learning systems, we mentioned the importance of the images as informational datapoints in medical sector and we defined the scope of the thesis. Then, we focused both on theoretical and technical point of view with regards to deep learning systems, and briefly discussed the discipline's course of history until today. Additionally, we analysed different deep learning architectures, and we focused on convolutional neural networks, and especially on Visual Geometry Group Networks (VGG), which are the most widely used neural network architectures for image processing. Furthermore, we discussed the various explainability techniques that have been developed in order to shed light with regards to the prediction that a deep learning system takes. Some of the most notable techniques are Class Activation Mapping, Grad-Class Activation Mapping, High-Resolution Class Activation Mapping, and Structured Attention Graphs. In this thesis we mainly focus on Structured Attention Graphs and on gradient-weighted Class Activation Mapping. Also, we analysed the implementation and creation of a convolutional neural network, and we utilized the Pytorch capabilities to build a generalized code which could build models based on a given dataset. The way this code is built is capable to receive any dataset sorted in folders and based on this to create a neural network ready for deployment.

The concept of this thesis is to adapt this algorithm to neural networks trained with specific datasets. In order for an application to have a wide adoption in industry and in cognitive systems, it needs to be adaptable and able to fit in various sectors, such as the medical sector which this thesis focuses on. Thus, we examine the ability of the aforementioned explainability techniques to adapt on prediction models. This could be considered as a measure regarding the generalization of each algorithm upon various models. It is challenging to build an explainability algorithm and examine it upon pretrained models, but it adds more value to the community to generalize and be able to work in customized models. We implemented the explainability techniques in order to create an explainable deep learning system which could classify anonymous patients on whether they have signs of Parkinson's disease or not, based on their corresponding DaTSCAN images. We used the high-level rules that are mentioned in section 2.3, in order to evaluate the performance of our models with regards to the applied explainability techniques. Moreover, the application should be able to visualize the decision of the model, by evaluating parts of the image and passing them through the model again with regards to its classification confidence. As we discussed, many experiments were held and many combinatorial configurations were taken into account in order to create a model which could reach an acceptable and most importantly useful outcome.

4.2 Challenges that were Met

This project was accompanied with many challenges in order to create a functioning work product. First challenge was to build an algorithm which could generalize with regards to the creation of a neural model. This necessity came from the fact that in order to find the necessary parameters, properties and changes that are needed to be made to create a model fit for the algorithm, many and different experiments should be implemented.

Next challenge was the maintenance of the dataset. The dataset was small in size. This created some challenges with regards to the training of the network, as in normal circumstances we need to have a vast amount of datapoints to properly train a convolutional neural network. In order to surpass this challenge, we used various techniques regarding the training, such as data augmentation, and transfer learning. These are some of the most popular techniques that help you deal with such kind of problems.

Another issue in training was the method and the definition of the hyperparameters value. For training we used multiple methods such as simple training with finite epochs, training with early stopping, training with learning rate schedulers, and 10-fold validation checking. With regards to the hyperparameters, we experimented on learning rate value, batch size of the data-loaders, the architecture and nearly every parameter that we could benefit from. We experimented on 3 main architectures, a "shallow" neural network, ResNet-50 and VGG-19.

In order to keep track which combination provides better results, and mostly why, a performance recording method

should be developed and implemented, and that is why we developed the metrics.py script, in order to help us to record the performance (Appendix A.6). Also, it should be considered what kind of statistics and metrics adds value to these performance reports. This allows to assess each and every model and pick the best performed ones.

Since we have to deal with models trained upon specific dataset, it is reasonable to assume that the explainability algorithms need to be adjusted to our needs, not only to run properly, but also to keep track of the statistics of the model. This was one of the major challenges, due to the fact that the algorithm not only needed to be customized in order to run the model, but also, to run properly with acceptable results.

4.3 Results

Results of the model with data augmentation were disappointing and in many cases the SAG algorithm crashed, as the augmented datapoints made things worse instead of helping the model perform better. The reason could be on the structure of the information inside the datapoints, when we distort the colors, the useful information is gone. Furthermore, the random rotation of the image showed that the performance was deteriorating as the angle increased, as well as the random crop. This indicates that not only the color played important role in the datapoint information, but also the position and the slope. This part of the experiment helped us to understand that in order to have at least adequate performance on the model training and on results with SAG, we need to treat the datapoints entirely differently with regards to the usual datapoints augmentation we utilize in images that depict a three-dimensional space.

Comparing the results between the three different architecture that were used, we can easily notice that the model architecture we built, and ResNet-50 performed poorly. On the contrary VGG-19 showed better performance in every experiment against the other architectures, even in situations where the performance was overall inadequate (e.g., data augmentation). These results directed our overall work to focus on VGG-19 in order to generate acceptable results.

Working mainly with VGG-19 model architecture, we experimented on various training methods. Simple transformations which can be seen in appendix A, unit A.2 were used. Simple training consisted of a finite number of epochs and experimenting on various hyperparameter configuration. As it was depicted in previous chapter, and overcoming the various challenges that were mentioned in detail in previous unit and in chapter 3.2.1, the results show that the model could focus in the right information for datapoints labeled as NPD (Non-Parkinson's disease patient). In the generated structured attention graphs, we observed that the model focuses on striatum and based on these pieces of the image predicts correct. Nevertheless, this was not the case with PD (Parkinson's disease patient). Despite the correct predictions of the model, the structured attention graphs in some cases were not focused on the damaged striatum, but in irrelevant places, e.g., the 'edges' of the skull. Below we will focus on the investigation on the reasons that the model predicts correct and with great confidence, but it focuses on the wrong parts of the image in some cases.

One assumption, was the training method. For this reason, we experimented with different hyperparametric variables, although this approach did not add much value and did not help on the SAG issue with regards to the PD labels. Another approach to tackle the issue was to use early stopping, which helped a lot on the increase of the model performance, and letting the model run without crashing the SAG, but despite this improvement, we still faced issues on SAG generation. Additionally, we tried to build models with 10-fold validation checking combined with early stopping, and again, the performance was great, but the SAG generation was not performing as well on PD cases. Furthermore, suspicion was raised on the dataset, which was small. For this issue, as we aforementioned, we tried data augmentation with no better results.

All the aforementioned issues, once they got fixed, excluded the possibility that the root cause of inadequate results, was entirely a fault of the development of the model. Then we turn our focus to the functionalities of the SAG generation algorithm. We experimented with various cases by tweaking the hyperparameters of the algorithm, and we observed something interesting. Once we set the probability threshold to 97%, the SAG for both cases is detailed and focused on the right parts of the image. This indicates that the SAG algorithm takes into consideration the roots that they have a confidence level closer to the probability threshold. The increase of the probability threshold, forced the SAG algorithm to narrow down its options, and to focus on the important parts of the image. Furthermore, this change improved the results, regarding the quality and the depth of the nodes in structured attention graphs. Of course, there were patches with no information in which the model showed great confidence level, nevertheless

the SAG algorithm did not go any deeper to analyze these irrelevant patches.

In order to compare the SAG results with a different technique, we run the models on the grad-CAM algorithm, results in chapter 3.2 show that for all the models except the ones trained upon augmented datapoints, we receive acceptable outcomes, pointing on the right direction of the information. Grad-CAM used mostly as a “sanity-check” to see whether or not the results of SAG agree on the quality of the answer. This helped us to narrow down the field of search between the training and the explainability implementation regarding the possible root causes. Even if the focus of this thesis was mainly in structured attention graphs, grad-CAM added also a lot of value to this research.

This work evaluated the possibility and the capability for these explainability techniques to be used in the field of medical sector. Obviously, the SAG algorithm has been built for research purposes which add much value to the expanding field of explainability in neural networks. Of course, in order for such implementation to be applied in commercial and industrial environments, continuous developing and testing needs to be done. Overall, results and work show that such techniques could potentially be able to provide helpful information, to medical research and doctors who specialize in the detection of Parkinson’s disease.

4.4 Final Thoughts and Future Work

This thesis shows the importance of explainability that we need in cognitive systems in order to start using them more as assistants than as tools. Of course, image processing is a sub-part of a cognitive assistant, and there are many modules and processes that could take place in general, in order to have an assistant. Nevertheless, in order for such systems to be trustworthy and be useful in critical tasks of critical sectors, we need to create them as transparent, as ethical and as fair as possible. Future work could include the improvement of the SAG algorithm regarding the customization and the adaptability. Additionally, it could include the analysis of future explainability techniques, or even the combination of those in order to increase the desirable characteristics that such systems need to have in order to be included in critical decision and classification problems. Last but not least, future work could include the development of explainability technique algorithms that are capable to adapt in a wide range of convolutional neural networks, providing sufficient outcomes depending on the sector that we are working.

Appendix A
Code for Creation of the Model

The code is available here:
https://github.com/karvo/ouc-cogsys-custom-structured-attention-graphs

A.1 The Hyperparameters Setup (config.py)

```
import time
import torch
from torchvision import transforms

class Hyperparameters():
    # In case more hyperparameters are added,
    # make sure to add them in the dictionary
    def __init__(self):
        print("Configuring the Hyperparameters...")
        #MODEL PARAMETERS
        model_architecture_lst = ['vgg19', 'resnet50', 'custom']
        # Define the model architecture
        self.model_architecture = model_architecture_lst[0]
        # Define if you want the custom model architecture to be pre-trained.
        #For this application this is set True always
        self.pretrained_model = True
        # Define the device for calculation.
        # If GPU is available, model is trained there, else in CPU
        self.device = torch.device("cuda:0" if torch.cuda.is_available() \
                                           else "cpu")

        # DIRECTORIES
        # Unique ID for the new model
        self.unique_id = time.strftime("%Y%m%d_%H%M%S")
        self.model_name =  self.model_architecture + "_" + self.unique_id
        saved_model_root_path = \
        r'/media/kv/Documents/git/mtkvcs-saved-models/'
        # Create new path for the model
        self.saved_model_path = saved_model_root_path + self.model_name + r'/

        # Define filename model based on the time of creation
        self.saved_model_filename = self.saved_model_path + \
        self.model_architecture + r'_' + time.strftime("%Y%m%d_%H%M%S") + \
        r'.pt'
        # Define the dataset path
        self.root = r'/media/kv/Documents/git/mtkvcs-dataset/datscan/'
```

```
# DATASET PARAMETERS
# Batch size of the dataloader
self.batch_size = 64
# Define if the train data will be shuffle. This is set always to True
self.train_dataset_shuffle = True

#OPTIMIZER PARAMETERS
# Select the optimizer
self.optimizer_function = 'Adam Optimizer'
# Define the learning rate
self.learning_rate = 0.0001
#LOSS FUNCTION PARAMETERS
# Define the loss function
self.loss_function_type = 'Cross Entropy Loss Function'

# LEARNING SCHEDULER PARAMETERS
self.early_stopping_on = False
self.steplr_on = not self.early_stopping_on
# Define the learning rate scheduler
lr_list = ['Step Learning Rate', 'Reduce LR On Plateau']
self.learning_scheduler_type = lr_list[1]
self.gamma = 0.1
self.learning_scheduler_step_size = 3

# TRAINING PARAMETERS
self.epochs = 12

self.hp = {
        # DIRECTORIES
        '\nMODEL DIRECTORIES' : "",
        'unique_id' : self.unique_id,
        'model_name': self.model_name,
        'saved_model_path' : self.saved_model_path,
        'saved_model_filename' : self.saved_model_filename,
        'dataset_path' : self.root,

        # MODEL PARAMETERS
        '\nMODEL PARAMETERS' : "",
        'model_architecture' : self.model_architecture,
        'pretrained_model' : self.pretrained_model,
        'device': self.device,

        #OPTIMIZER PARAMETERS
        '\nOPTIMIZER PARAMETERS' : "",
        'optimizer_function' : self.optimizer_function,
        'learning_rate' : self.learning_rate,
```

```
            #LOSS FUNCTION PARAMETERS
            '\nLOSS FUNCTION PARAMETERS' : "",
            'loss_function_type': self.loss_function_type,

            # LEARNING SCHEDULER PARAMETERS

            '\nLEARNING SCHEDULER PARAMETERS' : "",
            'early_stopping_on' : self.early_stopping_on,
            'learning_scheduler_on' : self.steplr_on,
            'learning_scheduler_type' : self.learning_scheduler_type,
            'gamma' : self.gamma,
            'learning_scheduler_step_size':  \
            self.learning_scheduler_step_size,

            #TRAINING PARAMETERS
            '\nTRAINING PARAMETERS' : "",
            'epochs' : self.epochs,

            #DATASET PARAMETERS
            '\nDATASET PARAMETERS' : "",
            'batch_size' : self.batch_size,
            'dataset_shuffle': self.train_dataset_shuffle
        }

    print("Hyperparameters have been setup!")
```

A.2 The Dataset Setup (dataset.py)

```
import os

from torchvision import transforms
from torchvision import datasets
from torch.utils.data import Dataset, DataLoader, ConcatDataset,
WeightedRandomSampler

class Dataset(Dataset):

    def __init__(self, hp):

        print("Creating the Dataset...")
        # Data directory root for training and testing
        self.root = hp['dataset_path']
        # Batch size of the dataset
        self.batch_size = hp['batch_size']


        self.train_transform = transforms.Compose([
                        # Rotate +/- 10 degrees
                        #transforms.RandomRotation(150),
                        #transforms.RandomAutocontrast(),
                        #transforms.RandomHorizontalFlip(),
                        # Reverse 50% of images
                        transforms.RandomVerticalFlip(),
                        # Resize shortest side to 224 pixels
                        transforms.Resize(256),
                        # Crop longest side to 224
                        # pixels at center
                        transforms.CenterCrop(224),
                        transforms.ToTensor(),
                        # Transforms.ColorJitter(contrast=1),
                        # Transforms.Grayscale(3),
                        transforms.Normalize([0.485, 0.456, 0.406],
                                             [0.229, 0.224, 0.225]),
                        transforms.RandomErasing()
                    ])
```

```
self.test_transform = transforms.Compose([
        # Resize shortest side to 224 pixels
        transforms.Resize(256),
        # Crop longest side to 224 pixels at center
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
            ])
 # Load the images in dataset
 self.train_data=datasets.ImageFolder(os.path.join(self.root,'train'),\
            transform = self.train_transform)
 self.test_data=datasets.ImageFolder(os.path.join(self.root,'test'),\
            transform = self.test_transform)

 # Create the dataloaders
 train_loader = DataLoader(self.train_data,\
               batch_size=hp['batch_size'], shuffle = True)
 test_loader = DataLoader(self.test_data, batch_size=hp['batch_size'],\
                                                        shuffle = True)
 self.dataloader = {'train': train_loader, 'test': test_loader}
 self.train_size = len(self.train_data) # Training dataset size
 self.test_size = len(self.test_data) # Test dataset size
 self.dataset_size = {'train': self.train_size, 'test': self.test_size}
 self.class_names = self.train_data.classes # Class names
 self.dataset_info = {
        # DATASET INFO
        'dataset_size' : self.dataset_size['train'] + \
                         self.dataset_size['test'],
        'training_dataset_size' : self.train_size,
        'testing_dataset_size' : self.test_size,
        'class_names' : self.class_names
 }

 print("================================================")
 print(f"Class names: {self.class_names}")
 print(f"Complete Dataset size: {self.dataset_size['train'] + \
                                 self.dataset_size['test']}")
 print(f"Training Dataset size: {self.train_size}")
 print(f"Testing Dataset size: {self.test_size}")
 print("Dataset Ready!")
 print("================================================")
```

A.3 The Model Setup (model.py)

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models
import torch.nn.functional as F
from torch.optim import lr_scheduler
from early_stopping import EarlyStopping
#from learning_rate_scheduler import LRScheduler

class CustomCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 3, 1)
        self.conv2 = nn.Conv2d(6, 16, 3, 1)
        self.fc1 = nn.Linear(54*54*16, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 2)

    def forward(self, X):
        X = F.relu(self.conv1(X))
        X = F.max_pool2d(X, 2, 2)
        X = F.relu(self.conv2(X))
        X = F.max_pool2d(X, 2, 2)
        X = X.view(-1, 54*54*16)
        X = F.relu(self.fc1(X))
        X = F.relu(self.fc2(X))
        X = self.fc3(X)
        return F.log_softmax(X, dim=1)

class Model(nn.Module):

    def __init__(self, hp, dataset):
        super().__init__()
        self.hp = hp
        self.class_names = dataset.class_names

        if self.hp['model_architecture'] == 'vgg19':
            self.model = models.vgg19(pretrained=self.hp['pretrained_model'])
```

```
        # Initialize the model
        for self.param in self.model.parameters():
            self.param.requires_grad = False

        # Configure the classifier
        self.model.classifier = nn.Sequential(nn.Linear(25088,3136),
                                nn.ReLU(inplace=True),
                                nn.Dropout(0.5),
                                nn.Linear(3136,3136, bias=True),
                                nn.ReLU(inplace=True),
                                nn.Dropout(0.5),
                                nn.Linear(3136,len(self.class_names),\
                                bias=True))
        # note: Newly constructed layer has requires_grad=True by default.
        # You don't need to do it manually.
        self.model_parameters = self.model.classifier.parameters()


    elif self.hp['model_architecture']=='resnet50':

        self.model = \
        models.resnet50(pretrained=self.hp['pretrained_model'])

        for self.param in self.model.parameters():
            self.param.requires_grad = False

        self.model.fc = nn.Linear(2048,len(self.class_names), bias=True)

        self.model_parameters = self.model.fc.parameters()
        # note: Newly constructed layer has requires_grad=True by default.
        # You don't need to do it manually.

    elif self.hp['model_architecture']=='custom':
        self.model = CustomCNN()
        self.model_parameters = self.model.parameters()

    print(f'Current model exists in {"GPU" if torch.cuda.is_available()\
                                        else "CPU"}.')
    #Pass model to GPU
    self.model = self.model.to(self.hp['device'])
    # Define loss function
    self.criterion = nn.CrossEntropyLoss()
    #Define the optimizer
    self.optimizer = optim.Adam(self.model_parameters,\
                lr=self.hp['learning_rate'])
```

```
if hp['learning_scheduler_on']:
    if hp['learning_scheduler_type'] == 'Step Learning Rate':
        self.exp_lr_scheduler = lr_scheduler.StepLR(self.optimizer,\
                                step_size=4, gamma=0.1)
        #self.exp_lr_scheduler = LRScheduler(self.optimizer)
    elif hp['learning_scheduler_type'] == 'Reduce LR On Plateau':
        self.exp_lr_scheduler = \
        lr_scheduler.ReduceLROnPlateau(self.optimizer, patience=5, \
                                                verbose=True)
else:
    self.exp_lr_scheduler = None

if hp['early_stopping_on']:
    self.estop = EarlyStopping()
else:
    self.estop = None

print("Optimizer: Adam *HARDCODED*")
print("Loss function: Cross Entropy *HARDCODED*")
```

A.4 Training, Testing and Recording Statistics and Saving(train.py)

```
import time
import torch
import numpy as np
import pandas as pd
import seaborn as sn
from tqdm import tqdm
import torch.nn as nn
from save import save_epoch, save_fold
from metrics import Metrics
import torch.optim as optim
from numpy.random import randn
import matplotlib.pyplot as plt
from torch.optim import lr_scheduler
from sklearn.model_selection import KFold
from torch.utils.data import Dataset, DataLoader, ConcatDataset,
SubsetRandomSampler
from matplotlib.backends.backend_pdf import PdfPages
from torchvision import datasets, models, transforms
from sklearn.metrics import f1_score, confusion_matrix
from dataset import Dataset
from model import Model
from torch.utils.tensorboard import SummaryWriter

def train_epoch(model, dataloaders, criterion, optimizer, dataset_size,
class_names, metrics, hp,epoch, saved_model_dir):

    for phase in ['train', 'test']:
        if phase == 'train':
            # Set model to training mode
            print("Training mode...")
            model.train()
        else:
            # Set model to evaluate mode
            print("Testing mode...")
            model.eval()

        running_loss = 0.0   # Initialize Loss rate for current epoch
        running_corrects = 0 # Initialize correct prediction of current epoch
      # Initialize empty list to store phase predictions
        all_preds = torch.tensor([]).to(hp['device'])
      # Initialize empty list to store phase predictions
        all_labels = torch.tensor([]).to(hp['device'])
```

```
        # Run the training batches
        for inputs, labels in tqdm(dataloaders[phase]):
            # Inputs are passed to device (GPU or CPU)
            inputs = inputs.to(hp['device'])
            # Outputs are passed to device (GPU or CPU)
            labels = labels.to(hp['device'])

            # If we are in train phase, enable gradient tracking,
            # else disable it
            with torch.set_grad_enabled(phase == 'train'):
                # Pass the input batch to the model
                outputs = model(inputs)
                # Get the prediction in the form of 0/1
                _, predictions = torch.max(outputs.data, 1)
                loss = criterion(outputs, labels)

            if phase == 'train':
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

            # Update the statistics
            all_preds = torch.cat((all_preds, predictions))
            all_labels = torch.cat((all_labels, labels))
            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(predictions == labels.data)

        epoch_loss = running_loss / dataset_size[phase]
        epoch_acc = running_corrects.double() / dataset_size[phase]
        print(f'{phase} Loss: {epoch_loss:.4f} Accuracy: {epoch_acc:.4f}')

        # Record the statistics of each phase
        metrics.get_metrics(all_labels,all_preds, class_names, epoch, \
                                    epoch_acc, epoch_loss,phase,all_preds)

        if phase == 'test':
            save_epoch(model, saved_model_dir,hp['model_architecture'], \
                                            str(epoch), hp['unique_id'] )

    return epoch_loss
```

```
def train(hp, model, dataset, dataset_size, class_names, optimizer, criterion,
exp_lr_scheduler=None, estop=None):

    print("Starting training...")
    dataloaders = dataset.dataloader
    #Initialize metrics
    metrics = Metrics(hp)
    #Start timer
    start_time = time.time()
    # Initialize training parameter for keeping statistics
    model = model.to(hp['device'])
    estop_status = False

    for epoch in range(hp['epochs']):

        if estop_status:
            break

        print('=' * 50)
        print(f"Epoch {epoch}/{hp['epochs'] - 1}")
        print('=' * 50)

        epoch_loss = train_epoch(model, dataloaders, criterion, optimizer, \
             dataset_size, class_names, metrics, hp,epoch,\
                                                 hp['saved_model_path'])

        if hp['learning_scheduler_on']:
            exp_lr_scheduler.step(epoch_loss)
        elif hp['early_stopping_on']:
            if estop(model, epoch_loss): estop_status = True
            print(f"Early stopping status: {estop.status}")

        # Collect the epoch performance
        metrics.collect_epoch_performance(epoch)

    metrics.gather_all_metrics(hp, dataset.dataset_info, str(model))

    total_time = time.time() - start_time
    print(f'Training complete in {total_time // 60:.0f}m \
                                                 {total_time % 60:.0f}s')
    print(f"Best val Acc: \
{metrics.performance['testing_accuracy_datapoints'].loc[metrics.best_epoch]},\
                                            in epoch: {metrics.best_epoch}")
```

```
def kfold_train(hp, dataset, dataset_size, class_names):

    kfold_dataset = ConcatDataset([dataset.train_data, dataset.test_data])


    k=10
    splits=KFold(n_splits=k,shuffle=True,random_state=42)
    for fold, (train_idx,val_idx) in
enumerate(splits.split(np.arange(len(kfold_dataset)))):
        model = Model(hp,dataset)
        model = model.to(hp['device'])

        train_sampler = SubsetRandomSampler(train_idx)
        test_sampler = SubsetRandomSampler(val_idx)
        train_loader = DataLoader(kfold_dataset, batch_size=hp['batch_size'],
sampler=train_sampler)
        test_loader = DataLoader(kfold_dataset, batch_size=hp['batch_size'],
sampler=test_sampler)
        dataloaders = {'train': train_loader, 'test': test_loader}
        train_size = len(train_sampler) # Training dataset size
        test_size = len(test_sampler) # Test dataset size

        dataset_size = {'train': train_size, 'test': test_size}
        start_time = time.time() #Start timer
        print('Fold {}'.format(fold + 1))
        model_dir = save_fold(hp['saved_model_path'],hp['model_architecture'],
fold, hp['unique_id'])

        metrics = Metrics(hp, fold_path=model_dir.split('/')[-2]+ r'/')

        estop_status = False

        for epoch in range(hp['epochs']):

            print('=' * 50)
            print(f"Epoch {epoch}/{hp['epochs'] - 1}")
            print('=' * 50)

            if estop_status:
                break

            epoch_loss = train_epoch(model.model, dataloaders, \
             model.criterion, model.optimizer, dataset_size, class_names, \
                                                    metrics, hp,epoch,model_dir)
```

```
            # Collect the epoch performance
            metrics.collect_epoch_performance(epoch)

            if hp['learning_scheduler_on']:
                model.exp_lr_scheduler.step(epoch_loss)
            elif hp['early_stopping_on']:
                if model.estop(model, epoch_loss): estop_status = True
                print(f"Early stopping status: {model.estop.status}")

        metrics.gather_all_metrics(hp, dataset.dataset_info, str(model))

    total_time = time.time() - start_time
    print(f'Training complete in {total_time // 60:.0f}m {total_time % \
                                                            60:.0f}s')
    print(f"Best val Acc:
{metrics.performance['testing_accuracy_datapoints'].loc[metrics.best_epoch]},\
                                          in epoch: {metrics.best_epoch}")
```

A.5 Save the Model (save.py)

```
def save_epoch(model, folder, architecture, epoch, unique_id):
    path = architecture + r'_' + unique_id + "_epoch_" + epoch
    os.makedirs(folder+path)
    torch.save(model.state_dict(), folder + path + r'/'+ path + r'.pt')

def save_fold(folder, architecture, fold, unique_id):
    path = architecture + r'_' + unique_id + "_fold_" + str(fold) +r'/'
    os.makedirs(folder+path)

    return folder+path
```

A.6 Record the Performance and Create a Report (metrics.py)

```
import numpy as np
import pandas as pd
import seaborn as sn
from fpdf import FPDF
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.metrics import f1_score, confusion_matrix

from torch.utils.tensorboard import SummaryWriter

class Metrics():

    def __init__(self, hp, fold_path = ""):
        self.fold_path = fold_path
        self.writer = SummaryWriter(hp['saved_model_path']+self.fold_path)
        # Start Tensorboard to track metrics
        self.training_cm_pdf = PdfPages(hp['saved_model_path'] + \
                        self.fold_path + 'Training_Confusion_Matrices.pdf')
        self.testing_cm_pdf = PdfPages(hp['saved_model_path'] +self.fold_path\
                                    + 'Validation_Confusion_Matrices.pdf')

        self.performance_datapoints = ['training_loss_datapoints', \
                                       'testing_loss_datapoints', \
                                       'training_accuracy_datapoints', \
                                       'testing_accuracy_datapoints', \
                                       'training_f1_score_datapoints', \
                                       'testing_f1_score_datapoints']

        self.performance = pd.DataFrame(columns=self.performance_datapoints)

        self.best_accuracy = 0.0

    def create_cm(self, labels, predictions, class_names, epoch, epoch_acc,
epoch_loss):
        # Create the Confusion matrix
        self.cm = confusion_matrix(y_true=labels.cpu(),\
                                           y_pred=predictions.cpu())
        self.df_cm = pd.DataFrame(self.cm, index = [i for i in class_names],\
                                    columns = [i for i in class_names])
```

```
plt.figure(figsize = (10,10))
        plt.title(f'Confusion matrix for epoch {epoch}, Accuracy: \
      {"{:.4f}".format(epoch_acc)}, Loss: {"{:.4f}".format(epoch_loss)}',\
                                                              loc='center')
        cm_img = sn.heatmap(self.df_cm, annot=True, fmt='g')
        self.fig = cm_img.get_figure()

    def get_metrics(self,labels, predictions, class_names, epoch, epoch_acc,
epoch_loss,phase,preds):

        # Create the confusion matrix
        self.create_cm(labels, predictions, class_names, epoch, epoch_acc,
epoch_loss)

        if phase == 'train':

            self.training_loss_datapoint = epoch_loss
            self.training_accuracy_datapoint = epoch_acc
            self.training_f1_score_datapoint = f1_score(labels.cpu().data, \
                                        preds.cpu(), average='macro')
            self.training_cm_pdf.savefig(self.fig)

            # Report to Tensorboard Loss, Accuracy, F1-Score
            # and Confusion Matrix of Training epoch
            self.writer.add_scalar('Training Loss', epoch_loss, epoch)
            self.writer.add_scalar('Training Accuracy', epoch_acc, epoch)
            self.writer.add_scalar('Training F1-Score', \
             f1_score(labels.cpu().data, preds.cpu(), average='macro'), epoch)
            self.writer.add_figure("Training Confusion matrix", self.fig,\
                                                                  epoch)
        else:

            # Record Test Statistics of Epoch
            self.testing_loss_datapoint = epoch_loss
            self.testing_accuracy_datapoint = epoch_acc
            self.testing_f1_score_datapoint = f1_score(labels.cpu().data, \
                                        preds.cpu(), average='macro')
            self.testing_cm_pdf.savefig(self.fig)

            # Report to Tensorboard Loss, Accuracy, F1-Score
            # and Confusion Matrix of Testing epoch
            self.writer.add_scalar('Testing Loss', epoch_loss, epoch)
            self.writer.add_scalar('Testing Accuracy', epoch_acc, epoch)
            self.writer.add_scalar('Testing F1-Score', \
             f1_score(labels.cpu().data, preds.cpu(), average='macro'), epoch)
            self.writer.add_figure("Testing Confusion matrix", self.fig, \
                                                                  epoch)
```

```
if epoch_acc > self.best_accuracy:

                self.best_epoch = epoch
                self.best_accuracy = epoch_acc
                self.best_cm = self.fig

    def collect_epoch_performance(self, epoch):
        self.performance.loc[epoch] = [round(self.training_loss_datapoint,3),
round(self.testing_loss_datapoint,3),\
                    round(self.training_accuracy_datapoint.cpu().item(),3),
round(self.testing_accuracy_datapoint.cpu().item(),3),\
                        round(self.training_f1_score_datapoint,3),
round(self.testing_f1_score_datapoint,3)]

    def gather_all_metrics(self,hp, dataset_info, model_architecture):
        self.tags = ['Loss', 'Accuracy', 'F1-Score']
        create_plots(self.performance, self.tags,
hp['saved_model_path']+self.fold_path)

        self.perfomance_recordings = {
                            'perfomance_data' : self.performance,
                            'best_epoch': self.best_epoch,
                            'best_accuracy': self.best_accuracy,
                            'cm_image' : hp['saved_model_path'] +
self.fold_path + "best_cm.jpg",
                            'loss_image': hp['saved_model_path'] +
self.fold_path + self.tags[0] + ".jpg",
                            'acc_image': hp['saved_model_path'] +
self.fold_path + self.tags[1] + ".jpg",
                            'f1_image' : hp['saved_model_path'] +
self.fold_path + self.tags[2] + ".jpg"
                        }

        self.training_cm_pdf.close()
        self.testing_cm_pdf.close()
        self.fig.savefig(hp['saved_model_path'] + self.fold_path +
"best_cm.jpg")
        create_report(hp, self.perfomance_recordings, dataset_info, \
                    model_architecture, fold_path=self.fold_path)
```

```
class PDF(FPDF):

    # Create header to pdf file
    def header(self):
        self.set_font('Arial', 'B', 30)
        self.cell(w=190, h = 10, txt = 'CNN REPORT', border = 1, ln = 2, align
= 'C', fill = False, link = '')
        self.ln(10)

    # Create footer to pdf file
    def footer(self):
        # Go to 1.5 cm from bottom
        self.set_y(-15)
        # Select Arial italic 8
        self.set_font('Arial', 'I', 8)
        # Print current and total page numbers
        self.cell(0, 10, 'Page %s' % self.page_no() + '/{nb}', 0, 0, 'C')

    # Create a chapter file
    def chapter_title(self, num, label):
        # Arial 12
        self.set_font('Arial', '', 12)
        # Background color
        self.set_fill_color(200, 220, 255)
        # Title
        self.cell(0, 6, 'Chapter %d : %s' % (num, label), 0, 1, 'L', 1)
        # Line break
        self.ln(4)

    # Create a chapter body
    def chapter_body(self, name):
        # Read text file
        try:
            with open(name, 'rb') as fh:
                txt = fh.read().decode('latin-1')
        except:
            txt=name

        # Times 12
        self.set_font('Times', '', 12)
        # Output justified text
        self.multi_cell(0, 5, txt)
        # Line break
        self.ln()
```

```
# Print a chapter
    def print_chapter(self, num, title, name):
        self.add_page()
        self.chapter_title(num, title)
        self.chapter_body(name)

    # Create a plot
    def plot_chart(df, tag, saved_model_path):
        df.plot(x ='Epochs', y=tag, kind = 'line')
        plt.savefig( saved_model_path + tag + '.jpg')
        plt.clf()
        plt.close()

def create_report(hp, perfomance_recordings, dataset_info, model, fold_path =
""):

    print("Creating Report...")
    if fold_path != '':
        fold = r'_' + fold_path.split('/')[0].split('_')[-2] + r'_' + \
                                    fold_path.split('/')[0].split('_')[-1]
    else:
        fold = fold_path

    report_name = hp['saved_model_path'] + fold_path + 'model_report_' + \
                                    hp['unique_id'] + fold + '.pdf'
    HEIGHT = 297
    WIDTH = 210
    pdf = PDF()

    data = perfomance_recordings['perfomance_data']
    best_epoch = perfomance_recordings['best_epoch']

    # Data info preparation

    chapter_1_model_info = ""
    chapter_2_performance_info = ""

    for key, value in hp.items():
        name = key.replace("_"," ")
        chapter_1_model_info = chapter_1_model_info + name[0].upper() +\
                                     name[1:] + ': ' + str(value) + "\n"

    for key, value in dataset_info.items():
        name = key.replace("_"," ")
        chapter_1_model_info = chapter_1_model_info + name[0].upper() + \
                                    name[1:] + ': ' + str(value) + "\n"
```

```
chapter_2_performance_info = f"Best model perfomance occured in epoch
{best_epoch}, where the following statitics were held:\n\n Training accuracy:
{data['training_accuracy_datapoints'].loc[best_epoch]}\n\n Training loss:
{data['training_loss_datapoints'].loc[best_epoch]}\n\n Training F1
Score{data['training_f1_score_datapoints'].loc[best_epoch]}\n\n Validation
accuracy: {data['testing_accuracy_datapoints'].loc[best_epoch]}\n\n Validation
loss: {data['testing_loss_datapoints'].loc[best_epoch]}\n\n Validation F1
Score: {data['testing_f1_score_datapoints'].loc[best_epoch]}\n\n"

    # CHAPTER 1
    pdf.set_title("custom neural model report")
    pdf.print_chapter(1, 'Model Summary', chapter_1_model_info)
    pdf.ln(10)

    #CHAPTER 2
    pdf.print_chapter(2, 'Model Architecture', model)
    pdf.ln(10)

    # CHAPTER 3
    pdf.print_chapter(3, 'Confusion Matrix', chapter_2_performance_info)
    pdf.ln(10)
    pdf.image(perfomance_recordings['cm_image'], x = 25, y = 105, w = \
                               WIDTH/1.2, h = 0, type = 'JPG', link = '')

    #CHAPTER 4
    pdf.print_chapter(4, 'Model Metrics', '')
    pdf.ln(10)

    pdf.image(perfomance_recordings['acc_image'], x = 0, y = 40, w = WIDTH, \
                                    h = HEIGHT/2.8, type = 'JPG', link = '')
    pdf.ln(10)

    pdf.image(perfomance_recordings['loss_image'], x = 0, y = 150, w = WIDTH,\
                                    h = HEIGHT/2.8, type = 'JPG', link = '')

    pdf.add_page()
    pdf.image(perfomance_recordings['f1_image'], x = 0, y = 40, w = WIDTH,\
                               h = HEIGHT/2.8, type = 'JPG', link = '')

    pdf.footer()
    pdf.alias_nb_pages()
    pdf.output(report_name)

    print(f"Report Ready! Saved at: {report_name}")
    print(hp['model_name'])
```

```
def create_plots(dataframe, tags, path):

    columns = iter(dataframe)

    for index, column in enumerate(columns):
        plot = dataframe.plot(y=[column,next(columns)],
                              title = tags[index] + " perfomance",
                              grid=True,
                              xlabel = 'Epochs',
                              ylabel = tags[index],
                              kind = 'line',
                              figsize=(20, 10))

        plot.title.set_size(18)
        fig = plot.get_figure()
        fig.savefig(path + tags[index] + '.jpg')
```

A.7 The Main Function (main.py)

```
import os
from train import save_epoch
from train import train, kfold_train
#from save import save_model
from metrics import create_report
from model import Model
from dataset import Dataset
from config import Hyperparameters
import tensorboard

def main():

    # 1. Define the hyperparameters of the custom CNN (config.py)
    config_setup = Hyperparameters()

    # 2. Create and prepare the dataseft in order to be model-compatible
(dataset.py)
    dataset = Dataset(config_setup.hp)

    # 3. Create, configure and modify the model properly  (model.py)
    model = Model(config_setup.hp, dataset)

    # 4. Train the model and record the statistics (train.py)
    train(config_setup.hp, model.model, dataset, dataset.dataset_size,
dataset.class_names, model.optimizer, model.criterion, model.exp_lr_scheduler,
model.estop)
    kfold_train(config_setup.hp, dataset, dataset.dataset_size,
dataset.class_names)

    # 5. Upload model metrics to tensorboard and create the appropriate url
    #os.system("tensorboard dev upload --logdir \ " +
config_setup.hp['saved_model_path'])

if __name__ == "__main__":
    main()
```

Appendix B
Structured Attention Graphs Code

The code is available here:
https://github.com/karvo/ouc-cogsys-custom-structured-attention-graphs

B.1 Main Script (main_generate_sag.py)

```
from cgitb import handler
import  numpy as np
import itertools
import random
import math
import os
import time
import scipy.io as scio
import datetime
import re
import matplotlib.pyplot as plt
import pylab
import csv
from skimage import transform, filters
from textwrap import wrap
import cv2
import sys
from PIL import Image
from utils import *
from get_perturbation_mask import *
from search import *
from diverse_subset_selection import *
from patch_deletion_tree import *
from custom_functions import purge
import signal
```

```
def main(model_path, folder, results_path):
    print(os.path.dirname(model_path))
    # HYPERPARAMS
    ups = 30
    prob_thresh = 0.97 # note that this is prob factor. So we are considering
                                            #0.9 * full_image_probability
    numCategories = 1
    node_prob_thresh = 40 # minimum score threshold to expand a node in the
                                                                    #sag
    beam_width = 3 # suggested values [3,5,10,15]
    max_num_roots = 10 # upper limit on number of roots obtained via search -
                                            #suggested values [10,20,30]
    overlap_thresh = 1 # number of patches allowed to overlap in roots -
                                            #suggested values [0,1,2]
    numSuccessors = 15 # should be greater or equal to beam_width - 'q'
                                            #hyperparam in the paper
    num_roots_sag = 3 # max number of roots to be displayed in the sag
    hparams = "ups: " + str(ups) + "\n" \
            "prob_thresh: " + str(prob_thresh) + "\n" \
            "numCategories: " + str(numCategories) + "\n" \
            "node_prob_thresh: " + str(node_prob_thresh) + "\n" \
            "beam_width: " + str(beam_width) + "\n" \
            "max_num_roots: " + str(max_num_roots) + "\n" \
            "overlap_thresh: " + str(overlap_thresh) + "\n" \
            "numSuccessors: " + str(numSuccessors) + "\n" \
            "num_roots_sag: " + str(num_roots_sag) + "\n" \
            "==================================\n\n"
    input_folder = 'Images'

    maxRootSize = 10 # max number of patches allowed for a root

    # enable cuda
    use_cuda = 0
    if torch.cuda.is_available():
        use_cuda = 1
    # load DNN model
    print("GPU is", "available" if torch.cuda.is_available() \
                                    else "NOT AVAILABLE")
    model = load_model_new(model_path, use_cuda, folder.split("_")[0])
    print(model)

    logfile = "Results with model: " + model_path.split('/')[-1] + "\n\n"
                                                    # log performance
    print(logfile)
    logfile = logfile + "Parameters:\n" + hparams + '\n'
    logfile = logfile + "Model Performance: \n\n ## | Folder | File |\
                                                Prediction | Run \n"
```

```
images_no_roots_found = 0

    # traverse input image folder
    input_path = './'+input_folder+'/'
    dirs = os.listdir(input_path)

    for idx, d in enumerate(dirs):

        if not os.path.isdir(input_path + d):
            continue
        files = os.listdir(input_path + d)

        total_files = len(files)
        file_counter = 0

        output_path =  r'./Results/' + d + r'/'
        if not os.path.isdir(output_path):
            os.makedirs(output_path)

        pre_existing_result_dirs = os.listdir(output_path)

        # start search
        success_counter = 0
        explained_images = []
        img_label = -1



        for imgname in files:

            signal.signal(signal.SIGALRM, handler)
            signal.alarm(120)


             try:
                # current support for jpg and png image formats
                if imgname.endswith('JPEG') or imgname.endswith('jpg') or \
                                                    imgname.endswith('png'):
                    input_img = input_path + d + '/' + imgname
                    print('imgname:', imgname)
                    imgprefix = imgname.split('.')[0]
                    img_label = -1

                    # check if this file is already processes (results exist)
                    for existing_dir in pre_existing_result_dirs:
                        if imgprefix in existing_dir:
                            print('skipping')
                            continue
```

```
# start time stamp
 start_time = time.time()

 # get low probability blurred image
 img, blurred_img = Get_blurred_img(
                    input_img,
                    img_label,
                    model,
                    resize_shape=(224, 224),
                    Gaussian_param=[51, 50],
                    Median_param=11,
                    blur_type='Black',
                    use_cuda=use_cuda)

 # get top "numCategories" predicted
 # categories with their probabilities
 top_cp = get_topn_categories_probabilities_pairs(img,\
             model, numCategories, use_cuda=use_cuda)

 for category, probability in top_cp:

     # get the ground truth label for the given category
     f_groundtruth = open('./GroundTruth1000.txt')
     category_name = f_groundtruth.readlines()[category]
     category_name = category_name[:-2]
     f_groundtruth.close()
     print("Directory: ",d)
     print(category_name)
     # get perturbation mask
      mask, upsampled_mask = Integrated_Mask(
                             ups,
                             img,
                             blurred_img,
                             model,
                             category,
                             max_iterations=2,
                             integ_iter=20,
                             tv_beta=2,
                             l1_coeff=0.01 * 100,
                             tv_coeff=0.2 * 100,
                             size_init=28,
                             use_cuda=use_cuda)
     # get all DISTINCT roots found via beam search
      roots_mp = beamSearch_topKSuccessors_roots(mask, \
      beam_width, numSuccessors, img, blurred_img,\
      model, category, prob_thresh, probability, \
      max_num_roots, maxRootSize, use_cuda=use_cuda)
```

```
numRoots = len(roots_mp)
print('numRoots_all = ', numRoots)
# get maximal set of non-overlapping roots
maximal_Overlap_mp = []
if numRoots > 0:
    maximal_Overlap_mp = \
    maximal_overlapThresh_set(roots_mp, \
                              overlap_thresh)
else:
    images_no_roots_found += 1
numRoots_Overlap = len(maximal_Overlap_mp)
print('numRoots_Overlap = ', numRoots_Overlap)

# prune number of roots to be shown in the sag
if numRoots_Overlap > num_roots_sag:
    maximal_Overlap_mp = \
    maximal_Overlap_mp[:num_roots_sag]
    numRoots_Overlap = num_roots_sag

# end time stamp
end_time = time.time()
# time taken
time_taken = end_time - start_time
time_taken = np.around(time_taken, decimals=3)

# deletion insertion on filtered set of masks -
 # just  to generate result figures
dnf = ""
for mask, ins_prob, rel_prob in maximal_Overlap_mp:
    output_file_videoimgs = imgprefix + '_'
    delloss_top2, insloss_top2,\
    minsufexpmask_upsampled, showimg_buffer =\
    Deletion_Insertion_Comb_withOverlay(

     maxRootSize, mask, model, output_file_videoimgs,\
    img, blurred_img, category=category, \
    use_cuda=use_cuda, \
    blur_mask=0, outputfig=1)

     output_path_img = output_path + imgprefix + \
    "_timetaken_"+\ str(time_taken) + "_category_" + \
    str(category_name)+ "_probthresh_" +\
    str(prob_thresh) + "/" \
    output_file_perturbation_heatmaps =\
    output_path_img + 'perturbation_'
    output_path_count = output_path_img + '_insprob_'\
 + str(ins_prob) + '_relprob_' + str(rel_prob) + "/"
    outvideo_path = output_path_count + 'VIDEO/'
```

```
            # create MDNF expression
            patch_boolean_list = get_patch_boolean(mask)
            conjunction = ""
            for b in patch_boolean_list:
                conjunction += ' & P'+str(b)
                conjunction = conjunction[3:]
                dnf += ' | '+conjunction
            # save obtained sample

            if not os.path.isdir(outvideo_path):
                   os.makedirs(outvideo_path)
            # save perturbation heatmaps
            save_perturbation_heatmap(output_file_perturba
            tion_heatmaps, upsampled_mask, img * 255,
            blurred_img, blur_mask=0)

            # unpack result images
            for item in showimg_buffer:
                 deletion_img, insertion_img, del_curve,\
                 insert_curve, out_pathx, xtick, line_i
            =item
                 out_pathx = outvideo_path + out_pathx
                 showimage(deletion_img, insertion_img,
            del_curve, insert_curve, out_pathx, xtick,
            line_i)

            # save perturbation minsufexp heatmaps
            output_file_perturbationminsufexp_heatmaps =
            output_path_count + imgprefix +
            '_perturbationminsufexp_'
            save_perturbation_heatmap(output_file_perturba
            tionminsufexp_heatmaps,
            minsufexpmask_upsampled, img * 255,
            blurred_img, blur_mask=0)
            insertion_img = cv2.cvtColor(insertion_img,
            cv2.COLOR_RGB2BGR)
            cv2.imwrite(output_path_count + imgprefix +
            'InsertionImg.png', insertion_img * 255)
            # write root conjunctions
            conjunction_file = open(output_path_count +
            imgprefix + 'conjunction.txt', 'w+')
            conjunction_file.write(conjunction)
            conjunction_file.close()

        # write MDNF expression
        dnf_file = open(output_path_img + 'dnf.txt', 'w+')
        dnf = dnf[3:]
        dnf_file.write(dnf)
```

```
## build patch deletion tree ##

# load original image
img_ori = cv2.imread(output_path_img +\
'perturbation_original.png')
img_ori = cv2.cvtColor(img_ori, cv2.COLOR_RGB2BGR)

# create patchImages folder if not exists
current_patchImages_path = output_path_img + \
 'SAG_PatchImages_'+str(numRoots_Overlap)+'roots'
if not os.path.isdir(current_patchImages_path):
    os.makedirs(current_patchImages_path)

# create and save grid image
gridimage(img_ori, output_path_img + 'gridimage.png')

# get set of conjunctions from DNF expression
conjuncts = get_conjuncts_set(dnf)

# prune conjunctions to required number
# of roots in SAG
if len(conjuncts) > numRoots_Overlap:
    conjuncts = conjuncts[:numRoots_Overlap]

# build tree
sag_tree = build_tree(conjuncts, ups, img_ori,\
blurred_img, model,category,current_patchImages_path,\
                    node_prob_thresh, probability)

# book-keeping and save generated result files
f = output_path_img + \
'SAG_'+str(numRoots_Overlap)+'roots.dot'
sag_tree.write(f)
img = pgv.AGraph(f)
img.layout(prog='dot')
f2 = output_path_img + \
'SAG_dag_'+str(numRoots_Overlap)+'roots.png'
img.draw(f2)
img.close()
f3 = output_path_img + 'SAG_final_'+ \
str(numRoots_Overlap)+'roots.png'
img_tree = cv2.imread(f2)
h,w,c = img_tree.shape
hp = h
wp = hp
off = 50
h1 = hp - (off*2)
w1 = h1
```

```
                    dim = (h1,w1)
                    tmp_img_ori = deepcopy(img_ori)
                    tmp_img_ori = cv2.resize(tmp_img_ori, dim,\
                                  interpolation=cv2.INTER_AREA)
                    tmp_img_ori = cv2.cvtColor(tmp_img_ori, \
                                               cv2.COLOR_RGB2BGR)
                    img_ori_padded = np.ones((hp,wp,c)) * 0
                    img_ori_padded[off:off+h1, off:off+w1, :] = \
                                               tmp_img_ori

                    # concatenate image and explanation tree
                    img_sag_final = np.concatenate((img_ori_padded, \
                                               img_tree), axis=1)
                    # save generated sag image
                    cv2.imwrite(f3, img_sag_final)

                file_counter += 1
                print('files processed: {}/{}'.format(file_counter, \
                                                      total_files))
                print('images with no roots found: ', \
                                               images_no_roots_found)

            logfile = logfile + str(idx) + ". |" + d + "|" + imgname +"|"\
                                          + category_name + "| SUCCESS \n"
        except:
            print(imgname, "FAILED!")
                  logfile = logfile + str(idx) + ". | " + d + "| " + \
                    imgname + " | " + category_name + " | FAILURE \n"

with open("./Results/" + folder+"log.txt", 'w') as f:
    f.write(logfile)

import shutil
src_path = "./Results/"
unique_id = time.strftime("%Y%m%d_%H%M%S")
dst_path = results_path + '/results_' + model_path.split('/')\
              [-1].split('.')[0]+ "_prob_" + str(prob_thresh) + "/"
shutil.copytree(src_path, dst_path)
purge(src_path)
```

B.2 Utility Functions (utils.py)

```
import torch
from torch.autograd import Variable
from torchvision import models
import torch.nn as nn
import cv2
import sys
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from skimage import filters
from math import exp
import itertools
import math
import imageio
import os

from custom_functions import load_model

use_cuda = torch.cuda.is_available()
FloatTensor = torch.cuda.FloatTensor if use_cuda else torch.FloatTensor
LongTensor = torch.cuda.LongTensor if use_cuda else torch.LongTensor
Tensor = FloatTensor


def preprocess_image(img, use_cuda=1, require_grad = False):
    means = [0.485, 0.456, 0.406]
    stds = [0.229, 0.224, 0.225]
    preprocessed_img = img.copy()[:, :, ::-1]
    for i in range(3):
        preprocessed_img[:, :, i] = preprocessed_img[:, :, i] - means[i]
        preprocessed_img[:, :, i] = preprocessed_img[:, :, i] / stds[i]
    preprocessed_img = \
        np.ascontiguousarray(np.transpose(preprocessed_img, (2, 0, 1)))
    if use_cuda:
        preprocessed_img_tensor = torch.from_numpy(preprocessed_img).cuda()
    else:
        preprocessed_img_tensor = torch.from_numpy(preprocessed_img)\
                                    preprocessed_img_tensor.unsqueeze_(0)
    return Variable(preprocessed_img_tensor, requires_grad=require_grad)
```

```
def numpy_to_torch(img, use_cuda=1, requires_grad=False):
    if len(img.shape) < 3:
        output = np.float32([img])
    else:
        output = np.transpose(img, (2, 0, 1))
        output = torch.from_numpy(output)
    if use_cuda:
        output = output.cuda()
        output.unsqueeze_(0)
        v = Variable(output, requires_grad=requires_grad)

    return v


def tv_norm(input, tv_beta):
    img = input[0, 0, :]
    row_grad = torch.mean(torch.abs((img[:-1, :] - img[1:, :])).pow(tv_beta))
    col_grad = torch.mean(torch.abs((img[:, :-1] - img[:, 1:])).pow(tv_beta))
    return row_grad + col_grad
def load_model_new(model_path, use_cuda = 1, model_name=''):

    model = load_model(model_path, model_name)
    model.eval()

    if use_cuda:
        model.cuda()

    for p in model.parameters():
        p.requires_grad = False

    return model

def get_topn_categories_probabilities_pairs(img, model, n, use_cuda=use_cuda):
    img = preprocess_image(img, use_cuda, require_grad=False)
    target = torch.nn.Softmax(dim=1)(model(img))
    print(f"target: {target}")
    target = target.squeeze()
    if use_cuda:
        target = target.data.cpu().numpy()
    else:
        target = target.data.numpy()
    topn_categories = np.argsort(-target)[:n]
    #print('top3_categories: ', top3_categories)
    topn_probabilities = [target[x] for x in topn_categories]
    #print('top3_probabilities: ', top3_probabilities)
    top3_cp = zip(topn_categories, topn_probabilities)
    return top3_cp
```

```
def topmaxPixel(HattMap, thre_num):
    ii = np.unravel_index(np.argsort(HattMap.ravel())[: thre_num],
HattMap.shape)
    OutHattMap = HattMap*0
    OutHattMap[ii] = 1
    img_ratio = np.sum(OutHattMap) / OutHattMap.size
    OutHattMap = 1 - OutHattMap
    return OutHattMap, img_ratio

def add_topMaskPixel(current_mask, original_mask):
    ii = np.unravel_index(np.argsort(original_mask.ravel())[0],
original_mask.shape)
    current_mask[ii] = 0
    original_mask[ii] = 1
    img_ratio = np.sum(current_mask) / current_mask.size
    return current_mask, original_mask, img_ratio

def filter_topmaxPixel(HattMap, thre_num):
    ii = np.unravel_index(np.argsort(HattMap.ravel())[: thre_num],
HattMap.shape)
    OutHattMap = np.ones(HattMap.shape)
    OutHattMap[ii] = HattMap[ii]
    img_ratio = np.sum(OutHattMap) / OutHattMap.size
    return OutHattMap, img_ratio

def write_video(inputpath, outputname, img_num, fps = 10):
    fourcc = cv2.VideoWriter_fourcc('M','J','P','G')
    videoWriter = cv2.VideoWriter(outputname, fourcc, fps, (1000, 1000))
    for i in range(img_num):

        img_no = i+1
        #print(inputpath+'video'+str(img_no) +'.jpg')
        img12 = cv2.imread(inputpath+'video'+str(img_no) +'.jpg',1)
        videoWriter.write(img12)
    videoWriter.release()
def save_perturbation_heatmap(output_path, mask, img, blurred, blur_mask=0):
    mask = mask.cpu().data.numpy()[0]
    mask = np.transpose(mask, (1, 2, 0))
    mask = (mask - np.min(mask))
    if not (np.max(mask) == np.min(mask)):
        mask = mask / (np.max(mask)-np.min(mask))
    mask = 1 - mask
    if blur_mask:
        mask = cv2.GaussianBlur(mask, (11, 11), 10)
        mask = np.expand_dims(mask, axis=2)

        heatmap = np.uint8(255 * mask)
        heatmap = np.float32(heatmap) / 255
```

```
    img = np.float32(img) / 255
    perturbated = np.multiply(1 - mask, img) + np.multiply(mask, blurred)
    perturb = 1 * (1 - mask ** 0.8) * img + (mask ** 0.8)* heatmap
    cv2.imwrite(output_path + "original.png", np.uint8(255 * img))
    cv2.imwrite(output_path + "heatmap.png", np.uint8(255 * heatmap))
    cv2.imwrite(output_path + "imposed_heatmap.png", np.uint8(255 * perturb))
    cv2.imwrite(output_path + "blurred.png", np.uint8(255 * blurred))


def create_minsufexp_gif(imagefolder):
    images = []
    dirs = os.listdir(imagefolder)
    for dir in dirs:
        dirpath = imagefolder + '/' + dir
        if os.path.isdir(dirpath):
            #print('d1: ', dirpath)
            files = os.listdir(dirpath)
            for f in files:
                if 'InsertionImg' in f:
                    imagepath = dirpath + '/' + f
                    im = imageio.imread(imagepath)
                    images.append(im)
                    break
    if len(images) > 0:
        imageio.mimsave(imagefolder + 'dnf_gif.gif', images, duration=2.0)
```

B.3 Functions for Combinatorial Search Over Perturbation Mask (search.py)

```
import  numpy as np
import itertools
import random
import math

from utils import *

import os
import time
import scipy.io as scio
import datetime
import re
import matplotlib.pyplot as plt
import pylab
import os
import csv
from skimage import transform, filters
from textwrap import wrap
import cv2
import sys
from PIL import Image
from copy import deepcopy


# beam search - returns all roots found
def beamSearch_topKSuccessors_roots(ref_mask, beam_width, numSuccessors, img,
blurred_img, model, category, prob_thresh, full_image_probability,
max_num_roots, root_size, use_cuda=use_cuda):
    roots_mp = []
    #preprocess image - needed for get_mask_insertion_prob()
    img = preprocess_image(img, use_cuda, require_grad=False)
    blurred_img = preprocess_image(blurred_img, use_cuda, require_grad=False)
    # init

    init_mask = np.ones((1,1,7,7))
    beam_masks = [init_mask]
    # beam_masks = beamSearch_Init(ref_mask, beam_width, numSuccessors)
    # num_patches_inserted += 1
    for i in range(root_size):
        # generate all successors
```

```
        all_successors_mp = beamSearch_get_all_successors_mp(ref_mask,\
        beam_masks, numSuccessors, full_image_probability, img, blurred_img,\
                                    model, category, use_cuda=use_cuda)

        if all_successors_mp == []: # no more successors left
            break
        # select top beam masks and add distict roots if found
        beam_masks, roots_mp = beamSearch_get_topk_masks_roots(roots_mp,\
        all_successors_mp, beam_width, prob_thresh, full_image_probability, \
                                          img, blurred_img, model, category)
        #print('roots_found: ', len(roots_mp))
        if len(roots_mp) > max_num_roots: # max roots limit reached
            break

    return roots_mp

# beam search status function
def beamSearch_topKSuccessors_status(ref_mask, beam_width, numSuccessors, img,
blurred_img, model, category, prob_thresh, full_image_probability,
use_cuda=use_cuda):
    status = False
    num_patches_inserted = 0
    #preprocess image - needed for get_mask_insertion_prob()
    img = preprocess_image(img, use_cuda, require_grad=False)
    blurred_img = preprocess_image(blurred_img, use_cuda, require_grad=False)

    # init
    init_mask = np.ones((1,1,7,7))
    beam_masks = [init_mask]
    # beam_masks = beamSearch_Init(ref_mask, beam_width, numSuccessors)
    # num_patches_inserted += 1
    while status == False:
        # generate all successors
        all_successors_mp = beamSearch_get_all_successors_mp(ref_mask,\
                    beam_masks, numSuccessors, full_image_probability, img,
            blurred_img, model, category, use_cuda=use_cuda)
        if all_successors_mp == []: # no more successors left
            break
        # select top beam masks
        beam_masks, status = beamSearch_get_topk_masks(all_successors_mp,\
                          beam_width, prob_thresh, full_image_probability,
                    img, blurred_img, model, category)
        num_patches_inserted += 1
        # print('num_patches_inserted: ', num_patches_inserted)
    return status, num_patches_inserted
```

```
# init for beam search - select k initialisations from prunned mask
def beamSearch_Init(ref_mask, beam_width, numSuccessors):
    mask_list = []
    init_mask = np.ones((1,1,7,7))
    invalid_indices = np.where(init_mask == 0)
    # get successor indices to select successors from
    successor_indices, num_successors =
beamSearch_get_successor_indices(invalid_indices, ref_mask, numSuccessors)
    # select initial points - draw samples equal to beam_width
    sampled_indices = np.random.choice(np.arange(num_successors),
size=beam_width, replace=False)
    for index in sampled_indices:
        # mask_index = [0, 0, int(index/ncols), index%ncols]
        # print('mask_index: ', mask_index)
        new_mask = np.ones((1,1,7,7))
        new_mask[successor_indices[0][index]][successor_indices[1][index]][suc
cessor_indices[2][index]][successor_indices[3][index]] = 0
        mask_list.append(new_mask)
    return mask_list


# function to get successors - locally optimized using initial perturbation
map as heuristic - since purely random generation of all successors is
computationally expensive (inefficient)
def beamSearch_get_successor_indices(invalid_indices, ref_mask,
numSuccessors):
    successor_indices = []
    # remove invalid indices from ref mask
    ref_mask_copy = deepcopy(ref_mask) # ref mask has continuous values in
interval [0,1]
    ref_mask_copy[invalid_indices] = 2 # value=2 implies invalid patch
    # get successor indices from valid indices
    num_total_indices = ref_mask_copy.shape[2]*ref_mask_copy.shape[3]
    num_valid_indices = num_total_indices - len(invalid_indices[0])
    if numSuccessors > num_valid_indices:
        numSuccessors = num_valid_indices
    successor_indices = np.unravel_index(np.argsort(ref_mask_copy.ravel())[:
numSuccessors], ref_mask_copy.shape)
    # print('numSuccessors_after: ', numSuccessors)
    return successor_indices, numSuccessors

# beam search - function to generate all successor (mask, prob) pairs
def beamSearch_get_all_successors_mp(ref_mask, beam_masks, numSuccessors,
full_image_probability, img, blurred_img, model, category, use_cuda=use_cuda):
    mp_list = []
    for mask in beam_masks:
        invalid_indices = np.where(mask == 0)
        # get successor indices
```

```
        successor_indices, num_successors = \
                        beamSearch_get_successor_indices(invalid_indices,
                    ref_mask, numSuccessors)
      for index in range(num_successors):
            new_mask = deepcopy(mask)
            new_mask[successor_indices[0][index]][successor_indices[1][index]]
[successor_indices[2][index]][successor_indices[3][index]] = 0 # add new patch
to generate successor mask
            insertion_prob = get_mask_insertion_prob(new_mask, img,\
                    blurred_img, model, category, view=0, use_cuda=use_cuda)
            rel_prob = insertion_prob/full_image_probability
            mp_list.append((new_mask, insertion_prob, rel_prob))
    # print('len(mp_list): ', len(mp_list))
    return mp_list

# beam search - function to filter top k successors (k = beam_width)
def beamSearch_get_topk_masks_roots(roots_mp, all_successors_mp, beam_width,
prob_thresh, full_image_probability, img, blurred_img, model, category,
use_cuda=use_cuda):
    mask_list = []
    # sort successors masks in descending order of insertion prob
    all_successors_mp_sorted = sorted(all_successors_mp, key=lambda x: x[1],
reverse=True)
    # select top k successors
    for mask, prob, rel_prob in all_successors_mp_sorted:
        if prob > prob_thresh * full_image_probability:
            # remove extra patches in root
            minimal_mask, minimal_ins_prob = remove_extra_patches(mask, prob,
prob_thresh, full_image_probability, img, blurred_img, model, category,
use_cuda)
            # add root if not duplicate
            if not duplicate(minimal_mask, roots_mp):
                minimal_rel_prob = minimal_ins_prob / full_image_probability
                item_mp = [minimal_mask, minimal_ins_prob, minimal_rel_prob]
                roots_mp.append(item_mp)
        else:
            mask_list.append(mask)
            if len(mask_list) > beam_width:
                break
    return mask_list, roots_mp
```

```
# beam search - function to filter top k successors (k = beam_width)
def beamSearch_get_topk_masks(all_successors_mp, beam_width, prob_thresh,
full_image_probability, img, blurred_img, model, category, use_cuda=use_cuda):
    mask_list = []
    status = False
    # sort successors masks in descending order of insertion prob
    all_successors_mp_sorted = sorted(all_successors_mp, key=lambda x: x[1],
reverse=True)
    # select top k successors
    for i in range(beam_width):
        mask_list.append(all_successors_mp_sorted[i][0])
    # status - True if a mask with insertion_prob > 0.9 * full_prob is found
    top_insertion_prob = all_successors_mp_sorted[0][1]
    if top_insertion_prob > prob_thresh * full_image_probability:
        status = True
    return mask_list, status


# this function returns success if one mask with insertion probability > 0.9
is found
def comb_search_status(mask, total_mask_pixels, max_patches, img, blurred_img,
model, category, prob_thresh, probability, use_cuda=use_cuda):
    success = False

    #preprocess image - needed for get_mask_insertion_prob()
    img = preprocess_image(img, use_cuda, require_grad=False)
    blurred_img = preprocess_image(blurred_img, use_cuda, require_grad=False)

    indices = np.argsort(mask.ravel())[: total_mask_pixels]
    for indices_subset in itertools.combinations(indices, max_patches):
        ii = np.unravel_index(indices_subset, mask.shape)
        outMask = np.ones_like(mask)
        outMask[ii] = 0
        insertion_prob = get_mask_insertion_prob(outMask, img, blurred_img,\
                          model, category, view=0, use_cuda=use_cuda)
        if insertion_prob > (prob_thresh * probability):
            # print('insertion_prob:{} \t \
                  total_prob:{}'.format(insertion_prob, probability))
            success = True
            break
    return success
```

```
# return all nCr combinations of mask for top n pixels
def topCombinations(mask, limit_n, size_r):
    indices = np.argsort(mask.ravel())[: limit_n]
    all_masks = []
    for indices_subset in itertools.combinations(indices, size_r):
        ii = np.unravel_index(indices_subset, mask.shape)
        outMask = np.ones_like(mask)
        outMask[ii] = 0
        all_masks.append(outMask)
    return all_masks

# get probability of masked image
def get_mask_insertion_prob(mask, img, blurred_img, model, category, view=0,
use_cuda=use_cuda):
    resize_wh = (img.data.shape[2], img.data.shape[3])
    if use_cuda:
        upsample = torch.nn.UpsamplingBilinear2d(size=resize_wh).cuda()
    else:
        upsample = torch.nn.UpsamplingBilinear2d(size=resize_wh)
    maskdata = mask.copy()
    # convert to insertion mask
    maskdata = np.subtract(1, maskdata).astype(np.float32)
    if use_cuda:
        Masktop = torch.from_numpy(maskdata).cuda()
    else:
        Masktop = torch.from_numpy(maskdata)
    Masktop = Variable(Masktop, requires_grad=False)
    MasktopLS = upsample(Masktop)
    Img_topLS = img.mul(MasktopLS) + \
                blurred_img.mul(1 - MasktopLS)
    outputstopLS = torch.nn.Softmax(dim=1)(model(Img_topLS))
    insertion_probability = outputstopLS[0,
category].data.cpu().numpy().copy()
    if view:
        print('insertion_probability: ', insertion_probability)
        image = Img_topLS.data.cpu().numpy()
        image = image.squeeze()
        image = image.transpose(1,2,0)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        cv2.imshow('window', image)
        cv2.waitKey(0)
    return insertion_probability
# check for duplicate mask
def duplicate(newmask, all_mp):
    for mask, p, rel_p in all_mp:
        if (mask == newmask).all():
            return True
    return False
```

```
# remove redundant patches - necessary to get monotonic DNF
def remove_extra_patches(mask, pob, pth, full_prob, img, blurred_img, model,
category, use_cuda=use_cuda):
    no_patches_removed_flag = False
    while not no_patches_removed_flag:
        z0, z1, rows, cols = np.where(mask == 0)
        no_patches_removed_flag = True
        for i in range(len(rows)):
            mask2 = mask.copy()
            index = (z0[i], z1[i], rows[i], cols[i])
            mask2[index] = 1
            ptmp = get_mask_insertion_prob(mask2, img, blurred_img, model,
category, 0, use_cuda)
            if ptmp > pth * full_prob:
                mask = mask2
                pob = ptmp
                no_patches_removed_flag = False
    return mask, pob


# get probabilities for all masks when imposed on image
def get_all_mp(all_masks, img_ori, blurred_img_ori, model, category,
prob_thresh, full_prob, use_cuda=use_cuda):
    all_mp = []
    img = preprocess_image(img_ori, use_cuda, require_grad=False)
    blurred_img = preprocess_image(blurred_img_ori, use_cuda,
require_grad=False)
    resize_wh = (img.data.shape[2], img.data.shape[3])
    if use_cuda:
        upsample = torch.nn.UpsamplingBilinear2d(size=resize_wh).cuda()
    else:
        upsample = torch.nn.UpsamplingBilinear2d(size=resize_wh)
    total = len(all_masks)
    count = 0
    for mask in all_masks:
        count += 1
        maskdata = mask.copy()
        # convert to insertion mask
        maskdata = np.subtract(1, maskdata).astype(np.float32)
        if use_cuda:
            Masktop = torch.from_numpy(maskdata).cuda()
        else:
            Masktop = torch.from_numpy(maskdata)
        Masktop = Variable(Masktop, requires_grad=False)
        MasktopLS = upsample(Masktop)
        Img_topLS = img.mul(MasktopLS) + \
                    blurred_img.mul(1 - MasktopLS)
        outputstopLS = torch.nn.Softmax(dim=1)(model(Img_topLS))
```

```
        insertion_probability = outputstopLS[0,
category].data.cpu().numpy().copy()
        if insertion_probability > (prob_thresh * full_prob):
            mask, pob = remove_extra_patches(mask, insertion_probability,
prob_thresh, full_prob, img, blurred_img, model, category, use_cuda)
            if not duplicate(mask, all_mp):
                all_mp.append((mask, insertion_probability))
    return all_mp
```

B.4 Functions to Build the Patch Deletion Tree (patch_deletion_tree.py)

```
import os
import sys
import matplotlib.pyplot as plt
import matplotlib.ticker as plticker
import cv2
import numpy as np
from copy import deepcopy
import pygraphviz as pgv
import torch

from utils import *


def get_patch_boolean(mask):
    boolean = []
    z0, z1, h, w = mask.shape
    z0, z1, rows, cols = np.where(mask == 0)
    for i in range(len(rows)):
        patchname = rows[i]*h + cols[i]*w
        boolean.append(patchname)
    return boolean


def get_edge_mask_red(mask, canny_param, intensity, kernel_size):
    upsampled_mask_newPatch_edge = deepcopy(mask)
    upsampled_mask_newPatch_edge = np.uint8(upsampled_mask_newPatch_edge *
255)
    upsampled_mask_newPatch_edge = cv2.Canny(upsampled_mask_newPatch_edge,
canny_param, canny_param)
    morphkernel = np.ones((25, 25), np.uint8)
    upsampled_mask_newPatch_edge =
cv2.morphologyEx(upsampled_mask_newPatch_edge, cv2.MORPH_CLOSE, morphkernel)
    upsampled_mask_newPatch_edge = cv2.Canny(upsampled_mask_newPatch_edge,
500, 500)
    upsampled_mask_newPatch_edge =
cv2.GaussianBlur(upsampled_mask_newPatch_edge, (kernel_size, kernel_size),
kernel_size-1)
    upsampled_mask_newPatch_edge *= intensity
    upsampled_mask_newPatch_edge =
np.expand_dims(upsampled_mask_newPatch_edge, axis=2)
    return upsampled_mask_newPatch_edge
```

```
def create_node_image(parent_chain, index, edgepatch, parent_prob, ups,
img_ori, blurred_img_ori, model, category, current_patchImages_path,
full_image_probability):
    width, height, channels = img_ori.shape
    resize_wh = (width, height)

    use_cuda = 0
    if torch.cuda.is_available():
        use_cuda = 1
        upsample = torch.nn.UpsamplingBilinear2d(size=resize_wh).cuda()
    else:
        upsample = torch.nn.UpsamplingBilinear2d(size=resize_wh)

    mask_w = int(width/ups)
    mask_h = int(height/ups)
    mask_insertion = np.zeros((mask_w, mask_h))
    mask_edgePatch = np.zeros((mask_w, mask_h))
    wh_mask_oldPatches = np.zeros((mask_w, mask_h))
    wh_mask_newPatch = np.zeros((mask_w, mask_h))
    wh_mask_combined = np.zeros((mask_w, mask_h))

    w = int(mask_w/7)
    h = int(mask_h/7)

    # create edgepatch mask
    edgepatch_flag = edgepatch != ""
    if edgepatch_flag:
        patchnum = int(edgepatch[1:])
        row = int(patchnum/7)
        col = int(patchnum%7)
        mask_edgePatch[row][col] = 1.1

    # create image
    for i in range(0,index+1):
        patchnum = parent_chain[i][0]
        patchnum = int(patchnum[1:])
        row = int(patchnum/7)
        col = int(patchnum%7)
        # y = h*row
        # x = w*col
        if i == index:
            wh_mask_newPatch[row][col] = 1.5 # 2.5
        else:
            wh_mask_oldPatches[row][col] = 1.5 # 2.5
        wh_mask_combined[row][col] = 1.5 # 2
        mask_insertion[row][col] = 1
```

```
    # get mask insertion probability
    mask_insertion = np.expand_dims(mask_insertion, axis=0)
    mask_insertion = np.expand_dims(mask_insertion, axis=0)
    mask_insertion = mask_insertion.astype(np.float32)
    if use_cuda:
        mask_insertion = torch.from_numpy(mask_insertion).cuda()
    else:
        mask_insertion = torch.from_numpy(mask_insertion)
    mask_insertion = Variable(mask_insertion, requires_grad=False)
    upsampled_mask_insertion = upsample(mask_insertion)
    img_ori_copy = deepcopy(img_ori)
    img_ori_copy = cv2.cvtColor(img_ori_copy, cv2.COLOR_BGR2RGB)
    img_ori_copy = np.float32(img_ori_copy) / 255
    blurred_img_copy = deepcopy(blurred_img_ori)
    img_ori_copy = preprocess_image(img_ori_copy, use_cuda=use_cuda,
require_grad=False)
    blurred_img_copy = preprocess_image(blurred_img_copy, use_cuda=use_cuda,
require_grad=False)
    insertion_img = img_ori_copy.mul(upsampled_mask_insertion) +
blurred_img_copy.mul(1-upsampled_mask_insertion)
    prob_vector = torch.nn.Softmax(dim=1)(model(insertion_img))
    if use_cuda:
        ins_prob = prob_vector[0, category].data.cpu().numpy()
    else:
        ins_prob = prob_vector[0, category].data.numpy()

    # convert ins_prob to relative_prob
    ins_prob = ins_prob / full_image_probability

    #create edge patch image
    if edgepatch_flag:
        mask_edgePatch = np.expand_dims(mask_edgePatch, axis=0)
        mask_edgePatch = np.expand_dims(mask_edgePatch, axis=0)
        # if use_cuda:
        #     mask_edgePatch = torch.from_numpy(mask_edgePatch).cuda()
        # else:
        mask_edgePatch = torch.from_numpy(mask_edgePatch)
        upsampled_mask_edgePatch = upsample(mask_edgePatch)
        upsampled_mask_edgePatch = upsampled_mask_edgePatch.data.numpy()
        upsampled_mask_edgePatch = upsampled_mask_edgePatch.squeeze(0)
        upsampled_mask_edgePatch = np.transpose(upsampled_mask_edgePatch,
(1,2,0))
        prob_drop = parent_prob - (ins_prob*100)
        if prob_drop < 20:
            ksize = 3
            intensity = 2
        elif prob_drop < 60:
            ksize = 3
```

```
            intensity = 10
        else:
            ksize = 7
            intensity = 50
             upsampled_mask_edgePatch_edge = \
             get_edge_mask_red(upsampled_mask_edgePatch, 75, intensity, ksize)

    ######################### NOW DO THE WHITE MASK VERSION
#######################
    wh_mask_oldPatches = np.expand_dims(wh_mask_oldPatches, axis=0)
    wh_mask_oldPatches = np.expand_dims(wh_mask_oldPatches, axis=0)
    # if use_cuda:
    #     wh_mask_oldPatches = torch.from_numpy(wh_mask_oldPatches).cuda()
    # else:
    wh_mask_oldPatches = torch.from_numpy(wh_mask_oldPatches)
    wh_upsampled_mask_oldPatches = upsample(wh_mask_oldPatches)
    wh_upsampled_mask_oldPatches = wh_upsampled_mask_oldPatches.data.numpy()
    wh_upsampled_mask_oldPatches = wh_upsampled_mask_oldPatches.squeeze(0)
    wh_upsampled_mask_oldPatches = np.transpose(wh_upsampled_mask_oldPatches,
(1,2,0))

    wh_mask_newPatch = np.expand_dims(wh_mask_newPatch, axis=0)
    wh_mask_newPatch = np.expand_dims(wh_mask_newPatch, axis=0)
    # if use_cuda:
    #     wh_mask_newPatch = torch.from_numpy(wh_mask_newPatch).cuda()
    # else:
    wh_mask_newPatch = torch.from_numpy(wh_mask_newPatch)
    wh_upsampled_mask_newPatch = upsample(wh_mask_newPatch)
    wh_upsampled_mask_newPatch = wh_upsampled_mask_newPatch.data.numpy()
    wh_upsampled_mask_newPatch = wh_upsampled_mask_newPatch.squeeze(0)
    wh_upsampled_mask_newPatch = np.transpose(wh_upsampled_mask_newPatch,
(1,2,0))

    wh_mask_combined = np.expand_dims(wh_mask_combined, axis=0)
    wh_mask_combined = np.expand_dims(wh_mask_combined, axis=0)
    # if use_cuda:
    #     wh_mask_combined = torch.from_numpy(wh_mask_combined).cuda()
    # else:
    wh_mask_combined = torch.from_numpy(wh_mask_combined)
    wh_upsampled_mask_combined = upsample(wh_mask_combined)
    wh_upsampled_mask_combined = wh_upsampled_mask_combined.data.numpy()
    wh_upsampled_mask_combined = wh_upsampled_mask_combined.squeeze(0)
    wh_upsampled_mask_combined = np.transpose(wh_upsampled_mask_combined,
(1,2,0))

    # img_mean = np.ones_like(img_white)
    # img_mean[0] *= int(0.485 * 255)
    # img_mean[1] *= int(0.456 * 255)
```

```
    # img_mean[2] *= int(0.406 * 255)

    # different image colors used
    img_black = np.ones_like(img_ori.shape) * 0
    img_white = np.ones_like(img_ori.shape) * 255
    img_red = np.zeros_like(img_ori)
    img_red[:,:,0] = 255
    img_transparent = img_ori * 0.1 + img_black * 0.9

    # create node image
    patch_image = (img_ori * wh_upsampled_mask_oldPatches) + (img_ori *
wh_upsampled_mask_newPatch) + (img_transparent * (1-
wh_upsampled_mask_combined))
    if edgepatch_flag:
        patch_image += img_red * upsampled_mask_edgePatch_edge

    #save image by uid and return path
    patch_img = np.zeros_like(patch_image)
    for j in range(0,3):
        patch_img[:,:,j] = patch_image[:,:,2-j]
    uid = parent_chain[index][1]
    patch_img_path = current_patchImages_path + '/' + str(uid) + '.png'
    cv2.imwrite(patch_img_path, patch_img)

    # if leaf image, save in leaf images folder - to be used for user study
setup
    # if index == len(parent_chain)-1:
    #     patch_img_leaf_path = current_patchImages_path_usrstudy + '/' +
str(uid) + '.png'
    #     cv2.imwrite(patch_img_leaf_path, patch_img)

    return patch_img_path, ins_prob


# creates and saves grid image
def gridimage(image, savepath):
    my_dpi=100.
    fig=plt.figure(figsize=(float(image.shape[0]/my_dpi),
float(image.shape[1]/my_dpi)), dpi=my_dpi)
    ax=fig.add_subplot(111)
    # Remove whitespace from around the image
    fig.subplots_adjust(left=0,right=1,bottom=0,top=1)
    # Set the gridding interval: here we use the major tick interval
    myInterval=float(image.shape[0]/7)
    loc = plticker.MultipleLocator(base=myInterval)
    ax.xaxis.set_major_locator(loc)
    ax.yaxis.set_major_locator(loc)
    # Add the grid
    ax.grid(which='major', axis='both', linestyle='-')
```

```
    # Add the image
    ax.imshow(image)
    # Find number of gridsquares in x and y direction
    nx=abs(int(float(ax.get_xlim()[1]-ax.get_xlim()[0])/float(myInterval)))
    ny=abs(int(float(ax.get_ylim()[1]-ax.get_ylim()[0])/float(myInterval)))
    # Add some labels to the gridsquares
    for j in range(ny):
        y=myInterval/2+j*myInterval
        for i in range(nx):
            x=myInterval/2.+float(i)*myInterval
            ax.text(x,y,'{:d}'.format(i+j*nx),color='w',ha='center',\
                                                              va='center')
    # Save the figure
    fig.savefig(savepath, dpi=my_dpi)


# returns a list of conjuncts from dnf string
def get_conjuncts_set(dnfstring):
    conjunct_sets = []
    conjuncts = dnfstring.split(' | ')
    for index, conj in enumerate(conjuncts):
        conj = conj.replace('(','') #filter for brackets
        conj = conj.replace(')','')
        literals = conj.split(' & ')
        conjunct_sets.append(literals)
    return conjunct_sets


def get_subconjuncts(conj):
    n = len(conj)
    if n == 1:
        return [] # base case
    subconjlist = []
    if n > 3:
        n = 3
    for i in range(n):
        subconj = deepcopy(conj)
        subconj.remove(conj[i])
        subconjlist.append((subconj, conj[i]))
    return subconjlist
def get_node_bg_color(ins_prob, basecolor):
    if ins_prob > 0.9:
        color = basecolor + '1'
    elif ins_prob > 0.5:
        color = basecolor + '3'
    else:
        color = basecolor + '4'
    return color
```

```
# global uid counter
incremental_uid = 0

# function to create node and add it to the tree
def add_lattice_node(conj, g, edgepatch, parent_prob, ups, img_ori,
blurred_img_ori, model, category, current_patchImages_path, isleaf,
full_image_probability):
    global incremental_uid
    # using global incremental uid
    incremental_uid += 1
    uid = incremental_uid
    # create parent chain
    parent_chain = []
    dummy_uid = uid
    for lit in conj:
        parent_chain.append((lit,dummy_uid))
    dummy_index = len(parent_chain) - 1
    # create node
    shape = 'box'
    style = 'filled'
    penwidth=2
    imagepath, ins_prob = create_node_image(parent_chain, dummy_index,
edgepatch, parent_prob, ups, img_ori, blurred_img_ori, model, category,
current_patchImages_path, full_image_probability)
    # get node background color
    node_bg_color = get_node_bg_color(ins_prob, 'snow')
    fillcolor = node_bg_color
    color = node_bg_color
    # add labels to node
    label = '< <table border="0"> <tr> <td width="2" height="2"
border="0"><img src="' + imagepath + '"/></td> </tr> <tr> <td>'
    ins_prob = np.around(ins_prob, decimals=3)
    ins_prob = int(ins_prob * 100)
    xlabel = str(ins_prob)+'%'
    # different annotation style for leaves
    if isleaf:
        penwidth=3
        group=str(float('-inf'))
        rank=str(float('-inf'))
        xlabel = '<font color="black">' + xlabel + '</font>'
    else:
        group = ''
        rank = ''
        xlabel = '<font color="black">' + xlabel + '</font>'
    xlabel = '<b>' + xlabel + '</b>'
    label += xlabel + '</td> </tr> </table> >'
    # add the node
```

```
    g.add_node(uid, shape=shape, style=style, fillcolor=fillcolor,
label=label, labelloc='b', fontsize=25, fontcolor='white', color=color,
penwidth=penwidth, group=group, rank=rank, width=2.4, height=2.9,
fixedsize='true')
    return uid, imagepath, ins_prob

def recursive_lattice(conjuncts, parent_uid, g, depth, parent_prob, ups,
img_ori, blurred_img_ori, model, category, current_patchImages_path,
node_prob_thresh, full_image_probability):
    # iterate over leaves
    for conj, edgepatch in conjuncts:
        # add node for current leaf
        uid, imagepath, node_prob = add_lattice_node(conj, g, edgepatch,
parent_prob, ups, img_ori, blurred_img_ori, model, category,
current_patchImages_path, False, full_image_probability)
        g.add_edge(parent_uid, uid, color='white')
        # build rest of the lattice originating from current leaf recursively
        if node_prob > node_prob_thresh:
            subconjlist = get_subconjuncts(conj)
            if len(subconjlist) > 0:
                recursive_lattice(subconjlist, uid, g, depth+1, node_prob,
ups, img_ori, blurred_img_ori, model, category, current_patchImages_path,
node_prob_thresh, full_image_probability)


def build_tree(conjuncts, ups, img_ori, blurred_img_ori, model, category,
current_patchImages_path, node_prob_thresh, full_image_probability):
    # initialize graph
    g = pgv.AGraph(directed=True, strict=True, overlap='false',
bgcolor='black')
    edgepatch = ""
    depth = 1
    # iterate over leaves
    for conj in conjuncts:
        # add node for current leaf
        uid, _, node_prob = add_lattice_node(conj, g, edgepatch, 100, ups,
img_ori, blurred_img_ori, model, category, current_patchImages_path, True,
full_image_probability)
        # build rest of the lattice originating from current leaf recursively
        if node_prob > node_prob_thresh:
            subconjlist = get_subconjuncts(conj)
            if len(subconjlist) > 0:
                recursive_lattice(subconjlist, uid, g, depth+1, node_prob,
ups, img_ori, blurred_img_ori, model, category, current_patchImages_path,
node_prob_thresh, full_image_probability)
    return g
```

B.5 Function to Obtain Perturbation Mask (get_pertubation_mask.py)

```
import  numpy as np
import itertools
import random
import math

from utils import *

import os
import time
import scipy.io as scio
import datetime
import re
import matplotlib.pyplot as plt
import pylab
import os
import csv
from skimage import transform, filters
from textwrap import wrap
import cv2
import sys
from PIL import Image
from copy import deepcopy


def Get_blurred_img(input_img, img_label, model, resize_shape=(224, 224),
Gaussian_param = [51, 50], Median_param = 11, blur_type= 'Gaussian', use_cuda
= 1):
    ########################
    # Generate blurred images as the baseline

    # Parameters:
    # -------------
    # input_img: the original input image
    # img_label: the classification target that you want to visualize
(img_label=-1 means the top 1 classification label)
    # model: the model that you want to visualize
    # resize_shape: the input size for the given model
    # Gaussian_param: parameters for Gaussian blur
    # Median_param: parameters for median blur
    # blur_type: Gaussian blur or median blur or mixed blur
    # use_cuda: use gpu (1) or not (0)
    ######################################################
```

```
    original_img = cv2.imread(input_img, 1)
    original_img = cv2.resize(original_img, resize_shape)
    img = np.float32(original_img) / 255
    if blur_type =='Gaussian':   # Gaussian blur
        Kernelsize = Gaussian_param[0]
        SigmaX = Gaussian_param[1]
        blurred_img = cv2.GaussianBlur(img, (Kernelsize, Kernelsize), SigmaX)

    elif blur_type == 'Black':
        blurred_img = img * 0

    elif blur_type == 'Median': # Median blur
        Kernelsize_M = Median_param
        blurred_img = np.float32(cv2.medianBlur(original_img, Kernelsize_M)) /
255

    elif blur_type == 'Mixed': # Mixed blur
        Kernelsize = Gaussian_param[0]
        SigmaX = Gaussian_param[1]
        blurred_img1 = cv2.GaussianBlur(img, (Kernelsize, Kernelsize), SigmaX)
        Kernelsize_M = Median_param
        blurred_img2 = np.float32(cv2.medianBlur(original_img, Kernelsize_M))
/ 255
        blurred_img = (blurred_img1 + blurred_img2) / 2

    return img, blurred_img

def Integrated_Mask(ups, img, blurred_img, model, category, max_iterations =
15, integ_iter = 20,
                    tv_beta=2, l1_coeff = 0.01*300, tv_coeff = 0.2*300,
size_init = 112, use_cuda =1):
    ########################
    # Obtaining perturbation mask using integrated gradient descent to find
the smallest and smoothest area that maximally decrease the
    # output of a deep model

    # Parameters:
    # -------------
    # ups: upsampling factor
    # img: the original input image
    # blurred_img: the baseline for the input image
    # model: the model that you want to visualize
    # category: the classification target that you want to visualize
(category=-1 means the top 1 classification label)
    # max_iterations: the max iterations for the integrated gradient descent
    # integ_iter: how many points you want to use when computing the
integrated gradients
```

```
    # tv_beta: which norm you want to use for the total variation term
    # l1_coeff: parameter for the L1 norm
    # tv_coeff: parameter for the total variation term
    # size_init: the resolution of the mask that you want to generate
    # use_cuda: use gpu (1) or not (0)
    ######################################################
    # preprocess the input image and the baseline (low probability) image
    img = preprocess_image(img, use_cuda, require_grad=False)
    blurred_img = preprocess_image(blurred_img, use_cuda, require_grad=False)
    resize_size = img.data.shape
    resize_wh = (img.data.shape[2], img.data.shape[3])

    # initialize the mask
    mask_init = np.ones((int(resize_wh[0]/ups), int(resize_wh[1]/ups)),
                                                          dtype=np.float32)
    mask = numpy_to_torch(mask_init, use_cuda, requires_grad=True)

    # upsampler
    if use_cuda:
        upsample = torch.nn.UpsamplingBilinear2d(size=resize_wh).cuda()
    else:
        upsample = torch.nn.UpsamplingBilinear2d(size=resize_wh)

    # You can choose any optimizer
    # The optimizer doesn't matter, because we don't need optimizer.step(), we
just use it to compute the gradient
    optimizer = torch.optim.Adam([mask], lr=0.01)

    # containers for curve metrics
    curve1 = np.array([])
    curve2 = np.array([])
    curvetop = np.array([])
    curve_total = np.array([])

    # Integrated gradient descent

    # hyperparams
    alpha = 0.0001
    beta = 0.2
    for i in range(max_iterations):

        upsampled_mask = upsample(mask)
        upsampled_mask = upsampled_mask.expand(1, 3, upsampled_mask.size(2),\
                                               upsampled_mask.size(3))

        # the l1 term and the total variation term
        loss1 = l1_coeff * torch.mean(torch.abs(1 - mask)) + tv_coeff *\
              tv_norm(mask, tv_beta)
```

```
loss_all = loss1.clone()

# compute the perturbed image
        perturbated_input_base = img.mul(upsampled_mask) + blurred_img.mul(1 -
upsampled_mask)

        loss2_ori = torch.nn.Softmax(dim=1)(model(perturbated_input_base))[0,
category] # masking loss (no integrated)

        loss_ori = loss1 + loss2_ori
        if i==0:
            if use_cuda:
                curve1 = np.append(curve1, loss1.data.cpu().numpy())
                curve2 = np.append(curve2, loss2_ori.data.cpu().numpy())
                curvetop = np.append(curvetop, loss2_ori.data.cpu().numpy())
                curve_total = np.append(curve_total,
loss_ori.data.cpu().numpy())
            else:
                curve1 = np.append(curve1, loss1.data.numpy())
                curve2 = np.append(curve2, loss2_ori.data.numpy())
                curvetop = np.append(curvetop, loss2_ori.data.numpy())
                curve_total = np.append(curve_total, loss_ori.data.numpy())
        if use_cuda:
            loss_oridata = loss_ori.data.cpu().numpy()
        else:
            loss_oridata = loss_ori.data.numpy()

        # calculate integrated gradient for next descent step
        for inte_i in range(integ_iter):

            # Use the mask to perturbated the input image.
            integ_mask = 0.0 + ((inte_i + 1.0) / integ_iter) * upsampled_mask
            perturbated_input_integ = img.mul(integ_mask) + blurred_img.mul(1
- integ_mask)

            # add noise
            noise = np.zeros((resize_wh[0], resize_wh[1], 3),
dtype=np.float32)
            noise = noise + cv2.randn(noise, 0, 0.2)
            noise = numpy_to_torch(noise, use_cuda, requires_grad=False)
            perturbated_input = perturbated_input_integ + noise

            outputs = torch.nn.Softmax(dim=1)(model(perturbated_input))
            loss2 = outputs[0, category]
            loss_all = loss_all + loss2/20.0
        # compute the integrated gradients for the given target,
        # and compute the gradient for the l1 term and the total variation
term
        optimizer.zero_grad()
```

```
        loss_all.backward()
        whole_grad = mask.grad.data.clone() # integrated gradient

        # LINE SEARCH with revised Armijo condition

        step = 200.0 # upper limit of step size
        MaskClone = mask.data.clone()
        MaskClone -= step * whole_grad
        MaskClone = Variable(MaskClone, requires_grad=False)
        MaskClone.data.clamp_(0, 1) # clamp the value of mask in [0,1]
        mask_LS = upsample(MaskClone)   # Here the direction is the whole_grad
        Img_LS = img.mul(mask_LS) + blurred_img.mul(1 - mask_LS)
        outputsLS = torch.nn.Softmax(dim=1)(model(Img_LS))
        loss_LS = l1_coeff * torch.mean(torch.abs(1 - MaskClone)) + tv_coeff *
tv_norm(MaskClone, tv_beta) + outputsLS[0, category]

        if use_cuda:
            loss_LSdata = loss_LS.data.cpu().numpy()
        else:
            loss_LSdata = loss_LS.data.numpy()
        new_condition = whole_grad ** 2  # Here the direction is the
whole_grad
        new_condition = new_condition.sum()
        new_condition = alpha * step * new_condition

        # finding best step size using backtracking line search
        while loss_LSdata > loss_oridata - new_condition.cpu().numpy():
            step *= beta
            MaskClone = mask.data.clone()
            MaskClone -= step * whole_grad
            MaskClone = Variable(MaskClone, requires_grad=False)
            MaskClone.data.clamp_(0, 1)
            mask_LS = upsample(MaskClone)
            Img_LS = img.mul(mask_LS) + blurred_img.mul(1 - mask_LS)
            outputsLS = torch.nn.Softmax(dim=1)(model(Img_LS))
            loss_LS = l1_coeff * torch.mean(torch.abs(1 - MaskClone)) +
tv_coeff * tv_norm(MaskClone, tv_beta) + outputsLS[0, category]
            if use_cuda:
                loss_LSdata = loss_LS.data.cpu().numpy()
            else:
                loss_LSdata = loss_LS.data.numpy()
            new_condition = whole_grad ** 2  # Here the direction is the
whole_grad
            new_condition = new_condition.sum()
            new_condition = alpha * step * new_condition

            if step<0.00001:
                break
```

```
        mask.data -= step * whole_grad # integrated gradient descent step - we
have the updated mask at this point

        if use_cuda:
            curve1 = np.append(curve1, loss1.data.cpu().numpy())
            curve2 = np.append(curve2, loss2_ori.data.cpu().numpy()) # only
masking loss
            curve_total = np.append(curve_total, loss_ori.data.cpu().numpy())
        else:
            curve1 = np.append(curve1, loss1.data.numpy())
            curve2 = np.append(curve2, loss2_ori.data.numpy())
            curve_total = np.append(curve_total, loss_ori.data.numpy())
        mask.data.clamp_(0, 1)
        if use_cuda:
            maskdata = mask.data.cpu().numpy()
        else:
            maskdata = mask.data.numpy()
        maskdata = np.squeeze(maskdata)
        maskdata, imgratio = topmaxPixel(maskdata, 40)
        maskdata = np.expand_dims(maskdata, axis=0)
        maskdata = np.expand_dims(maskdata, axis=0)
        if use_cuda:
            Masktop = torch.from_numpy(maskdata).cuda()
        else:
            Masktop = torch.from_numpy(maskdata)

        # Use the mask to perturb the input image.
        Masktop = Variable(Masktop, requires_grad=False)
        MasktopLS = upsample(Masktop)
        Img_topLS = img.mul(MasktopLS) + blurred_img.mul(1 - MasktopLS)
        outputstopLS = torch.nn.Softmax(dim=1)(model(Img_topLS))
        loss_top1 = l1_coeff * torch.mean(torch.abs(1 - Masktop)) + tv_coeff *
tv_norm(Masktop, tv_beta)
        loss_top2 = outputstopLS[0, category]
        if use_cuda:
            curvetop = np.append(curvetop, loss_top2.data.cpu().numpy())
        else:
            curvetop = np.append(curvetop, loss_top2.data.numpy())
        if max_iterations > 3:

            if i == int(max_iterations / 2):
                if np.abs(curve2[0] - curve2[i]) <= 0.001:
                    l1_coeff = l1_coeff / 10

            elif i == int(max_iterations / 1.25):
                if np.abs(curve2[0] - curve2[i]) <= 0.01:
                    l1_coeff = l1_coeff / 5
             upsampled_mask = upsample(mask)
```

```
    if use_cuda:
        mask = mask.data.cpu().numpy().copy()
    else:
        mask = mask.data.numpy().copy()

    return mask, upsampled_mask


def Deletion_Insertion_Comb_withOverlay(max_patches, mask, model, output_path,
img_ori, blurred_img_ori, category, use_cuda=1, blur_mask=0, outputfig = 1):
    ########################
    # Compute the deletion and insertion scores
    #
    # parameters:
    # max_patches: number of literals in a root conjunction
    # mask: the generated mask
    # model: the model that you want to visualize
    # output_path: where to save the results
    # img_ori: the original image
    # blurred_img_ori: the baseline image
    # category: the classification target that you want to visualize
(category=-1 means the top 1 classification label)
    # use_cuda: use gpu (1) or not (0)
    # blur_mask: blur the mask or not
    # outputfig: save figure or not
    ######################################################

    if blur_mask: # invert mask, blur and re-invert
        mask = (mask - np.min(mask)) / np.max(mask)
        mask = 1 - mask
        mask = cv2.GaussianBlur(mask, (51, 51), 50)
        mask = 1-mask

    blurred_insert = blurred_img_ori.copy()
    blurred_insert = preprocess_image(blurred_insert, use_cuda,
require_grad=False)
    img = preprocess_image(img_ori, use_cuda, require_grad=False)
    blurred_img = preprocess_image(blurred_img_ori, use_cuda,
require_grad=False)
    resize_wh = (img.data.shape[2], img.data.shape[3])
    if use_cuda:
        upsample = torch.nn.UpsamplingBilinear2d(size=resize_wh).cuda()
    else:
        upsample = torch.nn.UpsamplingBilinear2d(size=resize_wh)
    # containers to store curve metrics
    del_curve = np.array([])
    insert_curve = np.array([])
    xtick = np.arange(0, max_patches, 1)
    xnum = xtick.shape[0]
```

```
    xtick = xtick.shape[0]+ 10
    # get the ground truth label for the given category
    f_groundtruth = open('./GroundTruth1000.txt')
    line_i = f_groundtruth.readlines()[category]
    f_groundtruth.close()

    # initialize insertion and deletion masks
    insertion_maskdata = np.zeros(mask.shape)
    deletion_maskdata = np.ones(mask.shape)

    showimg_buffer = [] # buffer to store figures - we save them only if
target_insertion_prob is achieved

    maskdata = mask.copy()
    maskdata = maskdata.astype(np.float32)
    if use_cuda:
        Masktop = torch.from_numpy(maskdata).cuda()
    else:
        Masktop = torch.from_numpy(maskdata)


    # Use the mask to perturb the input image - deletion mask.
    Masktop = Variable(Masktop, requires_grad=False)
    MasktopLS = upsample(Masktop)
    Img_topLS = img.mul(MasktopLS) + blurred_img.mul(1 - MasktopLS) #
perturbed image

    outputstopLS = torch.nn.Softmax(dim=1)(model(Img_topLS)) # all
probabilities
    deletion_loss = outputstopLS[0, category].data.cpu().numpy().copy() #
probability of class under consideration
    del_mask = MasktopLS.clone()
    del_curve = np.append(del_curve, deletion_loss)

    # insertion mask
    maskdata = mask.copy()
    maskdata = np.subtract(1, maskdata).astype(np.float32)
    if use_cuda:
        Masktop = torch.from_numpy(maskdata).cuda()
    else:
        Masktop = torch.from_numpy(maskdata)
    Masktop = Variable(Masktop, requires_grad=False)
    MasktopLS = upsample(Masktop)
    Img_topLS = img.mul(MasktopLS) + \
                blurred_insert.mul(1 - MasktopLS)
    outputstopLS = torch.nn.Softmax(dim=1)(model(Img_topLS))
    insertion_loss = outputstopLS[0, category].data.cpu().numpy().copy()
    ins_mask = MasktopLS.clone()
    insert_curve = np.append(insert_curve, insertion_loss)
```

```
    # store result images
    if outputfig == 1:
        deletion_img = save_new(del_mask, img_ori * 255, blurred_img_ori)
        insertion_img = save_new(ins_mask, img_ori * 255, blurred_img_ori)
        showimg_buffer.append((deletion_img, insertion_img, del_curve,
insert_curve, output_path, xtick, line_i))

    # round decimals
    deletion_loss = np.around(deletion_loss, decimals=3)
    insertion_loss = np.around(insertion_loss, decimals=3)

    return deletion_loss, insertion_loss, del_mask, showimg_buffer
def Deletion_Insertion_Comb_Successive(mask, model, output_path, img_ori,
blurred_img_ori, category, use_cuda=1, blur_mask=0, outputfig = 1):

    if blur_mask: # invert mask, blur and re-invert
        mask = (mask - np.min(mask)) / np.max(mask)
        mask = 1 - mask
        mask = cv2.GaussianBlur(mask, (51, 51), 50)
        mask = 1-mask

    blurred_insert = blurred_img_ori.copy()
    blurred_insert = preprocess_image(blurred_insert, use_cuda,
require_grad=False)
    img = preprocess_image(img_ori, use_cuda, require_grad=False)
    blurred_img = preprocess_image(blurred_img_ori, use_cuda,
require_grad=False)
    resize_wh = (img.data.shape[2], img.data.shape[3])
    if use_cuda:
        upsample = torch.nn.UpsamplingBilinear2d(size=resize_wh).cuda()
    else:
        upsample = torch.nn.UpsamplingBilinear2d(size=resize_wh)

    # containers to store curve metrics
    del_curve = np.array([])
    insert_curve = np.array([])
    max_patches = mask.shape[2] * mask.shape[3]
    xtick = np.arange(0, max_patches, 1)
    xnum = xtick.shape[0]
    xtick = xtick.shape[0]+ 10

    # get the ground truth label for the given category
    f_groundtruth = open('./GroundTruth1000.txt')
    line_i = f_groundtruth.readlines()[category]
    f_groundtruth.close()
    # initialize insertion and deletion masks
    insertion_maskdata = np.zeros(mask.shape)
    deletion_maskdata = np.ones(mask.shape)
```

```
    showimg_buffer = [] # buffer to store figures - we save them only if
target_insertion_prob is achieved
    maskdata = np.ones(mask.shape)
    while not np.all(mask == 1):
        maskdata, mask, imgratio = add_topMaskPixel(maskdata, mask)
        # maskdata = mask.copy()
        maskdata_del = maskdata.astype(np.float32)
        if use_cuda:
            Masktop = torch.from_numpy(maskdata_del).cuda()
        else:
            Masktop = torch.from_numpy(maskdata_del)

        # Use the mask to perturb the input image - deletion mask.
        Masktop = Variable(Masktop, requires_grad=False)
        MasktopLS = upsample(Masktop)
        Img_topLS = img.mul(MasktopLS) + blurred_img.mul(1 - MasktopLS) #
perturbed image

        outputstopLS = torch.nn.Softmax(dim=1)(model(Img_topLS)) # all
probabilities
        deletion_loss = outputstopLS[0, category].data.cpu().numpy().copy() #
probability of class under consideration
        del_mask = MasktopLS.clone()
        del_curve = np.append(del_curve, deletion_loss)

        # insertion mask
        maskdata_ins = 1 - maskdata
        maskdata_ins = maskdata_ins.astype(np.float32)
        if use_cuda:
            Masktop = torch.from_numpy(maskdata_ins).cuda()
        else:
            Masktop = torch.from_numpy(maskdata_ins)
        Masktop = Variable(Masktop, requires_grad=False)
        MasktopLS = upsample(Masktop)
        Img_topLS = img.mul(MasktopLS) + \
                    blurred_insert.mul(1 - MasktopLS)
        outputstopLS = torch.nn.Softmax(dim=1)(model(Img_topLS))
        insertion_loss = outputstopLS[0, category].data.cpu().numpy().copy()
        ins_mask = MasktopLS.clone()
        insert_curve = np.append(insert_curve, insertion_loss)
        # store result images
        if outputfig == 1:
            deletion_img = save_new(del_mask, img_ori * 255, blurred_img_ori)
            insertion_img = save_new(ins_mask, img_ori * 255, blurred_img_ori)
            showimg_buffer.append((deletion_img, insertion_img, del_curve,
insert_curve, output_path, xtick, line_i))
```

```
            # flush mask if insertion prob > 0.9
            if insertion_loss > 0.9:
                maskdata = np.ones(mask.shape)

        # round decimals
        deletion_loss = np.around(deletion_loss, decimals=3)
        insertion_loss = np.around(insertion_loss, decimals=3)

        return deletion_loss, insertion_loss, del_mask, showimg_buffer


def Deletion_Insertion_Comb_Successive_Corrected(mask, model, output_path,
img_ori, blurred_img_ori, category, full_prob, prob_thresh, showimg_buffer,
use_cuda=1, blur_mask=0, outputfig = 1):
    success = False

    if blur_mask: # invert mask, blur and re-invert
        mask = (mask - np.min(mask)) / np.max(mask)
        mask = 1 - mask
        mask = cv2.GaussianBlur(mask, (51, 51), 50)
        mask = 1-mask

    blurred_insert = blurred_img_ori.copy()
    blurred_insert = preprocess_image(blurred_insert, use_cuda,
require_grad=False)
    img = preprocess_image(img_ori, use_cuda, require_grad=False)
    blurred_img = preprocess_image(blurred_img_ori, use_cuda,
require_grad=False)
    resize_wh = (img.data.shape[2], img.data.shape[3])
    if use_cuda:
        upsample = torch.nn.UpsamplingBilinear2d(size=resize_wh).cuda()
    else:
        upsample = torch.nn.UpsamplingBilinear2d(size=resize_wh)

    # containers to store curve metrics
    if showimg_buffer == []:
        del_curve = np.array([])
        insert_curve = np.array([])
    else:
        del_curve = showimg_buffer[-1][2]
        insert_curve = showimg_buffer[-1][3]
    max_patches = mask.shape[2] * mask.shape[3]
    xtick = np.arange(0, max_patches, 1)
    xnum = xtick.shape[0]
    xtick = xtick.shape[0]+ 10

    # get the ground truth label for the given category
    f_groundtruth = open('./GroundTruth1000.txt')
    line_i = f_groundtruth.readlines()[category]
```

```
    f_groundtruth.close()
    # initialize insertion and deletion masks
    insertion_maskdata = np.zeros(mask.shape)
    deletion_maskdata = np.ones(mask.shape)

    # initialize to avoid reference before assignment error
    deletion_img = deepcopy(blurred_img_ori)
    del_mask = deepcopy(mask)
    del_mask = del_mask.astype(np.float32)
    if use_cuda:
        del_mask = torch.from_numpy(del_mask).cuda()
    else:
        del_mask = torch.from_numpy(del_mask)
    del_mask = Variable(del_mask, requires_grad=False)
    del_mask = upsample(del_mask)

    # showimg_buffer = [] # buffer to store figures - we save them only if
target_insertion_prob is achieved
    maskdata = np.ones(mask.shape)
    while not np.all(mask == 1):
        maskdata, mask, imgratio = add_topMaskPixel(maskdata, mask)
        # maskdata = mask.copy()
        maskdata_del = maskdata.astype(np.float32)
        if use_cuda:
            Masktop = torch.from_numpy(maskdata_del).cuda()
        else:
            Masktop = torch.from_numpy(maskdata_del)

        # Use the mask to perturb the input image - deletion mask.
        Masktop = Variable(Masktop, requires_grad=False)
        MasktopLS = upsample(Masktop)
        Img_topLS = img.mul(MasktopLS) + blurred_img.mul(1 - MasktopLS) #
perturbed image

        outputstopLS = torch.nn.Softmax(dim=1)(model(Img_topLS)) # all
probabilities
        deletion_loss = outputstopLS[0, category].data.cpu().numpy().copy() #
probability of class under consideration
        deletion_loss = deletion_loss / full_prob
        del_mask = MasktopLS.clone()
        del_curve = np.append(del_curve, deletion_loss)

        # insertion mask
        maskdata_ins = 1 - maskdata
        maskdata_ins = maskdata_ins.astype(np.float32)
        if use_cuda:
            Masktop = torch.from_numpy(maskdata_ins).cuda()
```

```
            else:
                Masktop = torch.from_numpy(maskdata_ins)
            Masktop = Variable(Masktop, requires_grad=False)
            MasktopLS = upsample(Masktop)
            Img_topLS = img.mul(MasktopLS) + \
                        blurred_insert.mul(1 - MasktopLS)
            outputstopLS = torch.nn.Softmax(dim=1)(model(Img_topLS))
            insertion_loss = outputstopLS[0, category].data.cpu().numpy().copy()
            insertion_loss = insertion_loss / full_prob
            ins_mask = MasktopLS.clone()
            insert_curve = np.append(insert_curve, insertion_loss)

            # store result images
            if outputfig == 1:
                deletion_img = save_new(del_mask, img_ori * 255, blurred_img_ori)
                insertion_img = save_new(ins_mask, img_ori * 255, blurred_img_ori)
                showimg_buffer.append((deletion_img, insertion_img, del_curve,
insert_curve, output_path, xtick, line_i))

            # break if insertion prob > 0.9
            if insertion_loss > prob_thresh:
                success = True
                break

    # round decimals
    # deletion_loss = np.around(deletion_loss, decimals=3)
    # insertion_loss = np.around(insertion_loss, decimals=3)

    return success, deletion_img, del_mask, showimg_buffer

def save_new(mask, img, blurred):
    ########################
    # generate the perturbed image for saving as result
    #
    # parameters:
    # mask: the generated mask
    # img: the original image
    # blurred: the baseline image
    ######################################################
    mask = mask.cpu().data.numpy()[0]
    mask = np.transpose(mask, (1, 2, 0))
    img = np.float32(img) / 255
    perturbated = np.multiply(mask, img) + np.multiply(1-mask, blurred)
    perturbated = cv2.cvtColor(perturbated, cv2.COLOR_BGR2RGB)
    return perturbated
```

```
def showimage(del_img, insert_img, del_curve, insert_curve, target_path,
xtick, title):
    ########################
    # generate the result frame used for videos
    #
    # parameters:
    # del_img: the deletion image
    # insert_img: the insertion image
    # del_curve: the deletion curve
    # insert_curve: the insertion curve
    # target_path: where to save the results
    # xtick: xtick
    # title: title
    ########################################################
    pylab.rcParams['figure.figsize'] = (10, 10)
    f, ax = plt.subplots(2,2)
    f.suptitle('Category ' + title, y=0.04, fontsize=13)
    f.tight_layout()
    plt.subplots_adjust(left=0.005, bottom=0.1, right=0.98, top=0.93,
                        wspace=0.05, hspace=0.25)
    ax[0,0].imshow(del_img)
    ax[0,0].set_xticks([])
    ax[0,0].set_yticks([])
    ax[0,0].set_title("Deletion", fontsize=13)
    ax[1,0].imshow(insert_img)
    ax[1,0].set_xticks([])
    ax[1,0].set_yticks([])
    ax[1,0].set_title("Insertion", fontsize=13)
    ax[0,1].plot(del_curve,'r*-')
    ax[0,1].set_xlabel('number of blocks')
    ax[0,1].set_ylabel('classification confidence')
    ax[0,1].legend(['Deletion'])
    ax[0,1].set_xticks(range(0, xtick, 10))
    ax[0, 1].set_yticks(np.arange(0, 1.1, 0.1))
    ax[1,1].plot(insert_curve, 'b*-')
    ax[1, 1].set_xlabel('number of blocks')
    ax[1,1].set_ylabel('classification confidence')
    ax[1,1].legend(['Insertion'])
    ax[1, 1].set_xticks(range(0, xtick, 10))
    ax[1, 1].set_yticks(np.arange(0, 1.1, 0.1))

    plt.savefig(target_path + 'video'+ str(insert_curve.shape[0])+ '.jpg')
    plt.close()
```

B.6 Functions for Diverse Subset Selection of the Set of Candidate Masks (diverse_subset_selection.py)

```
import  numpy as np
import itertools
import random
import math

from utils import *

import os
import time
import scipy.io as scio
import datetime
import re
import matplotlib.pyplot as plt
import pylab
import os
import csv
from skimage import transform, filters
from textwrap import wrap
import cv2
import sys
from PIL import Image



def create_patch_weight_matrix(all_mp, coeff):
    pwm = np.zeros_like(all_mp[0][0])
    for k in range(len(all_mp)):
        mask, p = all_mp[k]
        pwm += (1-mask) * math.exp(coeff * p)
    return pwm

def union_bmasks(m1,m2):
    return m1*m2

def intersection_bmasks(m1,m2):
    m = m1+m2
    m = np.clip(m, 0, 1)
    return m

def intersection_score(m1,m2):
    intrmask = intersection_bmasks(m1,m2)
    return np.sum(1-intrmask)
```

```
def coverage(pwm, mask):
    pwm_intersection = pwm * (1-mask)
    return np.sum(pwm_intersection)

def max_coverage_set(pwm, all_mp):
    all_mc = []
    for mask, p in all_mp:
        cov = coverage(pwm, mask)
        all_mc.append((mask, p, cov))
    all_mc_sorted = list(sorted(all_mc, key=lambda x: x[2], reverse=True))
    #print('all_mc_sorted:\n', all_mc_sorted)
    maxcov = all_mc_sorted[0][2]
    mask_set = []
    for mask, p, cov in all_mc_sorted:
        if cov == maxcov:
            mask_set.append((mask,p))
        else:
            break
    return mask_set


def min_intersection_set(all_mp):
    mask_set = []
    l = len(all_mp)
    intr_mat = np.zeros((l,l))
    for i in range(l):
        for j in range(i+1,l):
            intr_mat[i][j] = intersection_score(all_mp[i][0], all_mp[j][0])
            intr_mat[j][i] = intr_mat[i][j]
    intr_array = np.sum(intr_mat, axis=0)
    minintr = np.min(intr_array)
    min_indices = np.where(intr_array == minintr)[0]
    size = np.size(min_indices)
    for k in range(size):
        mask_set.append(all_mp[min_indices[k]])
    return mask_set


def create_intersection_matrix(all_mp):
    l = len(all_mp)
    intr_mat = np.zeros((l,l))
    for i in range(l):
        for j in range(i+1,l):
            intr_mat[i][j] = intersection_score(all_mp[i][0], all_mp[j][0])
            intr_mat[j][i] = intr_mat[i][j]
        intr_mat[i][i] = intersection_score(all_mp[i][0], all_mp[i][0])
    return intr_mat
```

```
def tie_breaker_prob(indices, all_mp):
    maxprob = -1
    ans = -1
    for i in indices:
        if all_mp[i][1] > maxprob:
            maxprob = all_mp[i][1]
            ans = i
    return [ans]


def disparity_min_indices(all_mp, im, refset):
    minintersection = float('inf')
    minintersection_indices = []

    if len(refset) == 0:
        minintersection = im.min()
        for i in range(len(all_mp)):
            if im[i].min() == minintersection:
                minintersection_indices.append(i)
    else:
        maxintersection_pairs = []
        for i in range(len(all_mp)):
            if i not in refset:
                maxintersection = 0
                for j in refset:
                    if maxintersection < im[i][j]:
                        maxintersection = im[i][j]
                maxintersection_pairs.append((i,maxintersection))
                if minintersection > maxintersection:
                    minintersection = maxintersection
        for index, maxintersection in maxintersection_pairs:
            if maxintersection == minintersection:
                minintersection_indices.append(index)
    return minintersection_indices

def get_best_mask_index(all_mp, im, refset):
    filtered_mp_indices = disparity_min_indices(all_mp, im, refset)
    if len(filtered_mp_indices) > 1:
        filtered_mp_indices = tie_breaker_prob(filtered_mp_indices, all_mp)
    return filtered_mp_indices


def update_pwm(best_mp, pwm):
    best_mask, p = best_mp
    cov = coverage(pwm, best_mask)
    pwm = pwm * best_mask
    return pwm, cov
```

```
def set_filtering(all_mp):
    diverse_mp_indices = []
    disparity_min_set = []
    num_candidates = len(all_mp)
    im = create_intersection_matrix(all_mp)
    while len(diverse_mp_indices) < num_candidates:
        best_mp_index = get_best_mask_index(all_mp, im, diverse_mp_indices)
        assert len(best_mp_index) == 1
        best_mp_index = best_mp_index[0]
        diverse_mp_indices.append(best_mp_index)
        best_mp = all_mp[best_mp_index]
        disparity_min_set.append((best_mp[0], best_mp[1]))
    return disparity_min_set


def overlapThresh_dfs(n, im, refset, overlap_thresh):
    noOverlap_indices = []
    maximal_refset = []
    maximal_refset_length = 0
    index = refset[-1]
    for j in range(index+1,n): # consider only upper triangular elements as
symmetric matrix
            # check for no overlap with refset
            valid = True
            for k in refset:
                if (im[j][k] > overlap_thresh):
                    valid = False
                    break
            if valid:
                noOverlap_indices.append(j)
    # base case
    if noOverlap_indices == []:
        return refset


    # trigger dfs for new indices

    for j in noOverlap_indices:
        candidate_refset = overlapThresh_dfs(n, im, refset + [j], \
                                                overlap_thresh)
        # save maximal refset
        candidate_refset_length = len(candidate_refset)
        if maximal_refset_length < candidate_refset_length:
            maximal_refset_length = candidate_refset_length
            maximal_refset = candidate_refset
    return maximal_refset
```

```
# function to return maximal set of masks, all of which have zero overlap
def maximal_overlapThresh_set(all_mp, overlap_thresh):
    ans_mp = []
    max_set = []
    max_set_length = 0
    n = len(all_mp)
    im = create_intersection_matrix(all_mp)
    # trigger dfs for given overlap threshold
    for i in range(n):
        refset = [i]
        candidate_set = overlapThresh_dfs(n, im, refset, overlap_thresh)
        candidate_set_length = len(candidate_set)
        if max_set_length < candidate_set_length:
            max_set_length = candidate_set_length
            max_set = candidate_set
    for index in max_set:
        ans_mp.append(all_mp[index])
    return ans_mp
```

B.7 Custom Functions (custom_functions.py)

```
import os
import torch
import shutil
import torch.nn as nn
from torchvision import models
from torchvision import datasets
import torch.nn.functional as F

from custom_model_architectures import custom_model_architecture

def load_model(model_path,model_name):

    model = None
    dataset_path = r'/media/kv/Documents/git/mtkvcs-dataset/datscan/'

    print(f"Model name: {model_name}\n")
    train_data = datasets.ImageFolder(os.path.join(dataset_path,'train'))
    class_names = train_data.classes
    print(f"Class names {class_names}\n")

    if model_name == 'resnet50':
        model = models.resnet50(pretrained=True)

        model.fc = nn.Linear(2048,2, bias=True)

        model.load_state_dict(torch.load(model_path))

    elif model_name == 'vgg19':
        model = models.vgg19(pretrained=True)

        model.classifier = nn.Sequential(nn.Linear(25088,4096), # Configure
the classifier
                            nn.ReLU(inplace=True),
                            nn.Dropout(0.5),
                            nn.Linear(4096,4096, bias=True),
                            nn.ReLU(inplace=True),
                            nn.Dropout(0.5),
                            nn.Linear(4096,len(class_names), bias=True))
    elif model_name == 'custom':
        model = custom_model_architecture()
        pass
    model.load_state_dict(torch.load(model_path))
    return model
```

```
def purge(folder):
    for filename in os.listdir(folder):
        file_path = os.path.join(folder, filename)
        try:
            if os.path.isfile(file_path) or os.path.islink(file_path):
                os.unlink(file_path)
            elif os.path.isdir(file_path):
                shutil.rmtree(file_path)
        except Exception as e:
            print('Failed to delete %s. Reason: %s' % (file_path, e))
```

B.8 Custom Model Architectures (custom_model_architectures.py)

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class custom_model_architecture(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 3, 1)
        self.conv2 = nn.Conv2d(6, 16, 3, 1)
        self.fc1 = nn.Linear(54*54*16, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 2)

    def forward(self, X):
        X = F.relu(self.conv1(X))
        X = F.max_pool2d(X, 2, 2)
        X = F.relu(self.conv2(X))
        X = F.max_pool2d(X, 2, 2)
        X = X.view(-1, 54*54*16)
        X = F.relu(self.fc1(X))
        X = F.relu(self.fc2(X))
        X = self.fc3(X)
        return F.log_softmax(X, dim=1)
```

B.9 Function for Running Multiple Models (loop.py)

```
import os
import time
from main_generate_sag import main

folder = r'vgg19_20220910_201515'
subfolder = 'vgg19_20220910_201515_epoch_8'

ls = [8,11,12]
model_name = folder.split("_")[0]

root = r'/media/kv/Documents/git/mtkvcs-saved-models/'
dataset_path = r'/media/kv/Documents/git/mtkvcs-custom-cnn/dataset/datscan/'
unique_id = time.strftime("%Y%m%d_%H%M%S")
results_path = root + r'/' + folder + r'/Results/'+ unique_id

loop=0
if loop==0:
    for j in os.listdir(root + folder + r'/' + subfolder):
            if ".pt" in j:
                print("Run only model: " + j)
                main(root + folder + r'/' + subfolder + r'/' + j, folder,
results_path)
elif loop==1:
    for i in os.listdir(root+folder):
        if "epoch" in i:
            for j in os.listdir(root+folder + r'/' +i):
                if ".pt" in j:
                    main(root + folder + r'/' + i + r'/' + j, folder,
results_path)
elif loop == -1:
    print("Old format")
    main(root + folder + r'/' + folder + r'.pt', folder, results_path)
```

B.10 Notes for the Code Implementation

It should be noted that the code is customized in order to fit to the needs of the custom implementation.

Appendix C
Class Activation Mapping Code

The code is available here: https://github.com/karvo/ouc-cogsys-pytorch-grad-cam

C.1 Setup (setup.py)

```
import setuptools

with open('README.md', mode='r', encoding='utf-8') as fh:
    long_description = fh.read()

with open("requirements.txt", "r") as f:
    requirements = f.readlines()

setuptools.setup(
    name='grad-cam',
    version='1.3.7',
    author='Jacob Gildenblat',
    author_email='jacob.gildenblat@gmail.com',
    description='Many Class Activation Map methods implemented in Pytorch. '
                'Including Grad-CAM, Grad-CAM++, Score-CAM, Ablation-CAM and
XGrad-CAM',
    long_description=long_description,
    long_description_content_type='text/markdown',
    url='https://github.com/jacobgil/pytorch-grad-cam',
    project_urls={
        'Bug Tracker': 'https://github.com/jacobgil/pytorch-grad-cam/issues',
    },
    classifiers=[
        'Programming Language :: Python :: 3',
        'License :: OSI Approved :: MIT License',
        'Operating System :: OS Independent',
    ],
    packages=setuptools.find_packages(),
    python_requires='>=3.6',
    install_requires=requirements
)
```

C.2 Class Activation Mapping (cam.py)

```
import argparse
import cv2
import numpy as np
import torch
from torchvision import models
from pytorch_grad_cam import GradCAM, \
    ScoreCAM, \
    GradCAMPlusPlus, \
    AblationCAM, \
    XGradCAM, \
    EigenCAM, \
    EigenGradCAM, \
    LayerCAM, \
    FullGrad
from pytorch_grad_cam import GuidedBackpropReLUModel
from pytorch_grad_cam.utils.image import show_cam_on_image, \
    deprocess_image, \
    preprocess_image
from pytorch_grad_cam.utils.model_targets import ClassifierOutputTarget
import os

def get_args(img_path): #<---------------
    parser = argparse.ArgumentParser()
    parser.add_argument('--use-cuda', action='store_true', default=True,
                        help='Use NVIDIA GPU acceleration')
    parser.add_argument(
        '--image-path',
        type=str,
        default=img_path, #<-------------------------------------------------
----------------------
        help='Input image path')
    parser.add_argument('--aug_smooth', action='store_true',
                        help='Apply test time augmentation to smooth the CAM')
    parser.add_argument(
        '--eigen_smooth',
        action='store_true',
        help='Reduce noise by taking the first principle componenet'
        'of cam_weights*activations')
    parser.add_argument('--method', type=str, default='gradcam',
                        choices=['gradcam', 'gradcam++',
                                 'scorecam', 'xgradcam',
                                 'ablationcam', 'eigencam',
                                 'eigengradcam', 'layercam', 'fullgrad'],
                        help='Can be gradcam/gradcam++/scorecam/xgradcam'
                             '/ablationcam/eigencam/eigengradcam/layercam')
```

```
    args = parser.parse_args()
    args.use_cuda = args.use_cuda and torch.cuda.is_available()
    if args.use_cuda:
        print('Using GPU for acceleration')
    else:
        print('Using CPU for computation')

    return args


def main(model, img_path, output_path, imgname):
    """ python cam.py -image-path <path_to_image>
    Example usage of loading an image, and computing:
        1. CAM
        2. Guided Back Propagation
        3. Combining both
    """

    args = get_args(img_path) #<-----------------------------------------
--------
    methods = \
        {"gradcam": GradCAM,
         "scorecam": ScoreCAM,
         "gradcam++": GradCAMPlusPlus,
         "ablationcam": AblationCAM,
         "xgradcam": XGradCAM,
         "eigencam": EigenCAM,
         "eigengradcam": EigenGradCAM,
         "layercam": LayerCAM,
         "fullgrad": FullGrad}

    # Choose the target layer you want to compute the visualization for.
    # Usually this will be the last convolutional layer in the model.
    # Some common choices can be:
    # Resnet18 and 50: model.layer4
    # VGG, densenet161: model.features[-1] <---------------------------------
-------------------------------------------------
    # mnasnet1_0: model.layers[-1]
    # You can print the model to help chose the layer
    # You can pass a list with several target layers,
    # in that case the CAMs will be computed per layer and then aggregated.
    # You can also try selecting all layers of a certain type, with e.g:
    # from pytorch_grad_cam.utils.find_layers import
find_layer_types_recursive
    # find_layer_types_recursive(model, [torch.nn.ReLU])
    target_layers = [model.features[-1]]

    rgb_img = cv2.imread(args.image_path, 1)[:, :, ::-1]
    rgb_img = np.float32(rgb_img) / 255
```

```
    input_tensor = preprocess_image(rgb_img,
                                    mean=[0.485, 0.456, 0.406],
                                    std=[0.229, 0.224, 0.225])
    # We have to specify the target we want to generate
    # the Class Activation Maps for.
    # If targets is None, the highest scoring category (for every member in
the batch) will be used.
    # You can target specific categories by
    # targets = [e.g ClassifierOutputTarget(281)]
    targets = None

    # Using the with statement ensures the context is freed, and you can
    # recreate different CAM objects in a loop.
    cam_algorithm = methods[args.method]
    with cam_algorithm(model=model,
                       target_layers=target_layers,
                       use_cuda=args.use_cuda) as cam:

        # AblationCAM and ScoreCAM have batched implementations.
        # You can override the internal batch size for faster computation.
        cam.batch_size = 32
        grayscale_cam = cam(input_tensor=input_tensor,
                            targets=targets,
                            aug_smooth=args.aug_smooth,
                            eigen_smooth=args.eigen_smooth)

        # Here grayscale_cam has only one image in the batch
        grayscale_cam = grayscale_cam[0, :]

        cam_image = show_cam_on_image(rgb_img, grayscale_cam, use_rgb=True)

        # cam_image is RGB encoded whereas "cv2.imwrite" requires BGR
encoding.
        cam_image = cv2.cvtColor(cam_image, cv2.COLOR_RGB2BGR)

    gb_model = GuidedBackpropReLUModel(model=model, use_cuda=args.use_cuda)
    gb = gb_model(input_tensor, target_category=None)
    cam_mask = cv2.merge([grayscale_cam, grayscale_cam, grayscale_cam])
    cam_gb = deprocess_image(cam_mask * gb)
    gb = deprocess_image(gb)

    os.chdir(output_path) #<----------------------------------------------
    cv2.imwrite(f'{imgname}_{args.method}_cam.jpg', cam_image)
    cv2.imwrite(f'{imgname}_{args.method}_gb.jpg', gb)
    cv2.imwrite(f'{imgname}_{args.method}_cam_gb.jpg', cam_gb)
    os.chdir(os.path.abspath(os.path.join(os.getcwd(), os.pardir)))
    os.chdir(os.path.abspath(os.path.join(os.getcwd(), os.pardir)))
    #os.chdir("../nodes") #<----------------------------------------------
```

C.3 Custom Main Function (custom_main_cam.py)

```
from cam import main
import os, time, shutil
import torch
import shutil
import torch.nn as nn
from torchvision import models, transforms
from custom_functions import load_model



model, model_filename, dst_path = load_model()
model.eval()

input_folder = 'custom_datapoints'
input_path = './' + input_folder + '/'
dirs = os.listdir(input_path)
print(dirs)

for d in dirs:
    print("Current directory",os.getcwd())
    print("path: ", input_path+d)
    if not os.path.isdir(input_path + d):
        continue
    files = os.listdir(input_path + d)
    total_files = len(files)
    file_counter = 0
    output_path = './Results/' + d + '/'
    if not os.path.isdir(output_path):
        os.makedirs(output_path)
    pre_existing_result_dirs = os.listdir(output_path)
    for imgname in files:
        if imgname.endswith('JPEG') or imgname.endswith('jpg') or
imgname.endswith('png'):
            if not os.path.exists(output_path +imgname):
                shutil.copy(input_path+d+'/'+imgname, output_path)
            input_img = input_path + d + '/' + imgname
            print('imgname:', imgname)
            imgprefix = imgname.split('.')[0]
            print(imgprefix)
            # check if this file is already processes (results exist)
            if imgname in pre_existing_result_dirs:
                print('skipping')
                continue
            main(model, input_img, output_path, imgname)
```

```
src_path = "./Results/"
unique_id = time.strftime("%Y%m%d_%H%M%S")
dst_path = dst_path + model_filename + "/Results/" + unique_id  + "grad_cam" +
"/"
shutil.copytree(src_path, dst_path)
s
```

C.4 Custom Functions (custom_functions.py)

```
import torch
import torch.nn as nn
from torchvision import models

def load_model():

    folder = r'vgg19_20220918_211657'
    sub = r'vgg19_20220918_211657_fold_1'
    model_name = folder.split('_')[0]
    if model_name == 'resnet50':
        model = models.resnet50(pretrained=True)
    elif model_name == 'vgg19':
        model = models.vgg19(pretrained=True)
        root = r'/media/kv/Documents/git/mtkvcs-saved-models/'

        path = root + folder + r'/' + sub + r'/'
        filename = sub + r'.pt'
        path = path+filename

        model.classifier = nn.Sequential(nn.Linear(25088,3136), # Configure
the classifier
                                nn.ReLU(inplace=True),
                                nn.Dropout(0.5),
                                nn.Linear(3136,3136, bias=True),
                                nn.ReLU(inplace=True),
                                nn.Dropout(0.5),
                                nn.Linear(3136,2, bias=True))

        model.load_state_dict(torch.load(path))

    return model, folder, root
```

Appendix D
Augmented Datapoints Generator Code

The code is available here:
https://github.com/karvo/ouc-cogsys-custom-cnn

D.1 Augmented Datapoints Generator

```
import torch
import numpy as np
import torch.nn as nn
import torch.optim as optim
from tqdm import tqdm
import matplotlib.pyplot as plt
import os, time, copy, itertools
from torch.optim import lr_scheduler
from sklearn.metrics import f1_score
from torchvision.utils import save_image
from sklearn.metrics import confusion_matrix
from matplotlib.backends.backend_pdf import PdfPages
from torchvision import models, datasets, transforms
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler,
ConcatDataset
print("GPU is", "available" if torch.cuda.is_available() else "NOT AVAILABLE")
data_dir = r'/media/kv/Documents/git/mtkvcs-custom-cnn/dataset/datscan/train/'
dst_dir = r'/media/kv/Documents/git/mtkvcs-custom-cnn/dataset/datscan/train2/'
img_num = 0


augmented_dataset = datasets.ImageFolder(data_dir)
classes = augmented_dataset.class_to_idx
print(classes)
class_keys = list(classes.keys())
```

```
for _ in tqdm(range(10)):

    augmented_dataset.transform= transforms.Compose([
    transforms.Resize(224),             # resize shortest side to 224 pixels
    transforms.CenterCrop(224),         # crop longest side to 224 pixels at
center
    transforms.ColorJitter(contrast=1, saturation=0.5, hue=0.5),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor()
    ])
    for img, label in augmented_dataset:
        if label == 0:
            save_image(img, dst_dir + class_keys[label] + '/aug_' +
class_keys[label]+str(img_num)+'.png')
        elif label == 1:
            save_image(img, dst_dir + class_keys[label] + '/aug_' +
class_keys[label]+str(img_num)+'.png')
        img_num+=1
```

Bibliographical References

A.G. Ivakhnenko, and V.G. Lapa, “Cybernetics and Forecasting Techniques, Volume 8 of Modern analytic and computational methods in science and mathematics”, R.N. McDonough (ed), American Elsevier Publishing Company, 1967. ISBN: 978-0-444- 00020-0.

Aggarwal C.C. (2018) Convolutional Neural Networks. In: Neural Networks and Deep Learning. Springer, Cham. https://doi.org/10.1007/978-3-319-94463-0_8

Agrawal, Sachin. (2021). Online) 2581-9429 ISSN (Print) 2581-XXXX. International Journal of Advanced Research in Science, Communication and Technology. 10.48175/IJARSCT-807.

Alahmari A., (2021). Neuroimaging Role in Mental Illnesses. 10.31579/ijnp.2021/011

Attention (Stanford Encyclopedia of Philosophy). (2017, September 1). Stanford Encyclopedia of Philosophy. https://plato.stanford.edu/entries/attention/

Badr, W. (2019, July 01). Auto-Encoder: What Is It? And What Is It Used For? (Part 1). Retrieved November 10, 2020, from https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726

Bharati, P., & Chaudhury, A. (2004). An empirical investigation of decision-making satisfaction in web-based decision support systems. Decision Support Systems, 37(2), 187–197. https://doi.org/10.1016/s0167-9236(03)00006-x

Bonnefon, Jean-François & Shariff, Azim & Rahwan, Iyad. (2016). The Social Dilemma of Autonomous Vehicles. Science. 352. 10.1126/science.aaf2654.

C. (2021, March 12). Why is Tesla using Pytorch for Autopilot? Corporates Today. Retrieved April 6, 2022, from https://corporates.today/why-tesla-using-pytorch-for-autopilot

C. Szegedy et al., "Going deeper with convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594

Chen, Z. (2012). Object-based attention: A tutorial review. Attention, Perception, & Psychophysics, 74(5), 784–802. https://doi.org/10.3758/s13414-012-0322-z

Coello, Y. (2005). Spatial context and visual perception for action. Psicologica, 26, 39-59. Encyclopedia of Animal Cognition and Behavior [Abstract]. (2020). doi:10.1007/978-3-319-47829-6

DeMaagd G, Philip A. Parkinson's Disease and Its Management: Part 1: Disease Entity, Risk Factors, Pathophysiology, Clinical Presentation, and Diagnosis. P T. 2015 Aug;40(8):504-32. PMID: 26236139; PMCID: PMC4517533.

Draelos, R. L., & Carin, L. (2021, November 21). Use HiResCAM instead of grad-cam for faithful explanations of Convolutional Neural Networks. arXiv.org. Retrieved April 9, 2022, from https://arxiv.org/abs/2011.08891v4

Hayes-Roth, F. (1985). Rule-based systems. Communications of the ACM, 28(9), 921–932. https://doi.org/10.1145/4284.4286

Hinton G. (2011) Boltzmann Machines. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_83

Hinton GE, Osindero S, Teh YW. A fast-learning algorithm for deep belief nets. Neural Comput. 2006 Jul;18(7):1527-54. doi: 10.1162/neco.2006.18.7.1527. PMID: 16764513.

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. “Distilling the knowledge in a neural network.” arXiv preprint arXiv:1503.02531 (2015).

I. Kollia, A. -G. Stafylopatis and S. Kollias, "Predicting Parkinson's Disease using Latent Information extracted from Deep Neural Networks," 2019 International Joint
Conference on Neural Networks (IJCNN), 2019, pp. 1-8, doi: 10.1109/IJCNN.2019.8851995

Israel, M. M., Jolicoeur, P., & Cohen, A. (2016). Spatial attention across perception and action. Psychological Research, 82(2), 255–271. https://doi.org/10.1007/s00426-016-0820-z

J. Wingate, I. Kollia, L. Bidaut and S. Kollias, "Unified deep learning approach for prediction of Parkinson's disease" https://doi.org/10.48550/arXiv.1911.10653

J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.

Jacobi, A., Chung, M., Bernheim, A., & Eber, C. (2020). Portable chest X-ray in coronavirus disease-19 (covid-19): A Pictorial Review. Clinical Imaging, 64, 35–42. https://doi.org/10.1016/j.clinimag.2020.04.001
John R Zech, Marcus A Badgeley, Manway Liu, Anthony B Costa, Joseph J Titano, and Eric Karl Oermann. Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: a cross-sectional study. PLoS medicine, 15(11):e1002683, 2018.

Jadhav, Vinit & Ligay, Vladislav. (2016). Forecasting Energy Consumption using Machine Learning.

Jordan, J. (2018, October 20). Common architectures in convolutional neural networks. Retrieved November 09, 2020, from https://www.jeremyjordan.me/convnet-architectures/

Jayaweera, Pasan. (2021). Design and Implementation of Electromyography (EMG) based Real-Time Pattern Recognition model for Prosthetic hand Control. 10.31219/osf.io/rd2cf.

Karri, Sri Phani & Chakraborty, Debjani & Chatterjee, Jyotirmoy. (2017). Transfer learning-based classification of optical coherence tomography images with diabetic macular edema and dry age-related macular degeneration. Biomedical Optics Express. 8. 579. 10.1364/BOE.8.000579.

Khan, Fazeel & Abubakar, Adamu & Zeki, Akram. (2020). Environmental monitoring and disease detection of plants in smart greenhouse using Internet of Things. Journal of Physics Communications. 4. 10.1088/2399-6528/ab90c1.

Kingma, D.P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. CoRR, abs/1412.6980.

Kone, C. (2019, November 6). Introducing convolutional neural networks in Deep Learning. Medium. Retrieved December 16, 2022, from https://towardsdatascience.com/introducing-convolutional-neural-networks-in-deep-learning-400f9c3ad5e9

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84-90. doi:10.1145/3065386

Langevin, J.-P. (2019, August 2). 4 factors for successful deep brain stimulation in movement disorders. Pacific Neuroscience Institute. Retrieved March 16, 2022, from https://www.pacificneuroscienceinstitute.org/blog/movement-disorders/4-factors-for-successful-deep-brain-stimulation-movement-disorders/

Lappin, J. S., Seiffert, A. E., & Bell, H. H. (2020). A Limiting Channel Capacity of Visual Perception: Spreading Attention Divides the Rates of Perceptual Processes. Attention, Perception, & Psychophysics, 82(5), 2652-2672. doi:10.3758/s13414-020-01973-9

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. doi:10.1109/5.726791

Marek, K., Seibyl, J., Hogan, E., Hesterman, J., Lowery, J. (2020, June 23). Independent Review Charter: SPECT. Parkinson's Progression Markers Initiative. Retrieved June 20, 2022, from https://www.ppmi-info.org/sites/default/files/docs/PPMI2.0_002_SPECT_IRC_Final_FullyExecuted_v1.0_20200730.pdf

Markus, A. F., Kors, J. A., & Rijnbeek, P. R. (2021). The role of explainability in creating trustworthy artificial intelligence for Health Care: A Comprehensive Survey of the terminology, design choices, and evaluation strategies. Journal of Biomedical Informatics, 113, 103655. https://doi.org/10.1016/j.jbi.2020.103655

Mary Lou Cheal , Don R. Lyon & Lawrence R. Gottlob (1994). A framework for understanding the allocation of attention in location-precued discrimination, The Quarterly Journal of Experimental Psychology Section A, 47:3, 699-739. DOI: https://doi.org/10.1080/14640749408401134

Mishra, N. (2020, February 24). Canadian doctors use robot to perform surgery to treat brain aneurysm. Indus Scrolls. Retrieved December 5, 2022, from https://indusscrolls.com/canadian-doctors-use-robot-to-perform-surgery-to-treat-brain-aneurysm

Molnar, C. (2022). Interpretable machine learning: A guide for making Black Box models explainable. Christoph Molnar.

Moumene, Issam & Ouelaa, N. (2022). Gears and bearings combined faults detection using optimized wavelet packet transform and pattern recognition neural networks. The International Journal of Advanced Manufacturing Technology. 120. 1-20. 10.1007/s00170-022-08792-2

Orbach, J. (1962). Principles of neurodynamics. Perceptrons and the theory of brain mechanisms. Archives of General Psychiatry, 7(3), 218-219.

Pashler, H. E. (1998). The psychology of attention. The MIT Press. https://doi.org/10.7551/mitpress/5677.001.0001

Pilz, K. S., Roggeveen, A. B., Creighton, S. E., Bennett, P. J., & Sekuler, A. B. (2012). How Prevalent Is Object-Based Attention? PLoS ONE, 7(2), e30693. https://doi.org/10.1371/journal.pone.0030693

Posner MI, Snyder CR, Davidson BJ. Attention and the detection of signals. J Exp Psychol. 1980 Jun;109(2):160-74. PMID: 7381367

Pragati. (2021, August 29). How to change the learning rate in the PyTorch Using Learning Rate Scheduler? Knowledge Transfer. Retrieved December 16, 2022, from https://androidkt.com/change-learning-rate-in-the-pytorch-using-learning-rate-scheduler/

R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 618-626, doi: 10.1109/ICCV.2017.74.

S. Liu and W. Deng, "Very deep convolutional neural network-based image classification using small training sample size," 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), Kuala Lumpur, 2015, pp. 730-734, doi: 10.1109/ACPR.2015.7486599

SAE International, “Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles” [online] [Published: April 30, 2021], [Accessed: January 23, 2022] https://www.sae.org/standards/content/j3016_202104/

Shomstein, S., & Behrmann, (2008). Object-based attention: Strength of object representation and attentional guidance. Perception & Psychophysics, 70(1), 132–144. https://doi.org/10.3758/pp.70.1.132

Simonyan, K., & Zisserman, A. (2015, April 10). Very deep convolutional networks for large-scale image recognition. arXiv.org. Retrieved October 16, 2022, from https://arxiv.org/abs/1409.1556

Sophie Burkholder, “A new bill could put fully autonomous vehicles on PA roads. Does it pass the business ethics

test?” [online], [Published: January 6, 2022] [Accessed: January 23, 2022] https://technical.ly/2022/01/06/ethics-driverless-autonomous-vehicles-test/

Soto, D., & Blanco, M. J. (2004). Spatial attention and object-based attention: a comparison within a single task. Vision Research, 44(1), 69–81. https://doi.org/10.1016/j.visres.2003.08.013

Staden, Joshua & Brown, Dane. (2021). An Evaluation of YOLO-Based Algorithms for Hand Detection in the Kitchen. 1-7. 10.1109/icABCD51485.2021.9519307.

Tagaris, A., Kollias, D., Stafylopatis, A., Tagaris, G., & Kollias, S. (2018). Machine learning for neurodegenerative disorder diagnosis — survey of practices and launch of Benchmark Dataset. International Journal on Artificial Intelligence Tools, 27(03), 1850011. https://doi.org/10.1142/s0218213018500112

Thompson, P. M., Sowell, E. R., Gogtay, N., Giedd, J. N., Vidal, C. N., Hayashi, K. M., Leow, A., Nicolson, R., Rapoport, J. L., & Toga, A. W. (2005). Structural MRI and Brain Development. International Review of Neurobiology, 285–323. https://doi.org/10.1016/s0074-7742(05)67009-2

Univerzitet u Banjoj Luci, Svetlana & Luci, Banjoj & Fakultet, Filozofski & Petra, Bulevar & Gvozdenović, Vasilije. (2013). IS ATTENTION NECESSARY IN PERCEPTION?

Vivswan Shitole, Li Fuxin, Minsuk Kahng, Prasad Tadepalli, Alan Fern. One Explanation is Not Enough: Structured Attention Graphs for Image Classification. https://doi.org/10.48550/arXiv.2011.06733

Waa, J., Nieuwburg, E., Cremers, A., & Neerincx, M. (2021). Evaluating xai: A comparison of rule-based and example-based explanations. Artificial Intelligence, 291, 103404. https://doi.org/10.1016/j.artint.2020.103404

What are Recurrent Neural Networks? (IBM Cloud Ed.). Retrieved November 10, 2020, from https://www.ibm.com/cloud/learn/recurrent-neural-networks

Wilck, A. M., & Altarriba, J. (2019). Attention and Perception. Encyclopedia of Evolutionary Psychological Science, 1-6. doi:10.1007/978-3-319-16999-6_637-1

Zesiewicz, T. A., Sullivan, K. L., & Hauser, R. A. (2006). Nonmotor symptoms of Parkinson's disease. Expert review of neurotherapeutics, 6(12), 1811–1822. https://doi.org/10.1586/14737175.6.12.1811

Zhang, X., He, L., Chen, K., Luo, Y., Zhou, J., & Wang, F. (2018). Multi-View Graph Convolutional Network and Its Applications on Neuroimage Analysis for Parkinson's Disease. AMIA ... Annual Symposium proceedings. AMIA Symposium, 2018, 1147-1156.

Zhao, Y., Cumming, P., Rominger, A., Zuo, C., Shi, K., Wu, P., Wang, J., Li, H., Navab, N., Yakushev, I., Weber, W., Schwaiger, M., & Huang, S. C. (2019). A 3D Deep Residual Convolutional Neural Network for Differential Diagnosis of Parkinsonian Syndromes on 18F-FDG PET Images. Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual International Conference, 2019, 3531–3534. https://doi.org/10.1109/EMBC.2019.8856747

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning Deep Features For Discriminative Localization. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Published. https://doi.org/10.1109/cvpr.2016.319