NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PROGRAM OF POSTGRADUATE STUDIES

MSc THESIS

On visual explanation of supervised and self-supervised learning

Dimitrios A. Reppas

ATHENS

DECEMBER 2022

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Πάνω στην οπτική εξήγηση της επιβλεπόμενης και αυτοεπιβλεπόμενης μάθησης

Δημήτριος Α. Ρέππας

ΑΘΗΝΑ

ΔΕΚΕΜΒΡΙΟΣ 2022

MSc THESIS

On visual explanation of supervised and self-supervised learning

Dimitrios A. Reppas
A.M.:DS1200010

SUPERVISOR: Yannis Avrithis, Research Director, ATHENA Research and Innovation Center

EXAMINATION COMMITTEE:

Yannis Avrithis, Research Director, ATHENA Research and Innovation Center

Ioannis Emiris, President and General Director, ATHENA Research and Innovation Center

Stephane Ayache, Professor and Research Scientist, Aix-Marseille University

December 2022

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Πάνω στην οπτική εξήγηση της επιβλεπόμενης και αυτοεπιβλεπόμενης μάθησης

Δημήτριος Α. Ρέππας
Α.Μ.:DS1200010

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: Γιάννης Αβρίθης, Διευθυντής Έρευνας, Ερευνητικό κέντρο ”ΑΘΗΝΑ”

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:

Γιάννης Αβρίθης, Διευθυντής Έρευνας, Ερευνητικό κέντρο ”ΑΘΗΝΑ”

Ιωάννης Εμίρης, Πρόεδρος και Γενικός Διευθυντής, Ερευνητικό κέντρο ”ΑΘΗΝΑ”

Stephane Ayache, Καθηγητής και Ερευνητής, Πανεπιστήμιο Μασσαλίας

Δεκέμβριος 2022

ABSTRACT

In recent years, the rapid development of *Deep Neural Networks* (DNN) has led to a remarkable performance in many *Computer Vision* tasks. The increasing complexity of the models, the computational power, the amount available of data and the *supervision* during the training process are the main causes behind this success. As an alternative to supervised representation learning, *self-supervised* methods are becoming popular in dispensing the need for carefully labelled datasets.

Undoubtedly, the more complex the models get, the greater is the need for understanding their predictions. The primary objective of this thesis is to interpret both supervised and self-supervised models, using either *convolutional neural networks* or *visual transformers* as a backbone. Variations of visualization methods are used, based on *class activation maps* (CAM) and *attention* mechanisms. Given an input image, these methods provide us with a *saliency map* that is used to interpret the network prediction. This map indicates the regions of the image that the model pays the most attention to. We evaluate these methods qualitatively and quantitatively. We further propose new alternative or complementary visualization methods, which show where important information can be hidden inside the network and how to reveal it. These new methods further improve the quantitative results. Our study highlights the importance of *interpretability*, shows some common properties and differences in the way supervised and self-supervised models make their predictions and provides valuable information on both the models and the visualization methods.

Thanks to the knowledge we gain from the interpretability study, we further improve self-supervised learning, in particular using *mask image modeling* (MIM). Here, we indicate the regions of an image that are most important to be hidden from a student network and define a more challenging MIM-based self-supervision pre-text task. Based on this, we propose new masking strategies that achieve higher k-NN, linear probing scores and acceleration in the learning process of downstream tasks. Considering the computational efficiency challenge these methods face, we conduct experiments on different scales of a dataset and number of training epochs and show their impact on the scores. Here, we further visually explain the influence of each *masking strategy* and scale of a dataset by using interpretability methods during the learning and evaluation process. Finally, we introduce a new loss function based on *contrastive learning* and achieve improvements over the baseline when used with different masking strategies.

SUBJECT AREA: Computer Vision, Deep Learning

KEYWORDS: Interpretability, Self-supervised Learning

ΠΕΡΙΛΗΨΗ

Τα τελευταία χρόνια, η ταχεία ανάπτυξη των Νευρωνικών Δικτύων Βάθους έχει επιφέρει αξιοσημείωτα επιτεύγματα σε πολλές εργασίες Όρασης Υπολογιστών. Η αυξανόμενη πολυπλοκότητα των μοντέλων, ο μεγάλος όγκος δεδομένων καθώς και η επίβλεψη κατά τη διάρκεια της εκπαίδευσης των μοντέλων είναι οι κύριες αιτίες πίσω από αυτήν την επιτυχία. Ως εναλλακτική λύση για την επιβλεπόμενη εκμάθηση αναπαράστασης, προτάθηκαν μέθοδοι αυτοεπίβλεψης για να απαλλαγούμε από τα υψηλά κόστη που απαιτούνται για την παραγωγή προσεκτικά επισημασμένων δεδομένων.

Αναμφίβολα, όσο πιο πολύπλοκα γίνονται τα μοντέλα, τόσο μεγαλύτερη είναι η ανάγκη κατανόησης των προβλέψεών τους. Ο πρωταρχικός στόχος αυτής της διατριβής είναι η ερμηνεία τόσο των επιβλεπόμενων όσο και των αυτοεπιβλεπόμεων μοντέλων, που βασίζονται σε αρχιτεκτονικές είτε συνελικτικών είτε οπτικών transformers δικτύων . Ποικιλία μεθόδων οπτικοποίησης χρησιμοποιείται, η οποία είναι βασισμένη σε χάρτες ενεργοποίησης κλάσεων και μηχανισμούς προσοχής. Δοθείσας μίας εικόνας εισόδου, αυτές οι μέθοδοι μας παρέχουν με ένα χάρτη σημαντικότητας που χρησιμοποιείται για την ερμηνεία της πρόβλεψης του δικτύου. Αυτός ο χάρτης υποδεικνύει τις περιοχές της εικόνας στις οποίες το μοντέλο δίνει τη μεγαλύτερη προσοχή. Οι μέθοδοι που χρησιμοποιούμε αξιολογούνται τόσο ποιοτικά όσο και ποσοτικά. Προτείνουμε περαιτέρω νέες εναλλακτικές ή συμπληρωματικές μεθόδους οπτικοποίησης, οι οποίες υποδεικνύουν πού μπορούν να κρυφτούν σημαντικές πληροφορίες μέσα στο δίκτυο και πώς μπορούν να αποκαλυφθούν. Οι νέες μέθοδοι βελτιώνουν περαιτέρω τα ποσοτικά αποτελέσματα. Η μελέτη μας υπογραμμίζει τη σημασία της ερμηνευσιμότητας, δείχνει ορισμένες κοινές ιδιότητες και διαφορές στον τρόπο με τον οποίο τα επιβλεπόμενα και αυτοεπιβλεπόμενα μοντέλα κάνουν τις προβλέψεις τους και παρέχει πολύτιμες πληροφορίες τόσο για τα μοντέλα όσο και για τις μεθόδους οπτικοποίησης.

Χάρης των γνώσεων που αποκομίσαμε από τη μελέτη πάνω στην ερμηνευσιμότητα, βελτιώνουμε περαιτέρω την αυτοεπιβλεπόμενη μάθηση, ιδίως χρησιμοποιώντας τη μοντελοποίηση κάλυψης εικόνας. Εδώ, υποδεικνύουμε τις περιοχές μιας εικόνας που είναι πιο σημαντικό να αποκρύπτονται από ένα δίκτυο μαθητή και ορίζουμε μια πιο αποδοτική μέθοδο αυτοεπίβλεψης που βασίζεται στη μοντελοποίηση κάλυψης εικόνας. Με βάση αυτό, προτείνουμε νέες στρατηγικές κάλυψης που επιτυγχάνουν υψηλότερα k-NN και liner probing αποτελέσματα και επιτάχυνση στη διαδικασία εκμάθησης σε εργασίες που ακολουθούν. Λαμβάνοντας υπόψη την πρόκληση υπολογιστικού κόστους που αντιμετωπίζουν αυτές οι μέθοδοι, διεξάγουμε πειράματα σε διαφορετικές κλίμακες ενός συνόλου δεδομένων και αριθμού εποχών εκπαίδευσης και δείχνουμε το αντίκτυπό τους στα αποτελέσματα. Εδώ, εξηγούμε περαιτέρω οπτικά την επιρροή κάθε στρατηγικής κάλυψης και κλίμακας ενός συνόλου δεδομένων χρησιμοποιώντας μεθόδους ερμηνείας κατά τη διαδικασία μάθησης και αξιολόγησης. Τέλος, εισάγουμε μια νέα συνάρτηση απώλειας που βασίζεται στην αντιθετική μάθηση και επιτυγχάνουμε βελτιώσεις σε σχέση με τη βασική συνάρτηση όταν χρησιμοποιείται με διαφορετικές στρατηγικές κάλυψης.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Όραση Υπολογιστών, Βαθιά Μάθηση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Ερμηνευσιμότητα, Αυτοεπιβλεπόμενη Μάθηση

ACKNOWLEDGEMENTS

More than half of this work was conducted in the Laboratoire d'Informatique et Systèmes (LIS) research lab at Ecole Centrale Marseille (ECM) and the rest in Athens. Part of this work was funded by the French national research agency (ANR) UnLIR project (ANR-19-CE23-0009) and HPC resources were used from GENCI-IDRIS (Grant 2020-AD011013552). Extra resources were provided by the LIS lab as well.

First and foremost, I would like to thank my supervisor Yannis Avrithis for his support, valuable guidance, endless motivation and trust he showed me. Special thanks to the rest members of this work Stephan Ayache, Ronan Sicre, Hanwei Zhang, Felipe Torres and Shashanka Venkataramanan for their extra assistance and motivation. In addition, I am grateful to all the people at LIS research lab for being so hospitable, friendly and for creating such a pleasant and warm working environment. In closing, I would like to thank my family, friends, and Tania for being by my side throughout this project.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Computer vision

For years, people have been dreaming of machines with human intelligence that can think and act like human beings. One of the most exciting ideas was to create machines that could see and similarly interpret the world as humans do. The imagination of the past has become a reality nowadays to some extent. Thanks to advances in *artificial intelligence* and computational power, computer vision is progressively integrated into our daily lives.

Computer Vision is the field that focuses on how to make machines process and gain a high-level understanding of visual data in a similar way human vision works. Visual data can be images, video sequences, views from multiple cameras, multi-dimensional data from a 3D scanner, etc. With the acquired knowledge, computers can take actions or make recommendations. A few common computer vision tasks are the following:

- *Image classification:* Given an input image, the computer can predict accurately its certain class. For instance, if we give a computer an image depicting a dog, it can classify this image into the ”dog” category.
- *Object detection:* In this task, the computer deals with detecting objects of a certain class in visual data. Given an input image depicting four dogs, the computer can detect them all and provide their exact location on an image. Face detection and pedestrian detection are two well-researched domains of object detection.
- *Object tracking:* Here, the objects are initially detected and uniquely identified in a frame of video and then their trajectory is tracked. In autonomous cars, for instance, objects such as pedestrians and other cars are tracked to avoid collisions and obey traffic laws.
- *Content-based image retrieval:* The computer in this task deals with the problem of searching for visual data in large databases. The search is based on the content of the image rather than using metadata tags associated with them. In other words, if we query a database given an input image, similar images to the input will be retrieved. Through content-based image retrieval, manual image tagging can be replaced by automatic image annotation.

The aforementioned computer vision tasks include methods for acquiring, processing, analyzing and understanding visual data and the extraction of high-dimensional data representations which are further used, by computers, to automate predictions and recommendations. The way one computer provides recommendations and predictions relies on algorithms and machine learning strategies. Several factors contribute to the success of computer vision, including machine learning which we describe in the next section.

1.2 Machine learning

Technology has become an integral part of our daily lives. The continuous production of data, the increase of computation power in the past few years and the development of better algorithms result in machine learning being all around us, from autonomous cars, language translation machines, chatbots, etc. Probably, the general public has already used devices utilizing such technologies. For example, a smart watch that uses fitness tracker technology, or an intelligent home assistant like Google Home. As we input more

data into these machines, this helps them to gain a kind of experience, thus improving the delivered results. When we ask Alexa to play our favourite music station, she will choose to play the music station we play most often. Our listening experience can further be improved by telling Alexa to adjust the volume, skip songs, and many other possible commands. All of this is possible thanks to machine learning and the rapid advancement of artificial intelligence.

Machine learning is considered a branch of artificial intelligence (AI) field. AI tends to mimic human intelligence. To be more specific, its technologies are used to perform and optimize tasks that humans have historically achieved, such as decision-making, facial recognition and speech translation. Machine learning is a domain of study that uses computational algorithms for a variety of daily tasks. The purpose of machine learning is first to gain an understanding of the structure of data and then to fit that data into models that can be utilized for decision-making processes. Machine learning differs from traditional computational approaches. In traditional computing, the algorithms used by a computer for problem-solving are sets of explicitly programmed instructions. On the other hand, the algorithms used in machine learning allow computers to be trained on input data and use statistical analysis to output values that fall within a specific range. As a result, machine learning facilitates computers in building models from input data that can be applied to automate decision-making processes. In other words, these algorithms enable a computer to learn from data and even improve itself, without being explicitly programmed to do so.

Machine learning is categorized into four different categories based on the way an algorithm learns from data. The algorithm and the type of learning we choose, depends on the type of data and the task we want to complete. The four basic types of learning are the following:

- *Supervised learning:* In this type of learning, the available dataset consists of labelled examples, meaning that each data point contains an associated label. The purpose of supervised learning algorithms is to learn a function that maps inputs to labels, based on example input-label pairs. During training, for each input associated with a true label, the algorithm outputs a predicted label that is compared with the given true label. Here, there is an error function, called *loss function*, which measures the distance between the true and predicted labels. Learning is achieved by passing the training data many times to the algorithm and by minimizing the loss function between the true and predicted labels. Through this process, the *parameters* of the function, that is being learned, are changing each time we pass the data. The learned function also called inferred can be used then for mapping new examples. Ideally, through the inferred function the algorithm correctly determines the class labels for unseen instances. For the ideal scenario, the learning algorithm needs to *generalize* from the training data to unseen situations in a ”reasonable” way. The generalization ability of an algorithm is measured through *accuracy* and error metrics. Typical types of supervised learning are classification and regression problems.
- *Unsupervised learning:* This type of learning uses unlabeled data. Here, the algorithm finds commonalities among its input data. Machine learning methods that facilitate unsupervised learning are particularly valuable having in mind that labelled data is less abundant than unlabelled data. The straightforward purpose of unsupervised learning is to discover hidden patterns within a dataset. For instance, unsupervised models are ideal for exploratory data analysis, cross-selling strategies, customer segmentation, etc. A secondary goal is feature learning which allows the machine to discover the representations that are needed to classify raw data.

- *Semi-supervised learning:* The type of learning that falls between supervised learning and unsupervised learning is called semi-supervised learning. During the training procedure, the classification and feature extraction from a large and unlabeled dataset is guided by a smaller labelled dataset. When not having enough labelled data for supervised learning, or if it is too costly to label enough data, semi-supervised learning is a good alternative.
- *Reinforcement learning:* To perform this type of learning, three major components take part in this process which are the agent, the environment and the actions. The agent is the decision-maker. Everything that the agent interacts with, is the environment. Actions are called whatever the agent does. The agent is able to perceive and interpret its environment by taking actions and learning through trial and error. More precisely, the agent interacts with its environment and receives rewards for performing correctly and penalties for performing incorrectly. By maximizing its reward and minimizing its penalty, the agent learns without intervention from a human. Reinforcement learning is a type of dynamic programming in which an algorithm is trained using a system of reward and punishment. This kind of machine learning is found in robotics, video games, healthcare applications, etc.

1.3 Deep learning

The wide public often tends to confuse deep learning and machine learning and use them interchangeably. However, this is not the case as deep learning is only a sub-field of machine learning. The accelerating progress in areas such as computer vision, natural language processing and speech recognition is credited to deep neural networks. Applications powered by deep neural networks are now used by the majority of people. These networks have also become powerful tools for many scientific fields, such as bioinformatics, medicine and astronomy which usually involve a massive amount of data.

Deep learning has evolved from neural networks. Neural networks, or artificial neural networks (ANN), consist of node layers, also called artificial neuron layers. There is always an input layer, one or more hidden layers and an output layer. If a model consists of more than one hidden layer, it is called a deep neural network and if it has exactly one hidden layer it is called a neural network.

The basic purpose of these networks is to learn good representations from large amounts of raw data, which can then be used for several tasks. Deep neural networks are trained in a similar way as other machine learning algorithms and learning can be supervised, semi-supervised or unsupervised. During training, data associated with true labels are passed through the network multiple times while the distance between the true and predicted labels is progressively minimized by a loss function. After each pass of the data through the network, an *epoch* is completed and the weights of the network are updated. These weights, also called parameters, are found on each connection between the neurons. After the training process, the network with its learned parameters will be used to map unseen instances to certain labels. How good or bad a network generalizes on unseen examples is measured again through accuracy and error metrics.

A good question would be here when to use machine learning and when deep learning algorithms. Machine learning and deep learning differ primarily in the type of data used. Using structured data is crucial for machine learning algorithms and, here, there is a need for domain experts during data preparation. On the other hand, deep-learning algorithms

are autonomous and learn from raw data without the need for human intervention. While deep learning algorithms show their superiority over machine learning algorithms in several tasks, it is needed much more training samples for deep learning to achieve such performance. In general, deep learning is being used in more complex use cases. If the problem is simple, it is often preferred to use machine learning.

1.4 Challenges and motivation

Self-supervision The objective of deep learning is to learn good representations from raw data. While supervised models achieve this by being trained on datasets that consist of labelled examples, this is not the fact for *self-supervised* methods. These methods learn representations from data that are not associated with manual data labelling procedures. Self-supervised learning is regarded as an intermediate category between supervised and unsupervised learning. Although a labelled dataset is not required, this type of learning is being achieved with the use of several pseudo-labels created to accomplish a simple task which is based on the dataset and is predefined by us. This task is known as the pre-text task. For instance, given an input image, we can start to rotate it by 90° and by this way we create four augmented views of the image and the respective pseudo-labels 0°, 90°, 180°, and 270°. By using as a pre-text task the prediction of the rotation of one image, the model can progressively learn a good representation of the image. Another pre-text task in self-supervised learning is to mask some regions of an image and based on the original image reconstruct the hidden regions during learning, etc. Self-supervised models become popular since the high cost in time and money needed for the preparation of good quality labelled data is eliminated and because these methods are one step closer to embedding human cognition in machines.

Although these methods produce models that outperform sometimes even supervised models when transferred to a variety of downstream tasks, there are some serious challenges these methods face, related to the accuracy, computational efficiency and pre-text task. First and foremost, reaching high accuracy scores by using pseudo-labels is not as easy as using a good quality manually labelled dataset. To achieve competitive scores, the models trained in a self-supervised way need more data than supervised models. Considering the computation efficiency, the time needed for training a model is higher compared to supervised learning. Finally, choosing the right pre-text task and way to take advantage of a specific unlabeled dataset plays a vital role.

Interpretability Over the last few years, deep neural networks (DNN) have achieved remarkable success in natural language processing, speech recognition, computer vision and other fields. They have not only overcome many previous machine learning performances such as decision trees and support vector machines but also achieved state-of-the-art performances on specific real-world tasks.

However, deep learning still has some serious obstacles to overcome. To reach better performance, the models tend to become extremely complicated with millions of free parameters. In these complicated networks, unexpected behaviours are often observed. For instance, it has been shown that an arbitrary change in the prediction of a network could come from applying a certain imperceptible change to the input image. The modified input is called adversarial example and is a good indication that their underlying mechanisms have not yet been well understood. For this reason, deep neural networks are often re-

ferred to as ”black boxes”. There are many unsolved questions about the performance of the model. For example, why does the model generalize well or not and why is a model fooled by an adversarial example?

The more complex a model, the more accurate the predictions can be, but explaining to an individual how the output was determined can be difficult. Banking and insurance companies, for example, use simple machine learning models because they need to explain how every decision was made. Preferring to use simple machine learning while having state-of-the-art deep neural networks only because the prediction of these networks is not well understood, is an obstacle that must be overcome.

Based on these challenges, many researchers started to focus on model interpretability. A variety of interpretability methods are proposed to further investigate the inner mechanism of deep neural networks and better understand the way these models predict, although there is still room for improvement. For instance, for a simple image classification task, visualization tools are used to generate kind of heat maps, given an input image to the model, to indicate the regions the model takes mostly into account before its prediction for a certain class. Finally, which interpretability method to choose is based on the data and the model you want to interpret and is an extra challenge.

1.5 Our work

The aforementioned challenges show us the importance of the interpretability of deep neural networks and motivate us to further investigate the domain. Here, we choose to explore visualization methods used for the interpretability of pre-trained models; therefore, there is no need to train the models from scratch to interpret them. In this way, we have more freedom to conduct an extensive study on interpretability. We start this work, by providing visual explanations for a variety of deep neural networks pre-trained on the same benchmark dataset. The networks are based on different backbone architectures and are trained either in a supervised or a self-supervised way. To interpret the models, we use several baseline visualization methods. We further propose new methods that can be used as alternatives and complementary tools to better understand the networks. Through this study, we want to explore the inner mechanism of the networks and the visualization tools we use to interpret them. As this study is for both supervised and self-supervised models, an extra motivation here is to show common properties and differences in the way both models predict. Insights from this study are used to further improve a self-supervised approach, based on a pre-text task in which masked regions of an input image are recovered during training to achieve representation learning.

Motivated by the self-supervision challenges and thanks to the knowledge we gained from the interpretability study, we dedicate the rest of our work to finding ways to improve a self-supervised approach that is based on the recovery of masked regions of images. In this approach, first, some portions of an image are masked and then, during learning, these portions are recovered always by taking into account the respective non-masked coming from the original input image. The recovery of the masked regions of an image is used as a way for representation learning. A good question would be here, which portions of the image to mask to achieve better learning and a more competitive pre-text task. After long experimentation on this, we find an answer and propose new masking strategies that provide better accuracy scores than previous baseline strategies and acceleration in the learning process of downstream tasks.

Considering the accuracy and computational efficiency challenges, we conduct experiments on different scales of a standard benchmark dataset for various training durations and see the impact on accuracy scores. Then, the influence of the masking strategy and scale of a dataset is explained visually by the use of interpretability methods during the learning and evaluation process.

In every type of learning and network architecture, choosing the right loss function is always a challenge. Based on this, we introduce a new loss function and conduct experiments to show its impact on accuracy scores. We compare its results with the ones obtained from the baseline loss function, for models trained with different masking strategies.

1.6 Structure

In this section, we present the structure of this work. The chapters are organized as follows:

- *Chapter 1* gives an introduction to Computer Vision, Machine Learning, and Deep Learning fields while presenting the main challenges that motivate us to conduct this study.
- *Chapter 2* presents state-of-the-art deep neural networks that take part in this study. The networks are CNNs and transformers and are trained either in a supervised or self-supervised way. Then, it highlights the importance of interpretability, categorizes the methods used for the visual explanation of the predictions of the models and presents the ones we are using.
- *Chapter 3* presents our study on interpretability domain. More precisely, it detailed describes the methodology we follow. It shows our contributions and the experimental setup we use. It ends with the experimental results along with our conclusions.
- *Chapter 4* presents our study on MIM-based self-supervised methods. It shows the methodology and our contributions, the experimental setup and results along with the final conclusions we reach.
- *Chapter 5* gives the common points of the previous two chapters and comes up with an overall conclusion. Finally, it provides some future proposals.

2. BACKGROUND

This chapter is a summary of the basic concepts, domain of research and knowledge we use to fulfil the purposes of this work. The provided information is closely related to computer vision, deep learning and interpretability domains. More precisely, we provide a general overview of deep neural networks and give valuable information about their functionality. Here, we start with the fundamental architectures and dive into CNNs and transformers which are the two families of networks we use in this study. We further emphasize the difference between supervised and self-supervised learning and describe the architectures we use. We then proceed to the interpretability of deep neural networks. Here, we highlight its importance and present its basic categories. In the end, we describe the visualization methods we use. References and related work are attached to the entire chapter.

2.1 Deep neural networks

Deep learning is a branch of the machine learning methods and based on artificial neural networks with representation learning. A deep neural network (DNN) use an artificial neural network (ANN) as a backbone and has multiple layers between the input and output layers. There is a great variety of deep neural networks but all share five major components: neurons, connections, weights, biases, and functions. These components function similar to the human brain and can be trained like any other machine learning algorithms. Learning can be supervised, semi-supervised or unsupervised. Maybe the most famous deep learning algorithms nowadays are multilayer perceptrons (MLPs), convolutional neural networks (CNNs), transformers, long short term memory networks (LSTMs), recurrent neural networks (RNNs), generative adversarial networks (GANs) and autoencoders. In this work, we use CNNs and transformers. In the following subsections, we describe those two families of networks. Before diving into this, there is a need to briefly describe the fundamental architecture of deep neural networks which is *perceptron*.

2.1.1 How it started

Perceptron Perceptron, introduced by Frank Rosenblatt in 1962, is arguably the foundation of deep neural networks. Perceptron is a binary linear classifier that is capable of learning linearly separable patterns. Given an input $x \in \mathbb{R}^d$, perceptron is a generalized linear model

$$y = f(x; w) = \text{sign}(w^\top x), \tag{2.1}$$

where $w \in \mathbb{R}^d$ is a weight vector to be learned and

$$\text{sign}(x) = \begin{cases} +1, & x \geq 0 \\ -1, & x < 0 \end{cases}. \tag{2.2}$$

An input x is classified to the C_1 class if $y = 1$ and to the C_2 class if $y = -1$. Given a target variable $s \in \{-1, 1\}$ and an training sample $x \in \mathbb{R}^d$, x is correctly classified if the output y equals s. In the basic perceptron formula (2.1), apart from the $x_1, \ldots, x_n$ inputs, there is also a constant term $x_0 = 1$. The addition of the constant term x_0 incorporates an extra

weight coefficient w_0 to the formula, which is called bias b. Taking into account the bias term b the (2.1) is generalized to:

$$y = f(x; w, b) = \text{sign}(w^\top x + b). \tag{2.3}$$

The perceptron algorithm could be explained with Figure 1. Training samples $x_1, \ldots, x_n$ are given to the network along with their respective target variables $s_1, \ldots, s_n \in \{-1, 1\}$. Starting from an initial weight vector $w^{(0)}$, the algorithm learns iteratively by updating the weight vector following the rule $w^{(t+1)} \leftarrow w^{(t)} + \epsilon s_i x_i$, where ϵ is the learning rate. The weight vector is updated only if the input sample x_i is misclassified. When all the input samples x_i are classified correctly, the learning process stops and the final weight vector indicates the decision boundary.

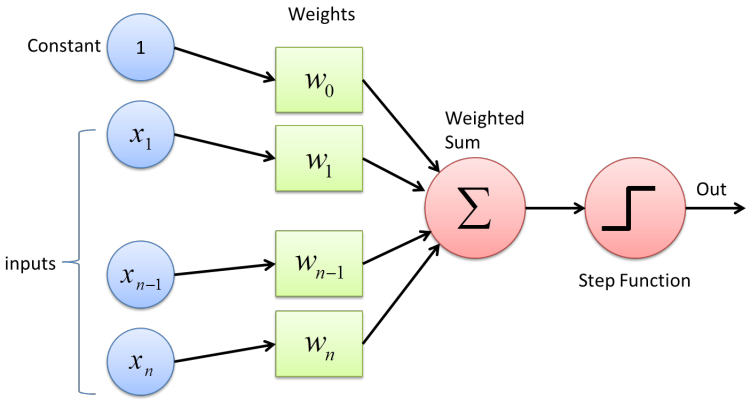


Figure 1: The architecture of perceptron [1].

Multilayer perceptron The story of deep neural networks continues with *multilayer perceptrons* (MLPs). In comparison with perceptron, MLPs have more than one neuron and the hidden layers can be more than one and thus changing the overall depth of the model. Each of the neurons is a perceptron and these models can be seen as efficient nonlinear function approximators. In each neuron, there is a non-linear function called *activation function* that should be chosen wisely. Examples of activation functions are the *step function* introduced in perceptron (2.2), the *sigmoid*, the *rectified linear unit*, the *hyperbolic tangent*, etc. If x is an input and f^* the approximation function, then a classifier $y = f^*(x)$ maps x to a category y, as in perceptron. In an MLP, the mapping is defined as $y = f(x; \theta)$, where θ are the learnable parameters that result in the best approximation function. The function f is composed of many different functions that are associated together according to a directed acyclic graph. Each function can be seen as a new representation of the previous one. In a naive example, where the MLP has two layers, the function is given by $f(x) = f^{(2)}(f^{(1)}(x))$, in which $f^{(1)}$ represents the first layer and $f^{(2)}$ the second layer. The functions are connected in a chain and the total length of this chain indicates the depth of the MLP. The first layer of this network is called input layer and the last output layer. The intermediate layers are the hidden layers. The general architecture can be seen in Figure 2.

During training, the objective is $f(x)$ and $f^*(x)$ to be matched by progressively minimizing a loss function. For different inputs x, we get approximate examples of $f^*(x)$. Each given input x is accompanied by a target label $y \approx f^*(x)$. As only the output of the last layer is specified, the algorithm should decide how to use the hidden layers and which approximation function f^* to implement. The weights of the network are updated by using

the *back-propagation* algorithm. This algorithm first calculates all the gradients and then back-propagates them within the network.

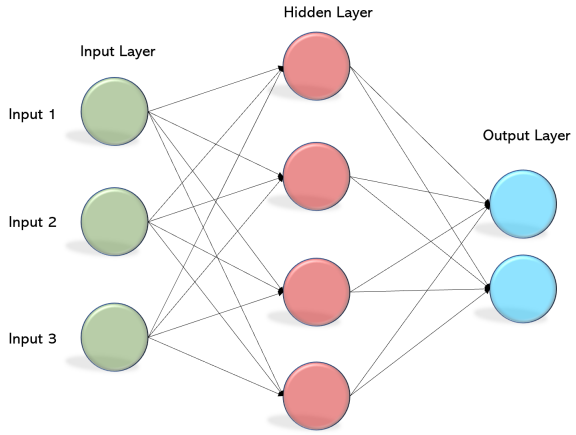


Figure 2: The architecture of MLP [2].

2.1.2 Convolutional neural networks

After the short description of the two fundamental architectures of DNNs, a quick overview of the convolutional neural networks (CNNs) follows. CNNs can be seen as regularized versions of multilayer perceptrons (MLPs). MLPs usually mean fully connected networks. In other words, each neuron in one layer is connected to all neurons in the next layer. The ”full connectivity” of these networks makes them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training, with weight decay, and trimming connectivity by applying skipped connections, dropout, etc. CNNs take a different approach toward regularization. They take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their convolutional filters also called kernels.

The main block of the architecture of a CNN is the convolutional layer. This layer consists of a bunch of kernels that are used to extract feature maps for a given input. Through the convolution operation, each kernel slides across the height and width of the input image and the dot product of the kernels and the image are computed at every spatial position. The output feature maps from the convolution operation are expressed by:

$$y_i = b_i + \sum_{x_i \in x} W_{ij} * x_i, \tag{2.4}$$

where $W_{ij} \in \mathbb{R}^{F \times F \times C}$ are the kernels, x_i is the spatial position in an image and b_i the bias term of the i_{th} filter. The total number of filters is D and equals the number of feature maps. This number indicates the depth of the convolutional layer. The size of W_{ij} is determined by a selected receptive field F and its input image depth C.

CNNs are used to process and extract features from data and consist of multiple layers. These networks have *rectified linear unit* layers (ReLU) to exclude negative values from the output feature maps. Then, there are the *pooling* layers that perform a down-sampling operation to reduce the dimensions of the feature map. The output of the pooling operation can be flattened and passed to a fully connected layer (MLP) from which the final predictions of the model are obtained. The general philosophy behind these networks is

that the deeper we go, the more filters we find but with decreased dimensions because of the pooling operations.

The overall architecture of CNNs results in some properties, such as sparse interactions, parameter sharing and equivariant representations. These networks do not have the ”full connectivity” of MLPs; therefore, fewer parameters need to be stored as also fewer operations are required. Considering the parameter sharing, the output of a convolutional layer has a depth component and if we partition each segment of the output, we will obtain a 2D plane of a feature map. Across the same 2D planes all the filters share their weights. Finally, the property of equivariance means that if the input changes, the output changes in the same way.

The story of CNN While many believe LeNet [20] is the first CNN model, in 1980 a biologically-inspired network is introduced that uses convolutions. The name of the model is neocognitron [21]. LeNet [20] was the first one to combine the ideas of sparse interactions, parameter sharing and equivariant representations in a learning scheme to further improve the performance of neural networks. The network was implemented in computer vision recognition tasks. It has 5 layers, the two of them are convolutional and the rest fully-connected layers. An improved CNN was then AlexNet [22], which had 5 convolutional and 3 fully connected layers. It was the first network to use ReLU as an activation function. Two years after, Inception v1 [23] was the best CNN by using an increased depth of 22 layers. This model achieved better performance than AlexNet while having 25 times fewer parameters. The secret behind this was the introduction of an Inception module. This module consists of a feature-wise concatenation of 3 convolutions and 1 max pooling layer in its naive version. To reduce the computation complexity and dimensionality, $1x1$ kernels were used as bottlenecks. The second version of this model, Inception v2 [24], was even better by applying *batch normalization*, which was first introduced [25]. This technique regularizes the models, avoids the training to get stuck in poor local minima, makes the training process to be influenced less by the parameter scale and allows higher learning rate values. Inception v3 introduces another idea, in which there is a factorization of 7×7 and 5×5 convolutions into three and two 3×3 convolutions respectively. In this approach, a reduction of parameters is succeeded because the weight between adjacent tiles is shared. The last CNN we present is the *residual neural network* (ResNet) [3]. Before ResNets, as shown previously, the architectures tend to stack more and more layers to achieve improvements in scores. In [3], it is highlighted the *vanishing gradients* problem. As the gradient is back-propagated from the deeper layers to the shallower ones, the gradients may reach infinitely small numbers by the repeated multiplications, when the network consists of many layers. To face this problem, the “identity shortcut connection” is introduced also called *residual block*. As shown in Figure 3, there is skip of connections between one or more layers. The vanishing gradients problem when going deeper is mitigated thanks to residual blocks and the ResNets achieve the best performance in ILSVRC 2015.

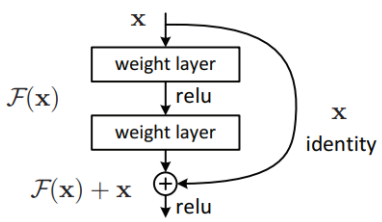


Figure 3: The residual block [3].

In this work, we use ResNet 50 in our study on interpretability. To be more specific, we use pre-trained ResNet 50 models, trained in a supervised and self-supervised way, to interpret their predictions with CAM-based visualization methods. The self-supervision of ResNet 50 is made with DINO [8] and MoCo v3 [4] approaches.

2.1.3 Transformers

Architectures based on *self-attention*, in particular transformers [26], are currently state-of-the-art methods in almost all natural language processing (NLP) tasks. Inspired by this success, the research community introduced these models to the computer vision domain as well. Some works combine CNNs with self-attention [6, 27–30] while others replace the convolutions entirely [4, 5, 31, 32]. Here, we take a closer look at the transformer architectures used in this work.

ViT [4] While convolutional neural network still remain the most standard approach on computer vision tasks, a *vision transformer*, called ViT [4], was introduced in 2020 and had comparable image recognition results with ResNets on several benchmarks. The model reached these results when pre-trained on large amounts of data (ImageNet-21k [33], JFT-300M [34]) and transferred to well-known small or medium-size benchmarks datasets (ImageNet [12], CIFAR-100 [35]).

The ViT philosophy is inspired by [26] and treats 16×16 image patches (tokens) the same way as word tokens in NLP. Given an input image $X \in \mathbb{R}^{h \times w \times c}$, where $h \times w$ is the spatial resolution and c is the number of channels, the first step is to tokenize it. This means to convert it to a sequence of $n = hw/p^2$ non-overlapping patches $P_i \in \mathbb{R}^{p \times p \times c}$ for $i = 1, \ldots, n$, where $p \times p$ is the patch resolution. Each patch is then flattened into a vector in $\mathbb{R}^{p^2 c}$ and projected to an embedding vector $z_i \in \mathbb{R}^d$ using a linear layer, where d is the embedding dimension. An extra learnable embedding $z^{[\text{cls}]} \in \mathbb{R}^d$, known as "classification token" [cls], is then added to the previous embedding vector to form the *tokenized image*

$$Z = (z^{[\text{cls}]}; z_1; \ldots; z_n) \in \mathbb{R}^{(n+1) \times d}, \tag{2.5}$$

where ";" denotes row-wise stacking and $n + 1$ the total number of tokens including [cls]. The role of the [cls] token will be to represent the image at the output layer. A sequence of position embeddings is also added to Z to retain positional information. The resulting sequence is the input of the transformer encoder.

The key building block of this encoder is the self-attention mechanism. This module operates on an input matrix $Z \in \mathbb{R}^{(n+1) \times d}$, where $n + 1$ is the number of tokens, each of dimensionality d. The input Z is linearly projected to *queries* $Q = ZW_q$, *keys* $K = ZW_k$ and *values* $V = ZW_v$, using the weight matrices $W_q \in \mathbb{R}^{d \times d_q}$, $W_k \in \mathbb{R}^{d \times d_k}$ and $W_v \in \mathbb{R}^{d \times d_v}$, where $d_q = d_k$. Keys and queries are used to compute the attention weights

$$A_{\text{ViT}}(K, Q) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right). \tag{2.6}$$

The Softmax is applied, such that each element of the $A_{\text{ViT}}(K, Q)$ lies in the range $[0, 1]$ and the sum of each row is equal to 1. The output of the self-attention operation is given by the weighted sum of the $n + 1$ token features in V, with the weights corresponding to $A_{\text{ViT}}(K, Q)$

$$O_{\text{ViT}} = A_{\text{ViT}}(K, Q)V. \tag{2.7}$$

The attention module repeats its computations multiple times in parallel inside one layer. To be more specific, the attention module splits its query, key, and value projections h ways and passes them independently through h separate self-attention operations called *attention heads*. From the attention heads, we obtain h attention calculations that are concatenated to produce a final attention output. This is called *multi-head attention* and is an extension of self-attention which gives the transformer greater power to encode multiple token relationships.

In addition to the multi-head self-attention, each layer of the transformer encoder is composed of skip connections, normalization layers and an additional multi-layer perceptron (MLP) block, as shown in Figure 4. A transformer encoder consists of B such layers, also known as blocks. Through all of its layers, the transformer encoder uses a sequence of fixed length $n+1$ of token embeddings of fixed dimension d, represented by a $(n+1) \times d$ matrix. From the output of the last encoder layer, only the embedding of the [cls] token is passed through a classification head. The classification head is implemented by a MLP with one hidden layer at pre-training and by a single linear layer at fine-tuning.

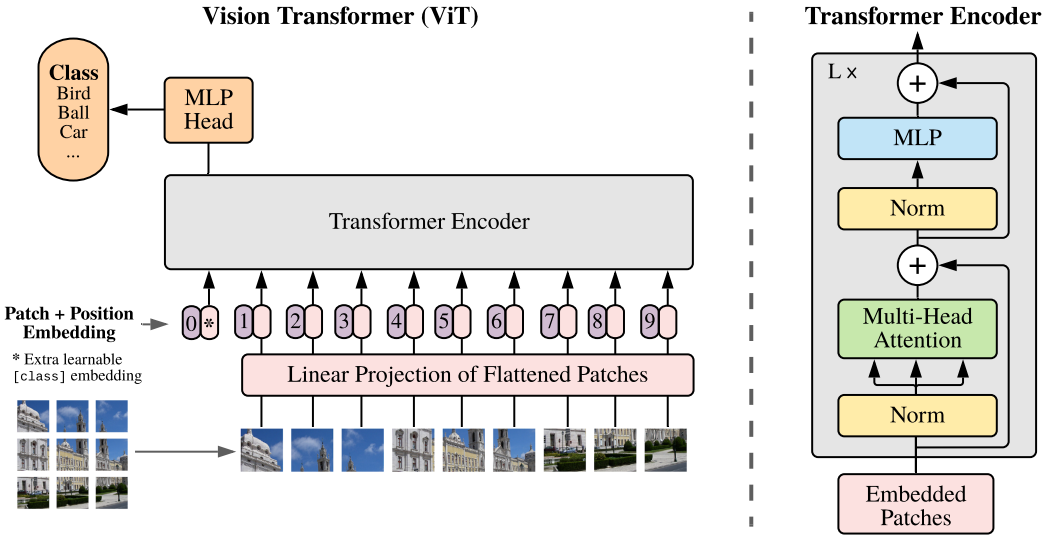


Figure 4: The ViT methodology and the transformer encoder overview [4].

In this work, in our study on interpretability, we use pre-trained ViT base models, trained in a supervised and supervised way, and in our study on self-supervised learning, we use a ViT-S/16 model as a transformer encoder of iBOT to train the model from scratch. The self-supervision of ViT base, in the interpretability study, is made with DINO [8] and iBOT [9] approaches.

DeiT [5] While ViT is competitive to convolutional neural networks when pre-trained on large and private datasets [33, 34], when trained on insufficient amounts of data, it does not generalize well. The superiority of convolutional neural networks when trained on less amount of data, is justified because transformers lack some of the inductive biases inherent to CNNs, such as translation equivariance and locality. In [5], it is proposed a data-efficient training procedure. Thanks to the novelty of a particular *distillation procedure*, DeiT reaches competitive results while not requiring a very large amount of data to be trained. Apart from the distillation procedure, specific augmentation, optimization and regularization techniques are followed to further improve the performance of the model.

The main architecture of the network remains the same as in [4], but there is an addition of an extra special token called *distillation token*, as shown in Figure 5. This token interacts

with the [cls] and patch tokens through the self-attention layers. It is employed in a similar way as the class token. The difference between the two special tokens is that on the output of the network the objective of the distillation token is to reproduce the (hard) label, predicted by a teacher network, while the class token tries to reproduce the true label. The teacher network is a convolutional neural network. Both class and distillation tokens are learned by back-propagation and contribute to the total loss. The final loss

$$L_{\text{DeiT}} = \frac{1}{2} L_{\text{CE}}(\text{Softmax}(Z_{\text{[cls]}}), y) + \frac{1}{2} L_{\text{CE}}(\text{Softmax}(Z_{\text{[dist]}}), y_t), \tag{2.8}$$

is constituted from two equally weighted *cross entropy losses* L_{CE}, where y is the true label, y_t the hard predicted label from the teacher network, $Z_{\text{[cls]}}$ the output class token embedding and $Z_{\text{[dist]}}$ the output distillation embedding. At test time, both the class and distillation embeddings contribute to the final model prediction. More specifically, the Softmax outputs by the two classifiers are fused (added) to make the final prediction of the label.

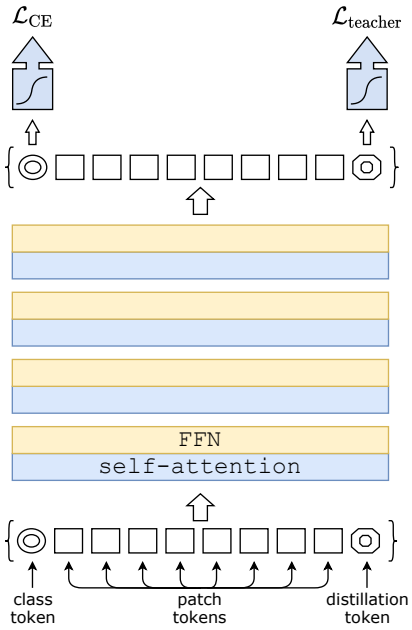


Figure 5: The DeiT architecture [5].

In this work, we use DeiT base in our study on interpretability. More precisely, we use pre-trained DeiT base models, trained in a supervised and self-supervised way, to interpret their predictions with CAM-based and attention-based visualization methods. The self-supervision of DeiT base is made with MoCo v3 [4] approach.

XCiT [6] The self-attention mechanism of transformers provides global interactions between all tokens and allows flexible image data modelling beyond the local interactions of CNNs. On the other hand, this flexibility has a negative impact on memory and time complexity. As a result, applications to high-resolution images and long sequences are hindered. In [6], a model, called XCiT, is proposed that uses a "transposed" version of self-attention (2.6). This self-attention version operates across feature channels rather than tokens. The interactions are based on the *cross-covariance matrix* between keys K and queries Q and the attention weights are given by:

$$A_{\text{XC}}(K, Q) = \text{Softmax}\left(\frac{\hat{K}^{\top}\hat{Q}}{\tau}\right), \tag{2.9}$$

where τ is a learnable temperature parameter which scales the inner products before the $\mathrm{Softmax}$, allowing for sharper or more uniform distribution of attention weights. The resulting cross-covariance attention is given by the weighted sum of the values V, with the weights corresponding to $A_{\mathrm{XC}}(K, Q)$

$$O_{\mathrm{XCA}} = V A_{\mathrm{XC}}(K, Q). \tag{2.10}$$

The V has linear complexity in the number of tokens and allows efficient processing of high-resolution images.

In the cross-covariance attention block (XCA), interaction between patches is made only through the shared statistics. To enable explicit communication across patches, there is a simple *local patch interaction* block (LPI) after each XCA block. This block consists of two depth-wise 3×3 convolutional layers with batch normalization and GELU non-linearity in between. LPI is a depth-wise structure; therefore, there is only negligible overhead in terms of parameters as well as a very limited overhead in terms of throughput and memory usage during inference. Finally, as is common in transformer models, there is an addition of a point-wise *feedforward neural network* (FNN). While the interaction between features is confined within the XCA block and no feature interaction takes place in the LPI block, the FFN allows for interaction across all features. Each of the XCA, LPI and FFN is preceded by *LayerNorm* and followed by a residual connection in a XCiT layer, as shown in Figure 6.

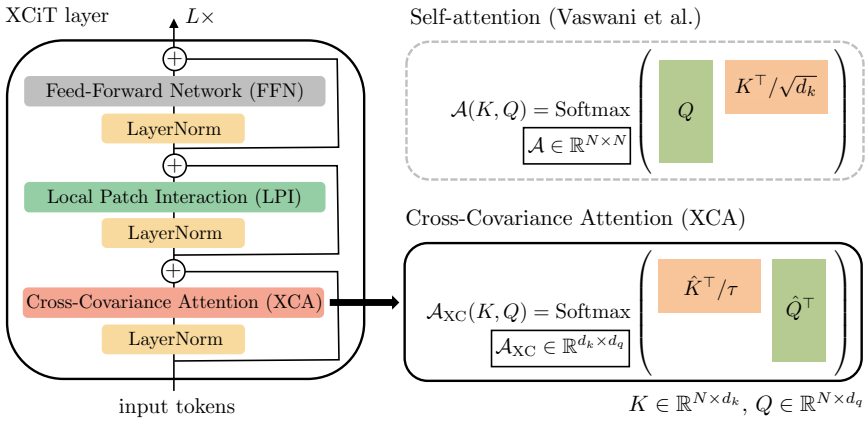


Figure 6: The XCiT layer [6].

In this work, we use XCiT small in our study on interpretability. To be more specific, we use pre-trained XCiT small models, trained in a supervised and self-supervised way, to interpret their predictions with CAM-based and attention-based visualization methods. The self-supervision of XCiT small is made with DINO [8] approach.

2.1.4 Self-supervised learning

The goal of deep learning itself is to learn good representations of raw data. Unsupervised learning is concerned with learning these representations without labels. Self-supervised is often used interchangeably with unsupervised learning and ”self” usually refers to the scenario where we can create our own supervision based on the data. It becomes crystal clear that learning good representations in an unsupervised way can be beneficial, considering the expenses and difficulty of producing new datasets for new tasks. Taking advantage of the vast amount of unlabeled data on the Internet, is something supervised learning cannot do. So no matter if one can appreciate the success of self-supervised

learning, there is still a lot more unlabeled data outside than labelled and it could be nice to leverage the power of the unlabeled data to further improve the performance of supervised systems. Combining supervised with self-supervised learning may result in much more cost, compute and time-efficient systems. Finally, there is also cognitive motivation. It is said that the way self-supervised models learn from data, can be seen as the way babies or animals learn from the outside environment. Maybe learning from data is the only way for the models to obtain ”real” intelligence. A relevant quote by Pierre Sermanet was “Give a robot a label and you feed it for a second, teach a robot to label and you feed it for a lifetime.”

Depending on the pre-text task and the way we take advantage of the unlabelled data, several types of self-supervision arise. The main categories are presented below, with the related work attached.

Generative methods The objective of the *generative* self-supervised methods is to reconstruct the original input while learning meaningful latent representation. Many of these approaches rely either on *auto-encoding* of images [36–38] or on *adversarial learning* [39], jointly modelling data and representation [40–43]. These methods usually operate directly in the pixel space. The high level of detail required when generating images for representation learning is computationally expensive.

Methods based on pre-text tasks Some self-supervised methods rely on using auxiliary handcrafted prediction tasks to learn their representation. In particular, relative patch prediction [44, 45], colorizing gray-scale images [46, 47], image inpainting [48], image jigsaw puzzle [49], image super-resolution [50] and geometric transformations [51, 52] have been shown to be useful.

Contrastive methods *Contrastive approaches* [7, 53–56] have shown great promise by achieving state-of-the-art results and are the best self-supervised methods compared to the previous two categories. These models are trained by reducing the distance between representations of different augmented views of the same image (positive pairs) and increasing the distance between representations of augmented views from different images (negative pairs). These methods need a careful treatment of negative pairs [57] by either relying on large batch sizes [53, 56], memory banks [7] or customized mining strategies [58, 59] to retrieve the negative pairs. Here, the choice of image augmentations is also important for their performance [53, 56].

Methods based on self-distillation Recent works have shown that the requirement of negative pairs can be eliminated while at the same time having competitive results. Self-supervised methods based on *self-distillation* use only positive pairs and two networks for representation learning. During learning, the representations of the augmented views coming from the same image get closer and knowledge from one network is distilled into the other. Grill et al. propose BYOL [60], which is a metric-learning formulation where features are trained by matching them to representations obtained with a *momentum encoder*. Methods like BYOL can work even without the momentum encoder, but their performance will drop [60, 61]. Finally, DINO [8] takes its inspiration from BYOL but operates with a different similarity matching loss. The DINO self-supervised learning can be seen as a form of mean teacher self-distillation [62] with no labels.

MIM-based methods Masking a portion of the input tokens (words in a text or patches in an image) and training a transformer-based architecture to predict these missing tokens [9, 14–16, 63] is called *masked language modelling* (MLM) in the language domain [63] and *masked image modelling* (MIM) [9, 14, 16] in the vision domain. MIM-based methods achieve representation learning through the reconstruction of masked patches and have already shown impressive results. Depending on the *masking strategy* that is followed, several methods were proposed. In BEiT [16], images are mapped to discrete patch tokens and then a *block-wise random strategy* is introduced to mask some patches. The pre-training objective of BEiT is to recover the original patch tokens based on the corrupted image patch embeddings, which are resulted from the masking strategy. SimMIM [14] randomly masks large patches and predicts the corresponding pixels by direct regression. In iBOT [9], the self-distillation loss of DINO [8] is extended by adding an extra dense term (patch level) to the final loss, that is used for reconstructing randomly picked image patches. Finally, in [10], a masking strategy of highly-attended patches is proposed. The highly-attended patches are picked from the attention map of the [cls] token and derived from the last layer of a transformer. On the contrary, MST [17] follows an opposite masking strategy that is based on low-attended patches.

2.1.4.1 The self-supervised models used in this study

In this subsection, we take a closer look at the self-supervised models that take part on the experiments of this work.

MoCo v3 [4] *Momentum contrast* (MoCo [7]) is introduced as a way of building large and consistent dictionaries for unsupervised contrastive learning. As shown in Figure 7, MoCo uses two encoders to match an encoded query q to a dictionary of encoded keys using a contrastive loss. This approach uses the InfoNCE loss [54], which can be expressed as:

$$L_{q,k^+,\{k^-\}} = -\log \frac{\exp(qk^+/\tau)}{\exp(qk^+/\tau) + \sum_{k^-} \exp(qk^-/\tau)}, \tag{2.11}$$

where q is a query representation, k^+ is a representation of the positive (similar) key sample, $\{k^-\}$ are representations of the negative (dissimilar) key samples and τ is a temperature hyper-parameter. It is considered that there is only one positive query-key pair originated from the same image under different random crop views and augmentations. All the other keys are not originated from the same image and are considered negative pairs to the query. The dictionary keys $\{k_0, k_1, k_2, \dots\}$ are defined on-the-fly by a set of data samples. The dictionary is treated like a *queue*. Here, the current mini-batch is enqueued while the oldest mini-batch is dequeued. The keys are progressively encoded driven by a moving average *momentum update* coming from the query encoder. The encoder of the query is denoted f_q and its parameters θ_q. Similarly, f_k is the encoder of the keys and θ_k its parameters. The θ_k is updated according to

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q, \tag{2.12}$$

where $m \in [0, 1)$ is a momentum coefficient. On the other hand, the parameters θ_q are updated by back-propagation. The momentum update makes θ_k evolve more smoothly than θ_q.

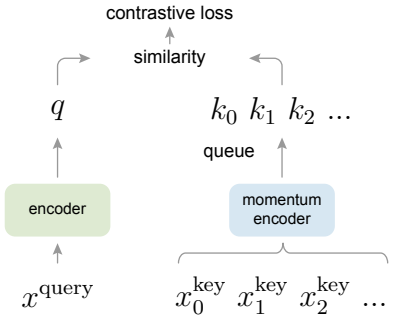


Figure 7: Momentum Contrast (MoCo) [7].

MoCo v2 [64] verifies the effectiveness of two design improvements of SimCLR [53] by implementing them into the MoCo framework. To be more specific, the fully-connected output layer that is used as the projection head in MoCo is replaced by a 2-layer MLP head. Then, the original MoCo augmentations are extended by including the blur augmentation of SimCLR. Finally, MoCo v3 [65] is an incremental improvement of MoCo v1/2 [7, 64], that achieves a better balance between simplicity, accuracy, and scalability. In this approach, the memory queue of the two previous versions is abandoned as it was found a diminishing gain if the batch was sufficiently large (e.g. 4096). While MoCo v1/2 is only tested with convolutional neural networks, MoCo v3 is generalized to ViT [4] model as well. In MoCo v3 architecture, there is a projection head, which is a 3-layer MLP, and an extra prediction head, which is a 2-layer MLP on top of the encoder f_q. The encoder f_k has the projection head only and not the prediction head. The encoder of keys f_k is updated again by the encoder of the query f_q according to (2.12), excluding the prediction head. The improvement of the 3_{rd} MoCo version is mainly a result of the extra prediction head and the large batch size. MoCo v3 further studies the *instability* issue when trained with ViT. Based on an empirical observation of gradient changes, it is found that instability happens mainly on the shallower layers. The instability challenge is faced by freezing the patch projection layer during the training process. In other words, a random patch projection layer to embed the patches is used. By this trick, the instability issue is alleviated in several scenarios and consistently the accuracy is increased.

In this work, pre-trained MoCo v3 models, that use ResNet 50 and ViT base as backbones, are used in the interpretability study. The predictions of MoCo v3 models are visually explained with CAM-based and attention-based methods.

DINO [8] Self-supervised learning in DINO can be interpreted as a form of self-distillation with no labels. This framework is flexible and works on both convolutional neural networks and transformers. A simplified version of this approach is illustrated in Figure 8. Given an input image x, two augmented views of the original image are generated x_1, x_2. The first view is given to the *student* network g_{θ_s} and the second one is given to the *teacher* network g_{θ_t}. Both networks use the same backbone architecture g with different sets of parameters, θ_s and θ_t. Only the parameters of student network θ_s are back-propagated while the parameters of the teacher θ_t are updated with an *exponential moving average* (EMA) of the θ_s parameters. The rule that updates θ_t parameters is $\theta_t \leftarrow \lambda\theta_t + (1-\lambda)\theta_s$, where λ follows a cosine schedule from 0.996 to 1 during the training procedure. The output of the teacher is *centered* with a mean calculated over the batch. Both network output a K dimensional feature that is normalized with a *temperature Softmax* over the

feature dimension. The output probability distribution from the student network is

$$P_s(x)^{(i)} = \frac{\exp(g_{\theta_s}(x)^{(i)}/\tau_s)}{\sum_{k=1}^{K} \exp(g_{\theta_s}(x)^{(k)}/\tau_s)}, \tag{2.13}$$

where $\tau_s > 0$ a temperature parameter that controls the *sharpness* of the output distribution. In a similar way, (2.13) can be used to compute P_t. The P_t and P_s are learned to be matched by minimizing a cross-entropy loss w.r.t. the parameters of the student network θ_s

$$\min_{\theta_s} H(P_t(x), P_s(x)), \tag{2.14}$$

where $H(a, b) = -a \log b$. The loss is applied globally between the [cls] tokens of a,b.

DINO follows a *multi-crop* strategy [66], where from a given image x a set V of different views is generated. The set consists of two *global* views, x_1^g and x_2^g and several *local* views of lower resolution. When taking into account all the generated views, Equation 2.14 is generalized to

$$\min_{\theta_s} \sum_{x \in \{x_1^g, x_2^g\}} \sum_{\substack{x' \in V \\ x' \neq x}} H(P_t(x), P_s(x')). \tag{2.15}$$

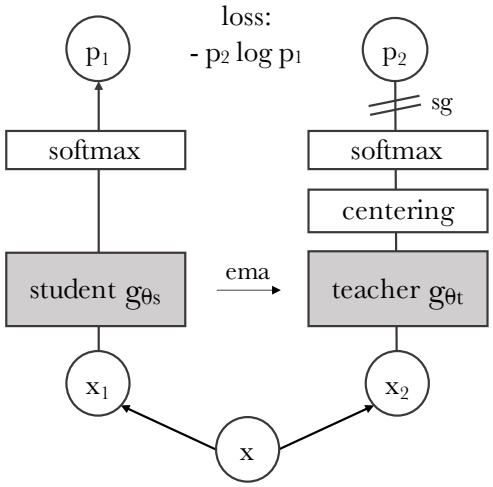


Figure 8: DINO self-distillation based approach [8]

.

In this work, pre-trained DINO models, that use ResNet 50, ViT base and XCiT small as backbones, take part in the interpretability study. The predictions of DINO models are visually explained with CAM-based and attention-based methods.

iBOT [9] This self-supervised framework performs *masked image modelling* (MIM) via self-distillation. Self-distillation is shown previously in DINO [8], in which the distillation loss is applied globally on the [cls] token. iBOT turns this task into masked image modelling by applying an extra loss densely on the masked tokens.

iBOT follows the *random block-wise strategy* of BEiT [16], in which a mask vector $m = (m_1, \ldots, m_n) \in \{0, 1\}^n$ is generated to randomly mask blocks of image patch tokens. An

input image X is tokenized as $Z = (z^{[\text{cls}]}; z_1; \ldots; z_n)$. The mask vector m generated from BEiT strategy, gives the masked tokenized image $\widetilde{Z} = (\tilde{z}^{[\text{cls}]}; \tilde{z}_1; \ldots; \tilde{z}_n)$, with

$$\tilde{z}_i = (1 - m_i)z_i + m_i z^{[\text{mask}]}, \tag{2.16}$$

for $i = 1, \ldots, n$, where $z^{[\text{mask}]} \in \mathbb{R}^d$ is the "mask" token. This special token is learnable as the [cls].

A simplified version of iBOT learning procedure is illustrated in Figure 9. The u and v are two *tokenized image* views of an image X and $\hat{u}$, $\hat{v}$ the respective masked versions of the views. The student network outputs for the masked view $\hat{u}$ the embedding of its patch tokens $\hat{u}_s^{\text{patch}} = P_{\boldsymbol{\theta}_s}^{\text{patch}}(\hat{u})$ and the teacher network outputs for the non-masked view u the embedding of its patch tokens $u_t^{\text{patch}} = P_{\boldsymbol{\theta}_t}^{\text{patch}}(u)$. The training objective of MIM is a cross-entropy loss and is applied densely between masked and unmasked output patch embeddings of an image. The loss for a specific view u is

$$L_{\text{MIMview}} = -\sum_{i=1}^{n} m_i \cdot P_{\theta_t}^{\text{patch}}(u_i)^\top \log P_{\boldsymbol{\theta}_s}^{\text{patch}}(\hat{u}_i), \tag{2.17}$$

where n is the number of patch tokens of the given image and m the mask vector. The second loss term of iBOT is the distillation loss of DINO. As shown in Figure 9, u is passed through the teacher and $\hat{v}$ through the student network to get the predictive categorical distributions from the [cls] token: $u_t^{[\text{cls}]} = P_{\theta_t}^{[\text{cls}]}(u)$ and $\hat{v}_s^{[\text{cls}]} = P_{\theta_s}^{[\text{cls}]}(\hat{v})$. Here, a cross-entropy loss is applied globally on the predictive categorical distributions and is formulated as:

$$L_{\text{glob}} = -P_{\theta_t}^{[\text{cls}]}(u)^\top \log P_{\theta_s}^{[\text{cls}]}(\hat{v}). \tag{2.18}$$

iBOT follows the multi-crop strategy of DINO. Based on this strategy, (2.18) and (2.17) must be generalized to take into account all the crops that are generated from the original image. From a given image X, a set of different views $\in V$ is generated. There are two global views $V_g \subseteq V$ and several local views $V_l \subseteq V$ of smaller resolution, where $V_g \nsubseteq V_l$. Through the teacher network, only the unmasked global views are passed. On the other hand, masked global and unmasked local views are given to the student network. The training objective of MIM considers only global views and when taking into account all of them (2.17) can be reformulated to:

$$L_{\text{MIM}} = -\sum_{v \in V_g} \sum_{i=1}^{n} m_i^v f_{\theta_t}(Z^v)_i \log(f_{\theta_s}(\widetilde{Z}^v)_i), \tag{2.19}$$

where Z^v is the tokenized unmasked image of a view $v \in V_g$ and $\widetilde{Z}^v$ its tokenized masked image. The $f_{\theta_t}(Z^v)$ and $f_{\theta_s}(\widetilde{Z}^v)$ are the output patch embeddings of teacher and student network respectively. With a similar way, 2.18 can be generalized and used for all the views of an image. This loss takes into account the global views passed through the teacher and both global and local views passed through the student. The cross-entropy loss is applied globally on the output [cls] embeddings of the teacher f_{θ_t} and student f_{θ_s} networks

$$L_{\text{CLS}} = -\sum_{v \in V_g} \sum_{j \in V} \mathbb{1}_{v \neq j} f_{\theta_t}(Z^v)^{[\text{cls}]} \log(f_{\theta_s}(Y^j)^{[\text{cls}]}), \tag{2.20}$$

where Z^v is a *tokenized unmasked image* of a specific global view $v \in V_g$, Y^j a tokenized image (masked or not) of a global or local view $j \in V$, $f_{\theta_t}(Z^v)^{[\text{cls}]}$ is the output [cls] embedding of the teacher and $f_{\theta_s}(Y^j)^{[\text{cls}]}$ the respective output of the student. The overall iBOT loss is a weighted sum of (2.19) and (2.20). In the learning process of iBOT framework, as in DINO, only the parameters of the student θ_s are back-propagated while the parameters of the teacher θ_s are updated with an exponential moving average (EMA) of the parameters of the student.

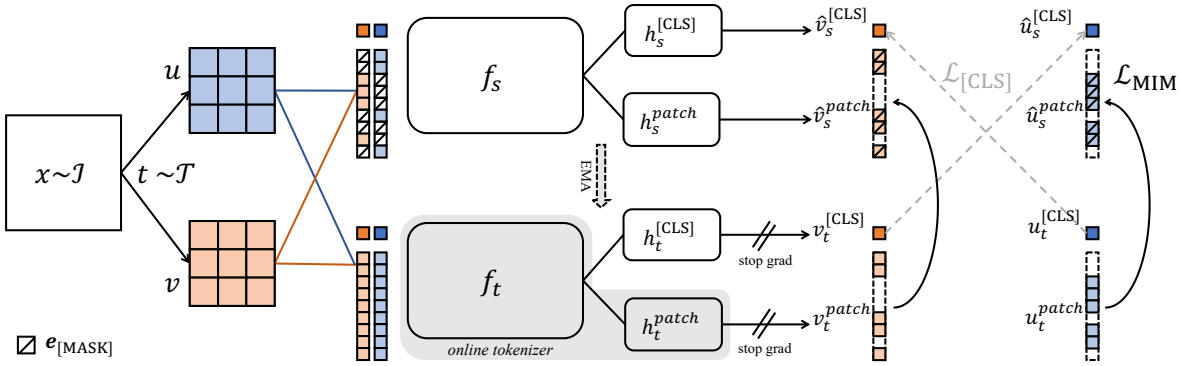


Figure 9: iBOT framework [9]

In this work, we use a pre-trained iBOT model, which has a ViT base backbone architecture, in the interpretability study. We interpret the model with attention-based methods. We further train from scratch iBOT models, that use a ViT-S/16 model as a backbone, in the self-supervised study.

AttMask [10] *Attention-guided masked image modeling* (AttMask) is again an idea in the context of distillation-based MIM. Here, the mask generation is based on the attention maps of a teacher network and it is shown that this type of informed masking demonstrates improvements over the previous baseline random masking strategies [14–16] and creates a more challenging MIM pre-text task for self-supervision.

In AttMask [10], both teacher and student use the same transformer as a backbone and the basic idea is that the teacher encoder generates an attention map which is used to guide masking for the student encoder. A simplified overview of this self-attention mechanism is illustrated in Figure 10, where a tokenized image Z is given as an input to the teacher network $f_{\theta'}$ and this encoder outputs the target features $f_{\theta'}(Z)$ and the attention map of the last layer $\overline{a}^{[\text{cls}]}$. The attention map definition is described in detail in subsection 2.2.4 and is expressed by (2.32). Based on the attention map, a mask m^H, based on the most high-attended tokens, is generated. Once, the high-attended token mask is generated per image, the masked tokenized image $\widetilde{Z}$ is created according to (2.16) and passed through a student encoder f_θ to generate the predicted features $f_\theta(\widetilde{Z})$. The multi-crop strategy is followed in this self-supervised approach as well. Learning is succeed by minimizing the dense distillation loss L_{MIM} (2.19), which is applied between the predicted projections of patch tokens and target non-masked patch tokens projections. As in iBOT, AttMask also uses the global cross-entropy loss $L_{[\text{CLS}]}$ (2.20), applied between the teacher and student [cls] output representations. Therefore, the final loss of AttMask is a weighted sum of both (2.19), (2.20). Following [8, 9], only the parameters of the student θ are back-propagated and the parameters of the teacher θ' are updated with an exponential moving average of the parameters of the student.

In this work, we train from scratch AttMask models, that use a ViT-S/16 model as a back-

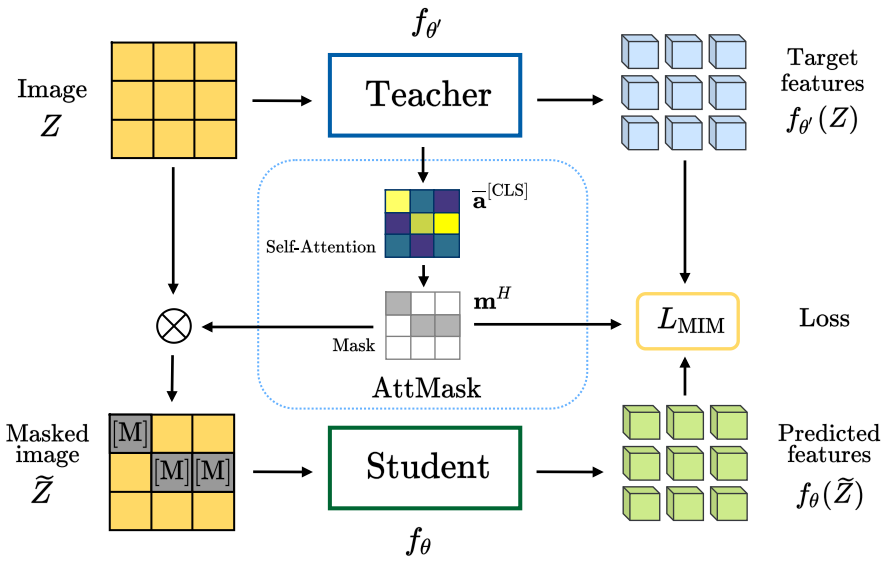


Figure 10: Attention-Guided Masked Image Modeling approach [10]

bone, in the self-supervised study. Based on this approach, we propose new masking strategies and define a more challenging MIM-based self-supervision pre-text task.

2.2 Interpretability of DNN

The basic purpose of this section is to highlight the importance of neural network interpretability, refer to its basic categories and finally present the baseline interpretability methods used in this project. But before proceeding to the next sections, we will try to define interpretability. The clearest and to-the-point definition for us is given by [67], where interpretability is defined as the ability to provide explanations in understandable terms to a human. Explanations can be logical rules and understandable terms should be from the domain knowledge related to the task. In other words, interpretability is about providing explanations that are built on top of understandable terms which can be specific to the targeted tasks.

2.2.1 Intepretability importance

Undoubtedly, the interpretability issue affects people's trust in deep learning systems and lack of interpretability could be harmful in many cases [67–69]. Having this in mind and based on [70], we summarize three essential reasons why interpretability is important.

1. *High-reliability requirement:* Although deep neural networks have proven so far great performance on some relatively large test sets, the real-world environment is way more complex. As some unexpected failures are inevitable, there is a crucial need for means to ensure control. Having in mind that some prediction systems are required to be highly reliable because an error may cause catastrophic results (e.g. human lives, heavy financial loss), interpretability can make potential failures easier to detect avoiding severe consequences.
2. *Ethical and legal requirements:* It is a fact, that there is a worry about fairness when deep neural networks are used in our daily routine and that seems to be normal because a neural network may be trained with a biased training set, which is often

not so easy to observe. So a way to examine the fairness of a network prediction is to interpret it. In addition, many companies in the industry, such as banking and insurance, use data from their clients and machine learning models for automated decision-making. It is a legal requirement for these companies to be able to explain to their clients how a decision was taken. Interpreting the predictions of models maybe is the only way to achieve this.

3. *Scientific usage:* According to Thomas Hobbes, “Science is the knowledge of consequences, and dependence of one fact upon another.” When a deep neural network reaches a better performance over its previous baseline model, it must have found some unknown ”knowledge”. Interpretability is a way to reveal it.

2.2.2 Interpretability categories

The more the need for interpreting models, the more methods are released. There have been proposed several ways of categorizing these methods. We believe that a novel taxonomy for the existing interpretability methods is introduced in [70]. The taxonomy consist of three dimensions: *passive vs. active*, *global vs. local interpretability* and *types of explanations*.

1. *Passive vs. active approaches dimension:* The passive interpretation process is made on trained networks. This means that the weights of the model are already learned from the dataset used in training. On the contrary, in the active methods, there is a need for some changes before the start of training. The modification could be on the network architecture or the training process.
2. *From local to global interpretability:* Global interpretability includes processes used to understand the overall prediction logic of a network. Local interpretability methods are used for individual explanations of the predictions of a model. However, multiple local explanations can be accumulated to achieve a certain level of *global interpretability*.
3. *The type/format of produced explanations:* According to [70], *logic rules*, *hidden semantics*, *attribution* and explanations *by examples* are the four most common types of explanations. Extracting logic rules is a type of explanation that usually provides global explanations. In these methods, a single rule set or a decision tree is extracted by a target model. The explanation is of the form ”If an input x is classified in a class y_1, it is because the features $f_i, \ldots, f_m$ are present and features $f_o, \ldots, f_r$ are not”. In the second type of explanation, the goal is to associate abstract concepts with the activation of hidden neurons or layers. For instance, in a classification task on a dataset that consists of car images, some neurons may have a high response to the main shapes. Others may respond to the detail or background information of the cars. These methods provide global interpretability and are based on representative inputs that maximize the response of the neurons. One way to reveal the response of the neurons is the provision of visual explanation, by using appropriate tools. The format of attribution is a way to show the impact the features of an input image have on the prediction of a model. The attribution in the computer vision field is represented by a mask called a *saliency map* that indicates the high-attended regions of an image. In the last type of explanation, to interpret the predictions of a model for specific inputs, similar examples to the inputs are used to support the interpretability.

In practice, each interpretability method has each own advantages and disadvantages. In

this work, we use passive (post-hoc) attribution-based methods which could be characterized local or semi-local. To be more specific, we use popular baseline methods which are based on CAM or attention. In section 3.1, we further propose new interpretability methods, based on the two aforementioned families, which can be used as alternative or complementary visualization tools for better understanding the networks.

2.2.3 CAM-based methods

Methods based on *class activation map* (CAM) can be defined in general as simple mechanisms to interpret predictions of convolutional neural networks (CNNs). Through these methods, *saliency maps* are generated that highlight the image regions which are most relevant to a particular class. For each class, the network learns a different set of weights. After training, these weights are used to linearly combine the feature maps of a given input image for the generation of the saliency map for a specific class. In other words, the saliency map is a result of the knowledge gained for a specific class by the network during training and the feature maps of a given input image. The weights used to combine the feature maps are either based on gradients or class scores. The variation of weighting schemes results in different CAM-based methods.

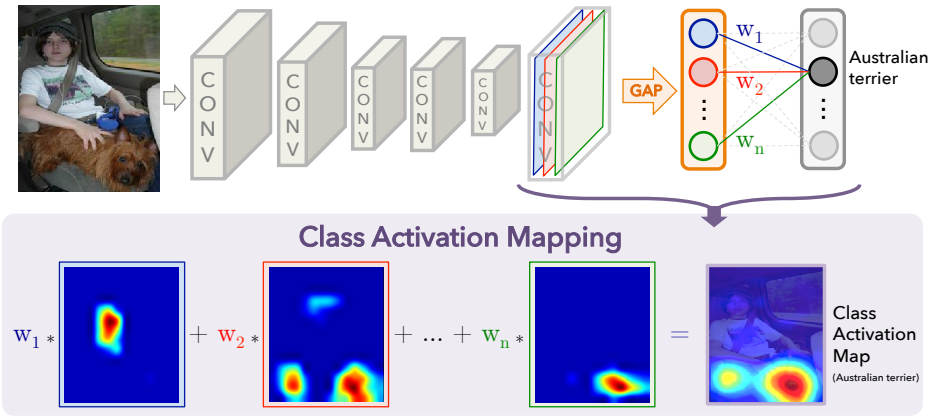


Figure 11: Class activation mapping method (CAM) [11]

.

The first CAM was introduced in [11], where it is shown that a CNN with a *global average pooling* (GAP) layer demonstrates localization capabilities, although the network was not trained for this purpose. In a network with a CNN followed by a GAP layer, the final classification score Y_c for a target class c can be expressed as a linear combination of its global average pooled feature maps A^k derived from the last convolutional layer

$$Y_c = \sum_k w_k^c \sum_x \sum_y A^k(x, y). \tag{2.21}$$

The saliency map M^c for a specific class c for each spatial location (x, y) is given by:

$$M_c(x, y) = \sum_k w_k^c A^k(x, y), \tag{2.22}$$

where w_k^c is the weight coefficient which is used to linearly combine the feature maps A^k. For a target class c, $M_c(x, y)$ is directly correlated with the importance of a specific spatial location (x, y) of an input image I and thus can be used as a visual explanation tool of the class predicted by the model. In the CAM method, a linear classifier is trained on

top of the global averaged pooled feature maps of the last convolutional layer for each target class c, for the weights w_k^c to be estimated. In other words, one linear classifier for each class needs to be trained. In Figure 11, it is shown the final saliency map derived from the CAM method for a given input image depicting an Australian terrier. Here, the global averaged pooled feature maps of the image are linearly combined by the weights $w_1, w_2, \dots w_n$, learned by the network for the specific class "Australian terrier", to produce the final saliency map.

Although the saliency maps obtained from the CAM method are class discriminative, the need for training multiple classifiers is too restrictive. To overcome the limitation of [11], new approaches, based on this idea, were proposed. Although there is a variety of CAM-based methods, we use only four of them in this study. All of them can be expressed with a common formula detailed described in the next paragraph.

Eq. (2.22) can be generalized for different selected target layers as follows. Given an input image I, a L-layer CNN, a target class c and a target layer l, the saliency map for all CAM-based methods

$$M_c(x,y) = \mathrm{ReLU}\left(\sum_{k=1}^{K} w_k^c A^{lk}(x,y)\right), \tag{2.23}$$

is expressed as linear combination of the K feature maps A^{lk}, determined by the weight coefficient w_k^c. The resulting saliency map is first *ReLU rectified* to filter negative units and then is upsampled to the size of the input image. While all the methods based on CAM use Equation 2.23 to obtain saliency maps, finding the most meaningful way to compute w_k^c and determine the importance of each feature map to the prediction of a target class results in a variety of CAM-based methods.

- In *GradCAM* [71], the weight coefficient of each feature map w_k^c for a class c is the summation of the gradients of the class score Y_c w.r.t every pixel of the feature map $A^{lk}(x,y)$

  $$w_k^c = \frac{1}{Z}\sum_{x,y}\frac{\partial Y_c(A^l)}{\partial A^{lk}(x,y)}, \tag{2.24}$$

  where Z is the total number of units in the k_{th} feature map. In other words, there is first computation of the gradient of the score for class c w.r.t feature map activations $\frac{\partial Y_c}{\partial A^{lk}}$ and then these gradients flowing back are global average pooled $\frac{1}{Z}\sum_{x,y}$ to give the importance weights w_k^c. In this approach, it is considered that all pixel gradients contribute equally to computing w_k^c, as an average from all of them is taken.

- *GradCAM++* [72] considers that pixels (x,y) that contribute more toward a class c should take more weight and not treated equally, as in GradCAM, while computing the weight coefficients w_k^c of feature maps A^{lk}. Based on this, it is introduced a pixel-wise weight coefficient $a_c^k(x,y)$ that consists of higher-order positive partial derivatives. Here, the weight coefficient of each feature map w_k^c is the summation of pixel-wise weighted ReLU rectified gradients of the class score Y_c w.r.t the feature maps $A^{lk}(x,y)$

  $$w_k^c = \sum_{x,y} a_c^k(x,y)\,\mathrm{ReLU}\left(\frac{\partial Y_c(A^l)}{\partial A^{lk}(x,y)}\right). \tag{2.25}$$

  In [72], it is shown that the addition of factor $a_c^k(x,y)$ improves robustness towards more objects on the image and the obtained saliency maps are more sharpened and better localized on images.

- In *XGradCAM* [73], it is considered, as previously, that for a class c the contributions of the pixels (x, y) is not the same; therefore, it is introduced a factor which is given by the normalized feature maps

$$s_c^k(x,y) = \frac{A^{lk}(x,y)}{\sum_{x,y} A^{lk}(x,y)}. \tag{2.26}$$

  Here, the weight coefficient of each feature map w_k^c is the summation of the weighted, by the normalized feature maps on each spatial location (x, y), gradients of the class score Y_c w.r.t the feature maps

$$w_k^c = \sum_{x,y} s_c^k(x,y) \frac{\partial Y_c(A^l)}{\partial A^{lk}(x,y)}. \tag{2.27}$$

  In [73], it is shown by using the $s_c^k(x, y)$ factor the *sensitivity* and *conservation* properties of GradCAM are boosted. Considering sensitivity, the importance of each feature map in the output probability of a model should be equal to the output change resulting from the removal of the corresponding feature of the input. In conservation, the sum of the importance of all the feature maps should match the magnitude of the output of the model.

- Finally, *Score-CAM* [74] is not dependent on gradients and obtains the weight of each feature map through its forward passing score on the target class. Again, Score-CAM can be defined for any target layer l according to (2.23) and the weight coefficient of the feature maps is

$$w_k^c = \mathrm{Softmax}\left(u^c\right)_k, \tag{2.28}$$

  where the Softmax normalization is used to consider only the positive feature contributions. This means fewer highlighted areas in saliency maps. The vector u^c computes the importance of each feature map by comparing the output probability scores of a baseline input image I and an image I_m masked according to each feature map A^{lk}

$$u_k^c = f\left(I_m \odot \mathrm{n}(\mathrm{up}(A^{lk}))\right)_c - f(I)_c, \tag{2.29}$$

  where $\odot$ is the Hadamard product which is used to mask I_m according A^{lk}, up denotes upsampling to the spatial resolution of input I_m and $\mathrm{n}(A)$ is a normalization of A into range $[0, 1]$ that is given by:

$$\mathrm{n}(A) = \frac{A - \min A}{\max A - \min A}. \tag{2.30}$$

  It is shown that Score-CAM provides the best quantitative and qualitative results over the previous CAM-based methods. Also, there is no need for gradients anymore to compute the weight coefficient w_k^c. For computing w_k^c, it is required as many forward passes through the model as the number of feature maps in the target layer, which is more computationally expensive than the three previous methods.

2.2.4 Attention-based methods

The multi-head self-attention has become the key building block in the architecture of transformers and visualizing the attention weights, alternatively called *attention maps*, is

the easiest and most popular approach to interpret the predictions of these networks and further gain insights about their internal mechanism.

To give a detailed description of how to visualize raw attention maps, we reuse and reformulate (2.6). A transformer encoder, with B total blocks, takes as an input a tokenized image $Z \in \mathbb{R}^{(n+1)\times d}$, where $n+1$ is the number of tokens, each of dimensionality d. The input Z is linearly projected to *queries* $Q = ZW_q$, *keys* $K = ZW_k$ and *values* $V = ZW_v$, using the weight matrices $W_q \in \mathbb{R}^{d\times d_q}$, $W_k \in \mathbb{R}^{d\times d_k}$ and $W_v \in \mathbb{R}^{d\times d_v}$, in h ways and will be used in h self-attention operations, called "heads", to compute independent attentions that are concatenated to produce a final attention output. The self-attention operation is applied on a small sub-space d_h of the input embedding dimension d. The d_h is computed by: $hd_h = d$, where h is the number of "heads". To compute the weights of the self-attention output of a specific block b, $Q^{(b)}$and$K^{(b)} \in \mathbb{R}^{h\times(n+1)\times d_h}$ are used

$$A^{(b)} = \mathrm{Softmax}\left(\frac{Q^{(b)}K^{(b)^\top}}{\sqrt{d_h}}\right). \tag{2.31}$$

The $\mathrm{Softmax}$ is applied, such that each element of the $A^{(b)} \in \mathbb{R}^{h\times(n+1)\times(n+1)}$ lies in the range $[0,1]$ and the sum of each row is equal to 1. A_i denotes the row i of A and represents the attention coefficients of each token w.r.t the token i. In [75], they focus on the raw attention map of the [cls] token that represents the pairwise relevancy of [cls] token and patch tokens. The attention map vector of [cls] token consists of all but the first element of the first row of A

$$\vec{a}_{\text{[cls]}} = (a_{1,2}, a_{1,3}, \ldots, a_{1,n+1}). \tag{2.32}$$

The attention map vector of [cls] token $\vec{a}_{\text{[cls]}}$ has n elements. This vector is first reshaped to 2D and then interpolated to the size of the input image. This raw attention map indicates the regions of an input image that the [cls] token attends. Visualizing the raw attention map, given an input image, is shown to be an easy and widely used method to interpret the predictions of models, based on self-attention mechanisms. In [75], the final raw attention map is derived from the last block of the model, in particular, from different heads or an average of all of them.

In [18], it is assumed that taking into consideration only attention maps from the last self-attention-block, is an unreliable way to interpret the prediction of the model because the information derived from different tokens across layers of transformers is incrementally mixed. With the main problem being how the information propagates from the input layer to the embeddings in deeper layers, it is proposed *Rollout* method in which average from all heads attention maps

$$\hat{A}^{(b)} = I + E_h A^{(b)}, \tag{2.33}$$

derived from each block b are combined linearly to produce the final attention map

$$\mathrm{rollout} = \hat{A}^{(1)}\hat{A}^{(2)}\ldots\hat{A}^{(B)}, \tag{2.34}$$

where E_h is the average across the dimension of heads and B is the last block. To account for the residual connections in transformer blocks b, in (2.33), it is added the identity matrix I to avoid self-inhibition for each token.

According [19], the main challenge in assigning attributions based on attention weights is that there is a non-linear combination of attentions from one layer to another. Based on

this, it is proposed a visualization method that *gradients* and *relevancies* are propagated through the model and integrated to generate the final attribution maps, called relevancy maps. This method adopts *LRP* to calculate the relevance scores for each attention head in each block of transformer [76] and the relevance propagation is based on the generic *deep Taylor decomposition* [77]. In this study, we call this method *TIBAV*. Given an input image, the final raw attention and Rollout maps are independent of a target class. On the other hand, TIBAV is the only class-specific visualization tool for transformers. We briefly describe this approach with the following two equations:

$$\bar{A}^{(b)} = I + E_h(\nabla A^{(b)} \odot R^{(n_b)})^+, \tag{2.35}$$

$$A_{\text{final}} = \bar{A}^{(1)}\bar{A}^{(2)}\dots\bar{A}^{(B)}. \tag{2.36}$$

Following the propagation process of gradients and relevance, each attention map $A^{(b)}$ of a specific block b has its gradients $\nabla A^{(b)}$ and relevance $R^{(n_b)}$ w.r.t a target class c. Here, n_b is the layer which corresponds to the Softmax normalization operation in (2.31) of block b and $R^{(n_b)}$ is the relevance of the layer. E_h is again the average across the heads dimension, I is the identity matrix to account for the residual connections in the model and $\odot$ is the Hadamard product. To compute the weighted attention relevance, only the positive values of the multiplication between gradients and relevance are considered $(^+)$. The final map of the method $A_{\text{final}} \in \mathbb{R}^{(n+1)\times(n+1)}$ is a result of multiplying the maps $\bar{A}^{(b)}$ from each block b.

3. INTERPRETABILITY

The whole section is dedicated to the presentation of our work in the interpretability domain. We first describe in detail the methodology we follow in our experiments and present the most important contributions of our work. We give then a general overview of the dataset, networks and the evaluation protocol we use. Here, we add some useful implementation details. The chapter ends with the presentation of the experimental results and the conclusions.

3.1 Methodology

The methodology of our work in this chapter is correlated with the interpretability of pretrained deep neural networks. Here, we interpret the predictions of both supervised and self-supervised models that are based on common backbone architectures. The networks we choose to investigate are state-of-the-art convolutional neural networks and transformers. Given a series of input images, we interpret the models by using variation of post-hoc visualization techniques to provide visual explanations. These explanations are known as saliency maps and are not evaluated only qualitatively but quantitatively as well. The evaluation of the saliency maps provides us with some significant observations on the post-hoc methods and the evaluation metrics we use. These saliency maps further help us to explore the inner mechanism of the models and give us the opportunity to compare their predictions. A more thorough description of the methodology we follow in this set of experiments is organized and presented below.

Interpretability of models with CAM-based methods We conduct extensive experiments using a variety of CAM-based methods to interpret both CNNs and transformers, trained in a supervised or self-supervised way. The visualization methods we use here are GradCAM [71], GradCAM++ [72], XGradCAM [73] and Score-CAM [74]. Although the aforementioned methods are proposed to interpret CNNs architectures, we use them for the transformers as well, following the implementation code [78]. As we've already mentioned in subsection 2.2.3, a saliency map for a specific class c of an input image for all CAM methods could be expressed as a linear combination of the feature maps in the target layer. In a transformer, the output of the layers is typical of shape $b \times (n+1) \times d$, where b is the batch size. In the $n+1$ dimension, the first element represents the [cls] token and the rest represent the $p \times p$ patches in the image. We treat the last n elements as a $p \times p$ spatial image, with d channels. This reformation gives the output of transformer the shape of feature maps learned by CNNs. By reshaping the output of the transformer along with the gradients, it is possible to use CAM-based methods for transformers as well.

The saliency maps we observe from all CAM-based methods are evaluated qualitatively and quantitatively. For the quantitative evaluation, we use classification metrics which are described in detail in subsection 3.3.3. All the qualitative and quantitative results along with their discussion can be found in subsection 3.4.1.

Interpretability of models with baseline attention-based methods Here, we want complementary attribution maps for transformers, obtained from more standard visualization approaches for these networks. Therefore, we use attention-based methods which

can be used to interpret only transformers and not CNNs. To be more specific, we visualize raw attention maps and use the Rollout [18] and TIBAV [19] methods to further interpret supervised and self-supervised transformers.

All the attribution maps, obtained from the aforementioned methods, are evaluated qualitatively and quantitatively with classification metrics. We summarize the results and our observations in subsection 3.4.2.

A closer study on the predictions of transformers At this set of experiments, we take a closer look at the inner mechanism of transformers and further investigate the way these models predict, given input images, by proposing new complementary visualization methods that are based on the previous attention-based approaches. The obtained attribution maps along with their evaluation are found in subsection 3.4.2.

- *Raw attention maps:* As previously mentioned in subsection 2.2.4 visualizing the raw attention map of the [cls] token is the easiest and most popular approach to interpret a prediction of a transformer. The most standard approach so far is to visualize the raw attention map from the last layer of a transformer, which is a result of averaging the attention map from each head for a specific layer. For abbreviations purposes, one can find this approach in our work as *Raw*$_{12}$ or simply *Raw*. Here, we study also the option where the $\mathrm{Softmax}$ operation is absent from the self-attention mechanism and we call this method *Raw**. We also choose to visualize attention maps from shallower layers as well, *Raw*$_8$,*Raw*$_5$,... etc. This will give us a better idea of where both supervised and self-supervised models focus on an image when going deeper into the networks and vice versa. For the attention map obtained from the last layer, we further investigate the difference in the maps when studying each head individually.
- *Rollout complementary methods:* Rollout [18] as we mention in subsection 2.2.4 is a linear combination of the attention maps derived from all the layers of transformer. Based on this, we use the method for specific groups of layers and not for all. So for instance, if we want to investigate the predictions of the model in a group of middle layers, we linearly combine their attention maps and evaluate the resulting map. We call *Rollout*$_{3-5}$ the method from which the final attention map is derived from the $3_{rd}, 4_{th}$ and 5_{th} layer etc.
- *Pre-processing raw attention maps:* Here, we use two mathematical functions to pre-process Raw to reveal the next highest attended regions on an image. The first function we use is the $\log$. As we said, the final attention map is derived from the average of the attention maps of all heads in a specific layer. Now, we first apply the $\log$ function on the attention map of each head and then we calculate the average. The $\log$ function brings all the attention values closer to the most highly attended ones. By this way, we reveal the next more important regions on an input image that a model takes into account before its prediction. The name we give in this method is *Raw-Log*. A similar influence on the attention map has the power function when the map of each head is raised to powers $\in (0, 1)$. The smaller the value of the power, the more "hidden" regions are revealed, always following the priority of the most important pixels. The method when using the power value 0.5 has name *Raw-Pow*(0.5).
- *Taking advantage of the tensor of keys:* As we showed in subsection 2.2.4, the multi-head self-attention module operates on a tokenized image $Z \in \mathbb{R}^{(n+1)\times d}$, where $n + 1$ is the number of tokens, each of dimensionality d. The input Z is linearly pro-

jected to queries Q, keys K and values V all $\in \mathbb{R}^{h\times(n+1)\times d_h}$, where h is the number of heads and $d_h = d/h$. Then, Q and K values are used for the attention weights $A \in \mathbb{R}^{h\times(n+1)\times(n+1)}$ calculation according (2.31). So far, all the attention-based methods that are independent of a certain class take advantage only of A to provide visual explanations. As transformers are complicated architectures and information is shared across multiple linear projections, we strongly believe that valuable information for interpreting the prediction of the models is not hidden only in A. Therefore, we take advantage of K and treat it in a similar way we treat A so far to provide visual explanations. Here, it must be mentioned that we use no $\mathrm{Softmax}$ operation on K. To be more specific, first, we estimate the average keys tensor over all heads $\bar{K} \in \mathbb{R}^{(n+1)\times d_h}$ for the last block of the network and then we calculate its $L2$ *norm* over d_h, $\|\bar{K}^{(b)}\| \in \mathbb{R}^{n+1}$. From $\|\bar{K}^{(b)}\|$, we exclude the first element and we take

$$\vec{k} = (k_{1,2}, k_{1,3}, \ldots, k_{1,n+1}). \tag{3.1}$$

The $\vec{k}$ consists of n elements and is first reshaped to $p\times p$, which is the patch grid size, and then upsampled back to the size of the original image using bilinear interpolation. This will be the final attribution map we use to reveal some hidden information inside K and as we show, in subsection 3.4.2, it can be used for interpretability purposes. Later, we call this method ***Keys***$_{\mathrm{norm}}$.
Here, we propose a second method where we obtain attribution maps derived simply from the pairwise multiplication of (2.32), (3.1). The final vector obtained from $\vec{a}_{\text{[cls]}} \cdot \vec{k}$ has n elements. As before, it is reshaped to the patch grid size and then upsampled back to the size of the input image. Here, it must be clarified that there is no $\mathrm{Softmax}$ operation on $\vec{a}_{\text{[cls]}}$ in this method. In the experiments, we call this approach ***Raw***$^*\times$ ***Keys***$_{\mathrm{norm}}$.

3.2 Contributions

In this section, we highlight the most important contributions of the methodology we follow in section 3.1.

1. We provide you with an extensive study on visual explanations of supervised and self-supervised learning. It is the first study that uses more than 14 different post-hoc attribution-based methods to interpret 10 deep neural networks that are based on 4 different backbone architectures that are either CNNs or transformers. Of the 14 visualization methods, 7 are new and proposed by us to further understand the predictions of transformers. For all methods and models, there are both qualitative and quantitative results.
2. From the experimental results for both CNNs and transformers, we observe important insights about supervised and self-supervised models when sharing the same backbone. Furthermore, we determine what information the models need to feel confident for their predictions. We identify also which of the selected post-hoc methods are proper visualization tools for CNNs and which are for transformers and show if there is a tool that can be used for both network families.
3. From the experiments we conduct on transformers, we show where important information can be hidden inside the network and how to reveal it thanks to the new visualization methods we propose. By using the new visualization methods, we further obtain better classification metrics than previous post-hoc attribution-based methods.

3.3 Experimental setup

This section is dedicated to experimental setup we follow in chapter 3. There is a detailed report about the dataset, networks and evaluation protocol we use. We further provide some implementation details of the experiments we conduct.

3.3.1 Dataset

The dataset we use for the experiments of this chapter is a subset of the validation set of ImageNet [12], which is briefly described in the next paragraph. To be more specific, from the total 50k images of the validation set, we use only 1000, 1 random image per class. The data was downloaded from:

`https://github.com/EliSchwartz/imagenet-sample-images`

ImageNet ImageNet [12] is a database with over 14 million varying-resolution images. This dataset was one of the first of its kind regarding its scale. *ImageNet large scale visual recognition challenge* (ILSVRC) was an annual computer vision competition that took place between 2010 and 2017. For this challenge, the training data, which is a subset of ImageNet, consists of 1.2 million images that are shared in 1000 classes. The validation set consists of 50k images and the testing set is comprised of 150k. The ILSVRC contest included three tasks. The first was the image classification task and since 2011 there has also been a single-object localization task. Since 2013, there has been an object detection task as well.

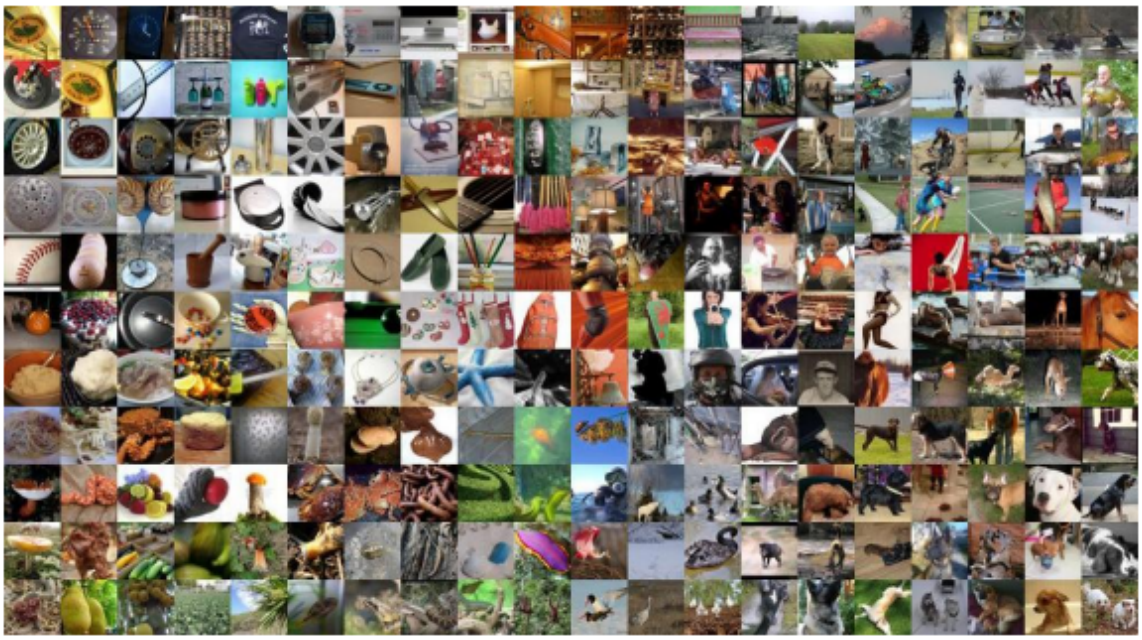


Figure 12: Random image samples derived from ImageNet [12].

3.3.2 Networks

In this chapter, we want to interpret the predictions of deep neural networks that are pre-trained on ImageNet [12]. We conduct experiments on both supervised and self-supervised networks that use as backbone architectures ResNet, ViT, DeiT and XCiT. The exact models we use along with the source where we found the weights are all summarized in Table 1.

Table 1: Pre-trained networks used in chapter 3.

Backbone	Method	Weights
ResNet 50	Supervised	`https://pytorch.org/vision/stable/models.html`
	DINO	`https://github.com/facebookresearch/dino`
	MoCo v3	`https://github.com/facebookresearch/moco-v3/blob/main/CONFIG.md`
ViT base	Supervised	`https://timm.fast.ai/`
	DINO	`https://github.com/facebookresearch/dino`
	iBOT	`https://github.com/bytedance/ibot`
DeiT base	Supervised	`https://github.com/facebookresearch/deit`
	MoCo v3	`https://github.com/facebookresearch/moco-v3/blob/main/CONFIG.md`
XCiT small	Supervised	`https://github.com/facebookresearch/xcit`
	DINO	`https://github.com/facebookresearch/dino`

3.3.3 Evaluation protocol

Here, we first perform a visual evaluation of the saliency maps, obtained from each model and visualization method, given an indicative number of input images. We evaluate then quantitatively the saliency maps of 1000 ImageNet samples derived from the validation set with the following classification metrics:

- *Average drop [72]:*A good saliency map, for a specific given input image and class, should highlight the regions that are most relevant for the prediction of model. This metric is computed as the average drop (AD) in the confidence of model for a particular class c in an image when giving to a model only the saliency map regions

$$AD = \sum_{i=1}^{N} \frac{\max(0, Y_i^c - O_i^c)}{Y_i^c}. \tag{3.2}$$

  Y_i^c is the confidence score of model for class c given the i^{th} image and O_i^c is the output score of model for class c when only the saliency map regions are given as input. The $\max$, in the numerator, is used to handle cases where $O_i^c > Y_i^c$. The average drop is calculated per image and then averaged over the full dataset. We expect law average drop values for good saliency maps.
- *Average increase [72]:* Average increase (AI) measures the number of times in the full dataset, the confidence of model increased when providing only the saliency map regions as input

$$AI = \sum_{i=1}^{N} \frac{\mathbb{1}_{Y_i^c < O_i^c}}{N}, \tag{3.3}$$

  where N is the number of input samples, $\mathbb{1}$ an indicator function that returns 1 when the argument is true, Y_i^c the confidence score for the original image and O_i^c the score when providing only the saliency map regions as input to the model. The higher the average increase scores the better the saliency map.

- *Deletion [13]:*Deletion metric (D) measures a decrease in the confidence score of the predicted class as pixels are removed progressively according to their descending importance. The importance is obtained from the saliency map of an image. As shown in Figure 13 for a good saliency map, we expect a sharp drop and consequently a low area under the probability curve. The deletion score is determined by the *area under the curve* and given by:

$$AUC = \frac{Y_0^c/2 + Y_s^c/2 + \sum_{i=0}^{s} Y_i^c}{s}, \tag{3.4}$$

  where Y_0^c the probability score before adding or removing important pixels, Y_s^c the score in the last step, Y_i^c the score in the i_{th} step and s the total number of steps.
- *Insertion [13]:*The insertion metric, takes the complementary approach. It measures the increase in the confidence score of model as more and more pixels are introduced. Again, first, the most important pixels are added. Here, we expect a big area under the probability curve Figure 13 and high AUC values according to (3.4) for good saliency maps.

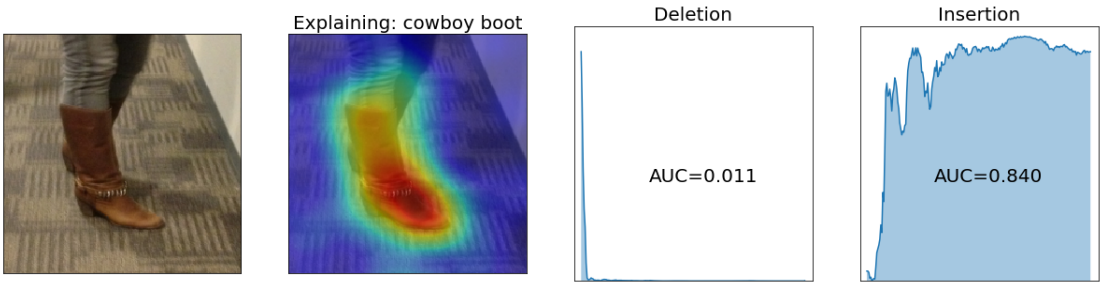


Figure 13: The deletion and insertion metrics for a random input image [13].

3.3.4 Implementation details

The baseline code for this set of experiments is [78]. It is used for CAM-based method experiments and is further developed to include attention-based approaches as well:

`https://github.com/DimitrisReppas/On_visual_explanation_of_supervised_and_self-supervised_learning`

When we conduct CAM-based experiments we have to choose a *target layer* from which we want the saliency map to be generated. For networks based on ResNet 50 backbone, we choose the last convolutional layer by default, i.e., convolutional layer 3 of bottleneck 2 of block 4. For networks that use transformers as backbones, we choose from their last block the norm1 layer, following [78] strategy.

As far as it concerns the image *pre-processing* and *normalization*, we use the same techniques, for a fair comparison, in all models and visualization methods. To be more specific, we resize all images to 224×224. To normalize them, we first divide them by 255 and then we subtract the mean ImageNet vector $[0.485, 0.456, 0.406]$ and divide channel-wise by the standard deviation of ImageNet $[0.229, 0.224, 0.225]$

To obtain saliency maps with TIBAV [19] visualization method, we use its official released code. For a fair comparison with the other methods, we use again the same image preprocessing and normalization techniques.

3.4 Experimental results and discussion

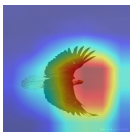
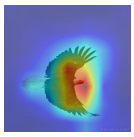
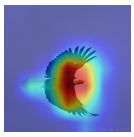
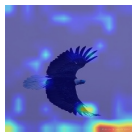
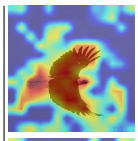
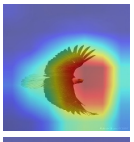
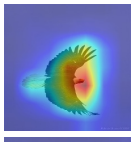
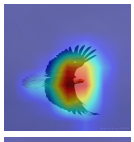
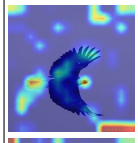
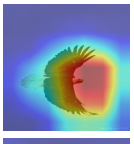
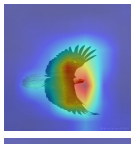
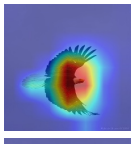
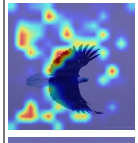
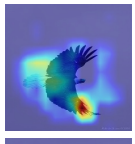
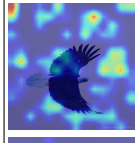
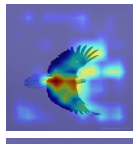
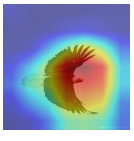
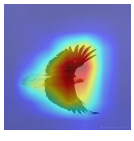
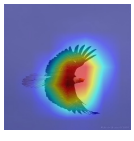
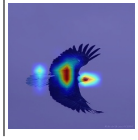
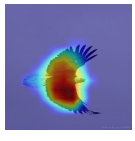
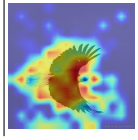
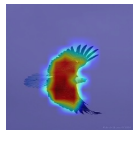
This section is dedicated to experimental results and is organized as follows. We first present qualitative and quantitative results of saliency maps derived from both CNNs and transformers when using CAM-based methods. Then, we focus only on transformers and present qualitative and quantitative results of their saliency maps, when using standard attention-based methods and alternative approaches proposed by us. All the results are discussed in detail.

3.4.1 Interpretability of models with CAM-based methods

In this experiment, we use GradCAM [71], GradCAM++ [72], XGradCAM [73] and Score-CAM [74] to interpret CNNs and transformers, trained in a supervised or self-supervised way. The obtained saliency maps are visually evaluated for several random samples derived from the ImageNet validation set. All visualization methods are quantitatively evaluated on the 1000 images validation subset with the average decrease (AD), average increase (AI), deletion (D) and insertion (I) metrics.

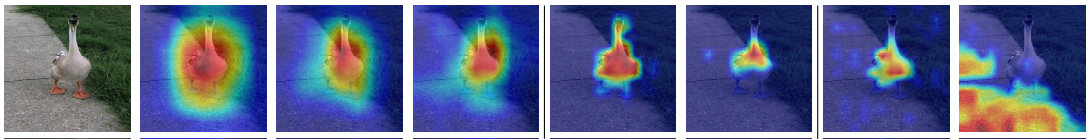
In Table 2, we show the saliency maps for a random input image, from both CNNs and transformers for the aforementioned CAM-based approaches. For networks using ResNet 50 as a backbone, a change of the visualization method does not cause a serious variation on the saliency map, although quantitatively Score-CAM [74] has the best performance at least at AD, AI metrics, as shown in Table 5. The rest visualization methods have similar metrics for these models. On the other hand, the influence of the chosen visualization method on the saliency map is great when the network has a transformer as a backbone. This is an insight that interpreting transformers with the CAM-based method is not the best option, taking into account their quantitative results in Table 5 as well. Although it is hard to say, from Table 2 and Table 5 we observe that the two best methods for transformers are GradCAM [71] and Score-CAM [74] while the other two are equally bad.

Table 2: Saliency maps for a given input image, obtained from different deep neural networks, when using four different CAM-based methods.

Method	Input image	ResNet 50			XCiT small		DeiT base	
		Supervised	DINO	MoCo v3	Supervised	DINO	Supervised	MoCo v3
GradCAM [71]								
GradCAM++ [72]	bald eagle							
XGradCAM [73]								
Score-CAM [74]								

In Table 3, we show saliency maps derived from Score-CAM [74], given several input images to the previous models. We visualize them to show if there is a standard effect on the maps when going from supervised to self-supervised learning. First and foremost, when looking at the CNNs we observe reasonable maps for both supervised and self-supervised networks. From Table 2, Table 3 and Table 5, we can clearly say that the CAM-based methods are great interpretability methods for networks that use CNNs as backbones. By looking closer to Table 3, we observe that the saliency maps derived from self-supervised CNN networks are smaller, with less background information than the maps obtained from the respective supervised models, although all are equally well localized. This is important because we show that self-supervised models learned to predict by mostly paying attention to the object and not to the background. This can be explained quantitatively as well, from Table 5. The metrics we choose are classification ones and are closely related to the confidence scores of the network when seeing only the salient regions of an image. When passing through a network equally good saliency regions, the ones that contain information from the background other than information from the object will help most of the time the network to be more confident in its prediction. This is the reason, supervised models have better quantitative results, at least in AD and AI metrics. On the other hand, the qualitative analysis of transformer-based models, in Table 3, does not show us that the type of learning has a clear impact on the saliency map. For some specific inputs, all methods provide us with meaningful maps and for others we observe good maps coming only from the supervised or only from the self-supervised models. The previous qualitative observations are expressed in the same unclear way quantitatively, in Table 5. Once again, this means that CAM-based methods do not work very well with transformer-based models.

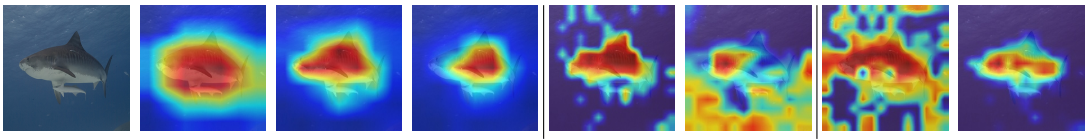
Table 3: Saliency maps for several input images, obtained from different deep neural networks, when using Score-CAM method.

Target class	Sample	ResNet 50			XCiT small		DeiT base	
		Supervised	DINO	MoCo v3	Supervised	DINO	Supervised	MoCo v3
goose								
dugong								
meerkat								
envelope								
megalith								

In Table 4, we present a qualitative and quantitative evaluation of saliency maps derived from GradCAM [71], given a random input image to the previous networks. Here, we

see that only when all the metrics are good the saliency map is acceptable. For instance, saliency maps derived from the networks based on ResNet and the DeiT model pre-trained with MoCo v3 self-supervised approach provide good quantitative and qualitative results. Additionally, the saliency map obtained from the XCiT model pre-trained with the DINO approach only gives a great AD score, but the rest are not impressive; thus, the qualitative results are also not reasonable. This means that the metrics are not good representatives of the saliency map when used separately. So the more metrics we use, the better the interpretability of the models. It is noteworthy that the aforementioned insight is observed only for a small number of samples whose saliency maps have been assessed qualitatively and quantitatively. It may not be the case for the entire dataset.

Table 4: Qualitative and quantitative evaluation of saliency maps obtained for a given input image. The maps are derived from different deep neural networks, when using GradCAM method. AD/AI: average drop/increase I/D: insertion/deletion ↓ / ↑: lower / higher is better. For one input image, AI returns either 1 (true) or 0 (false).

Metric	Input image	ResNet 50			XCiT small		DeiT base	
		Supervised	DINO	MoCo v3	Supervised	DINO	Supervised	MoCo v3
								
AD↓	tiger shark	**0.02**	0.11	0.10	0.39	0.05	0.20	0.24
AI↑		0	0	0	0	0	0	0
D↓		0.14	0.17	0.14	**0.09**	0.49	0.14	0.19
I↑		**0.94**	0.86	0.84	0.41	0.74	0.41	0.88

In summary, in these experiments, we show meaningful saliency maps and comparisons of supervised and self-supervised models that use ResNet as a backbone. The insights hold for four different CAM-based methods. In contrast, these methods do not allow us to reveal important observations about transformer-based models. This is why in the next set of experiments we focus on alternative visualization methods to interpret supervised and self-supervised transformers.

Table 5: Quantitative evaluation of CAM-based saliency maps. Classification metrics on 1000 randomly picked images from the ImageNet validation set. AD/AI: average drop/increase I/D: insertion/deletion ↓ / ↑: lower / higher is better.

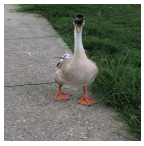
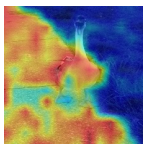
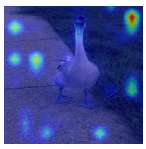
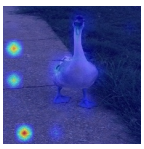
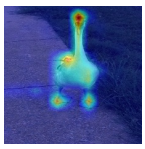
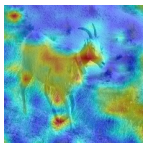
Metric	Method	ResNet 50			XCiT small		DeiT base	
		Supervised	DINO	MoCo v3	Supervised	DINO	Supervised	MoCo v3
AD↓	GradCAM [71]	0.21	0.40	0.45	**0.45**	0.66	**0.46**	**0.49**
	GradCAM++ [72]	0.21	0.40	0.43	0.67	0.84	0.62	0.71
	XGradCAM [73]	0.21	0.40	0.45	0.87	0.78	0.86	0.84
	Score-CAM [74]	**0.17**	**0.29**	**0.26**	0.50	**0.51**	0.66	0.75
AI↑	GradCAM [71]	0.44	0.17	0.12	**0.11**	0.13	**0.07**	**0.17**
	GradCAM++ [72]	0.41	0.16	0.12	0.05	0.04	0.04	0.08
	XGradCAM [73]	0.44	0.17	0.12	0.02	0.05	0.02	0.04
	Score-CAM [74]	**0.49**	**0.22**	**0.23**	**0.11**	**0.16**	0.06	0.10
D↓	GradCAM [71]	**0.08**	**0.08**	**0.08**	**0.13**	0.17	**0.16**	**0.10**
	GradCAM++ [72]	**0.08**	0.09	0.09	0.22	0.24	0.20	0.13
	XGradCAM [73]	**0.08**	**0.08**	**0.08**	0.35	0.27	0.31	0.21
	Score-CAM [74]	0.09	0.10	0.10	0.18	**0.16**	0.21	0.14
I↑	GradCAM [71]	**0.51**	**0.49**	**0.51**	**0.58**	0.54	0.53	**0.54**
	GradCAM++ [72]	**0.51**	0.48	0.50	0.52	0.47	0.51	0.50
	XGradCAM [73]	**0.51**	**0.49**	**0.51**	0.43	0.43	0.43	0.41
	Score-CAM [74]	**0.51**	0.47	0.48	0.55	**0.57**	**0.54**	**0.54**

3.4.2 Interpretability of transformers

In this subsection, we conduct extensive experiments using a variety of visualization methods to interpret transformers, trained in a supervised or self-supervised way. The obtained saliency maps are visually evaluated on several random samples derived from the ImageNet validation set. Then, 1000 random images, derived again from the Imagenet validation set, are used to evaluate all methods quantitatively with the average decrease (AD), average increase (AI), deletion (D) and insertion (I) metrics.

Raw attention maps In this experiment we visualize the attention map of the [cls] token from the last layer for supervised and self-supervised transformers. The models we choose, use as backbones DeiT base and ViT base. For the DeiT base experiments, we obtain the attention map when the network is untrained as well. From a quick look in Table 6, it is easy to see that both supervised networks based on different backbones have a similar way of predicting. According to their maps, both networks pay more attention to scattered information on the background of the images and less to the object. This is unexpected for us and we choose to further investigate this subject in the next experiments. On the other hand, the three self-supervised models work in a completely different way. These models pay attention mainly to regions on the object. All attention maps of these models seem reasonable to us and there is no considerable difference between the maps when changing the self-supervised approach or the backbone. For the untrained network, we characterize its attention map as spread out in general and sometimes acceptable on the object with some context information.

Table 6: Raw attention maps for two input images, obtained from untrained, supervised and self-supervised deep neural networks, that are based on different backbones.

Target class	Sample	DeiT base			ViT base		
		Untrained	Supervised	MoCo v3	Supervised	DINO	iBOT
goose							
bighorn							

All the previous observations are expressed quantitatively for the DeiT base experiments in Table 7. We can see the clear superiority of MoCo v3 over the supervised model. Here, it is unexpected that the untrained model achieves better AD scores than MoCo v3. This is justified by the fact that the attention maps from the untrained model are always spread out, although not acceptable most of the time. As we saw previously in subsection 3.4.1, big saliency maps that contain both object and background information usually help the model to be more confident and therefore the AD or AI can be favoured. Here, we strongly believe that the more evaluation metrics one uses, the better understanding of the saliency maps he/she has. A complete study on this should provide *localization metrics* as well [74, 79–81].

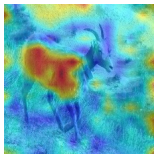
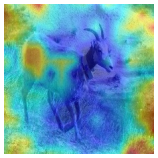
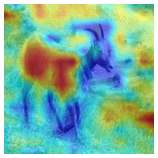
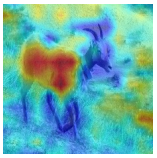
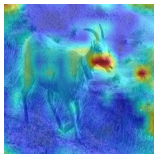
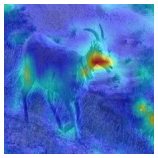
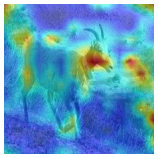
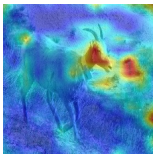
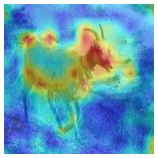
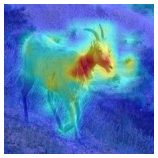
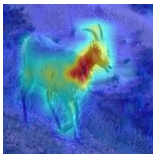
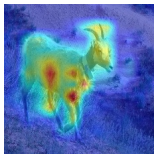
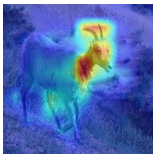
Briefly, in this experiment, we show reasonable attention maps for the self-supervised models when visualizing the raw attention map of the [cls] token of the last layer. On the

Table 7: Quantitative evaluation of raw attention maps. Classification metrics on 1000 randomly picked images from the ImageNet validation set. AD/AI: average drop/increase I/D: insertion/deletion ↓ / ↑: lower / higher is better.

Metric	DeiT base		
	Untrained	Supervised	MoCo v3
AD↓	**0.40**	0.95	0.52
AI↑	0.08	0.01	**0.13**
D↓	0.29	0.28	**0.10**
I↑	0.45	0.46	**0.55**

other side, the raw attention maps derived from the supervised models show us that these models pay attention mainly to context information from the background. Not focusing much on the object and achieving such great performance in multiple computer vision tasks was something we did not expect. In the next experiments, we investigate this further and search for the information these models use before predicting a class category.

Table 8: Raw attention maps of supervised and self-supervised transformers derived from different layers, given an input image.

Layer	Input image	DeiT base		ViT base		
		Supervised	MoCo v3	Supervised	DINO	iBOT
1_{st}						
2_{nd}	bighorn					
4_{th}						
6_{th}						
8_{th}						
10_{th}						
12_{th}						

Discovering hidden information from the raw attention maps visualization Here, in the first experiment, we visualize raw attention maps derived from different layer depths for both supervised and self-supervised models. The backbone networks these models use are again DeiT base and ViT base. Initially, Table 8 shows that from supervised models we observe similar attention maps no matter what backbone networks they use. This holds for self-supervised models as well. Our next step is to analyze how the attention map changes from the first layer to the deepest layer of the supervised models. In the beginning, the salient region is more random or noisy, but as we reach the middle layers we observe the salient regions on the object. Moving from the middle layers to the last one, we see the salient regions scattered across the background. Similarly, in the self-supervised models, we see random salient regions at first, but as we move to the middle layers, we see salient regions covering the entire object. As we go deeper, the salient regions stay on the object but get limited in extent. The $6th$ layer seems to be the one in that both supervised and self-supervised networks have similar attention maps. In Figure 14, we express the previous observations quantitatively for models that use DeiT base as a backbone. For the attention maps of supervised models, we observe that the best AD, AI, D and I metrics are achieved in the middle layers. When using the self-supervised approach, we have again the best AD, AI in the middle layers and the best D, I in the last layer. We further estimate the classification metrics for the attention maps of an untrained DeiT base model and as we expected there is no significant variation as we change depth layers.

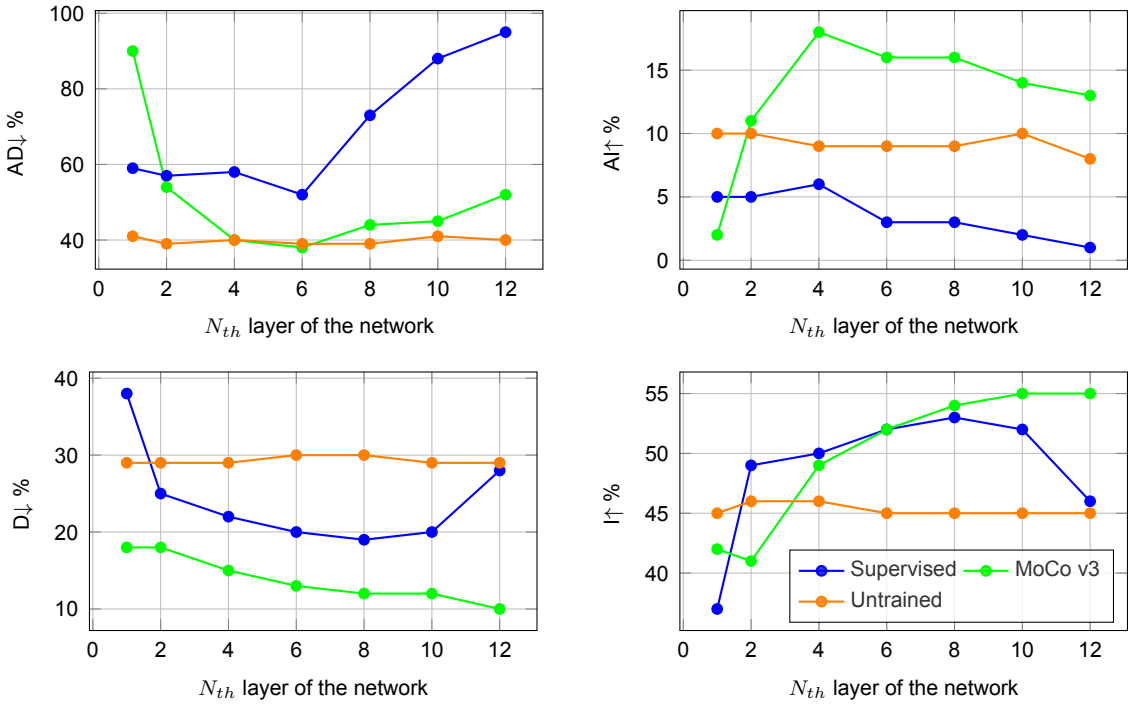


Figure 14: Quantitative evaluation of raw attention maps obtained from different layers. Classification metrics on 1000 randomly picked images from the ImageNet validation set. AD↓/AI↑: average drop/increase I↑/D↓: insertion/deletion ↓ / ↑: lower / higher is better.

In Figure 15, we present the raw attention maps for the supervised DeiT base model for a random input image, derived from different heads of the last layer. We show also the average attention map of all heads. While the majority of the heads pay attention to scatter information of the background, there are only a few that focus on the object, as a result, we observe salient regions in the context of the image for the average attention map as well. In short, in these experiments, we show for supervised transformers that although

(a) Input image (b) 1_{st} head (c) 6_{th} head (d) 12_{th} head (e) Average

Figure 15: Average attention map and maps from different heads. All the maps are obtained from the last layer of the supervised DeiT base model.

there are no salient regions on the object in the raw attention map of the last layer, saliency maps on the object can be found inside different heads or in the raw attention maps of the middle layers. We also prove that we can improve the AD, AI metrics for self-supervised models when obtaining raw attention maps from middle layers.

Pre-processing raw attention maps for better interpretability As described in detail in section 3.1, in this experiment, we use the power and $\log$ to boost the next highest attended regions on an image to improve interpretability, especially on the supervised transformers. In Figure 16, we see that for power values $\in (0, 1)$, as we reduce the value, the raw attention map becomes more reasonable. This kind of filtering helps us to interpret easier the predictions of the supervised transformers. The $\log$ function has the same influence on the raw attention maps as power when raising the raw attention map to power values close to 0.

(a) Input image (b) Raw (c) Raw-Pow(0.5) (d) Raw-Pow(0.3) (e) Raw-Pow(0.05) (f) Raw-Log

Figure 16: Pre-processing functions of raw attention maps for better interpretability. The attention maps are obtained from the supervised DeiT base model.

Alternative visualization methods based on the tensor of keys Just as we explain in section 3.1, in this experiment, we visualize information coming from the tensor of keys of transformer. From Figure 17, we observe that by simply visualizing the Keys$_{norm}$ map, we notice that reasonable salient regions are presented on the image despite being spread out. As we have already proved, the raw attention maps of the last layer for supervised models contain mostly salient context information, but when visualizing Raw*, which is the map when the Softmax operation is absent, there are salient regions both on the object and background. The Raw* maps are too noisy but if pairwise multiplied with the Keys$_{norm}$ maps, we observe acceptable and less noisy saliency maps.

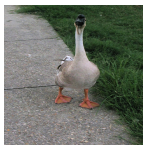
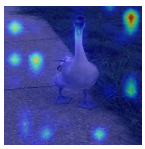
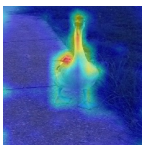
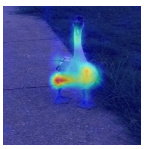
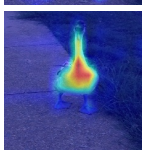
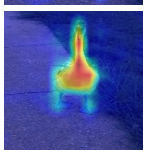
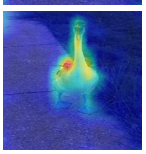
(a) Input image (b) Raw (c) Raw* (d) $\text{Keys}_{\text{norm}}$ (e) Raw* $\times$ $\text{Keys}_{\text{norm}}$

Figure 17: Visualization of saliency maps based on the raw attention map and the tensor of the keys. The maps are obtained from the supervised DeiT base model.

Rollout [18] Rollout is a standard visualization approach for transformers which linearly combines the attention maps from all layers. Qualitatively from Table 9 and quantitatively from Table 10, we observe better results when using the Rollout method compared to the raw attention maps from the last layers. Based on the Rollout, we further use it for specific groups of layers. Rollout_{10-12} provides similar but less noisy attention maps compared to the standard Rollout method. This is expressed, as we expect, with a little worse results on the AD, AI metrics for both supervised and self-supervised models. Then, we linearly combine attention maps from middle layers and visualize and evaluate the obtained Rollout_{3-5} maps. As we expected, these maps are informative, especially for supervised models. As Rollout_{3-5} map is derived from middle layers, its final attention map provides us with the salient region on the object. Quantitatively, when compared with the Raw method, all the metrics are improved as well. Rollout_{3-5} when used with self-supervised methods keeps the reasonable maps but adds some context information on them. This is expressed quantitatively with better AD, AI and worse D, I scores.

TIBAV [19] TIBAV is the only attention-based visualization method for transformers that is dependent on a certain class of a given input image. In this experiment, we use the method to interpret both supervised and self-supervised transformers. TIBAV is the method that succeeds to bring closer the qualitative and quantitative results of supervised and self-supervised models, according to Table 9 and Table 10. The comparison shows us again that the maps derived from the self-supervised model are better.

Table 9: Attention maps derived from Raw, standard Rollout and its alternatives [18] and TIBAV [19] methods

DeiT base	Input image	Raw	Rollout [18]	Rollout_{10-12}	Rollout_{3-5}	TIBAV [19]
Supervised						
	goose					
MoCo v3						

Discussion on quantitative results of all visualization methods In Table 10, we summarize all the visualization methods proposed in the previous experiments for interpreting the prediction of supervised and self-supervised transformers. The supervised and self-supervised models use the DeiT base as backbone. To begin with, there are several

methods we propose that improve the evaluation metrics of the Raw method for both supervised and self-supervised models. From a quick look, we observe that in almost all methods the saliency maps obtained from the self-supervised model achieve the best evaluation metrics. The best AD, AI performances are reached in the supervised model when using the Raw$^{*}\times$ Keys$_{\mathrm{norm}}$ method and in the self-supervised model when using either the Raw* or the Raw-Log method. The best D, I scores are achieved for both supervised and self-supervised models when using the TIBAV [19] method.

Table 10: Quantitative evaluation of saliency maps derived from a supervised and self-supervised DeiT base model. Classification metrics on 1000 randomly picked images from the ImageNet validation set. AD/AI: average drop/increase I/D: insertion/deletion ↓ / ↑: lower / higher is better.

Method	Classification metrics for quantitative evaluation							
	AD↓		AI↑		D↓		I↑	
	Supervised	MoCo v3	Supervised	MoCo v3	Supervised	MoCo v3	Supervised	MoCo v3
Raw	0.95	0.52	0.01	0.13	0.28	0.10	0.46	0.55
Raw-Log	0.54	**0.25**	0.07	**0.25**	0.24	0.10	0.47	0.55
Raw-Pow(0.5)	0.59	0.26	0.07	0.24	0.25	0.10	0.47	0.55
Raw*	0.54	**0.25**	0.07	**0.25**	0.24	0.10	0.47	0.55
Keys$_{\mathrm{norm}}$	0.49	0.41	0.07	0.15	0.30	0.34	0.42	0.29
Raw$^{*}\times$ Keys$_{\mathrm{norm}}$	**0.40**	0.40	**0.08**	0.17	0.19	0.14	0.52	0.45
Rollout [18]	0.88	0.39	0.02	0.17	0.22	0.10	0.50	0.55
Rollout$_{10-12}$	0.90	0.43	0.02	0.16	0.23	0.10	0.49	0.56
Rollout$_{3-5}$	0.52	0.37	0.07	0.18	0.21	0.14	0.51	0.50
TIBAV [19]	0.70	0.68	0.04	0.08	**0.14**	**0.09**	**0.57**	**0.57**

3.5 Conclusion

The chapter 3 is dedicated to the visual explanation of supervised and self-supervised learning. This study starts with the interpretability of both CNNs and transformers with four CAM-based methods. In these experiments, we show that CAM-based methods are proper visualization tools only for CNNs. The comparison of supervised and self-supervised models, that use the ResNet 50 as backbone architecture, gives us some valuable information. The saliency maps derived from both self-supervision and supervision are acceptable on the object, but the latter is spread more across the background. By taking some extra information from the background while having acceptable maps on the object, the supervised methods result in better quantitative results than self-supervised. This is an insight that models need other than the information from the foreground of an object some extra context from the background to be more confident in their predictions. Quantitatively, we observe that Score-CAM [74] is the best method for all the CNN architectures.

CAM-based methods do not help us to reveal important observations for transformers; therefore, we continue our study by further investigating them with attention-based methods. We choose first to visualize the raw attention map of the [cls] token from the last layer of the model. The visualizations give us reasonable saliency maps on the object for the self-supervised methods. On the other hand, the Raw$_{12}$ of supervised models show us that these models pay attention mainly to the context information from the background. Not focusing on the object while achieving high performances in multiple computer vision tasks was something we did not expect.

The unexpected Raw$_{12}$ results for supervised transformers urge us to explore more these networks. In the next experiments, we try to discover hidden information inside the models and start this by visualizing raw attention maps derived from different layers. Here, we

see that supervised models focus on the object in the middle layers of their architecture and therefore for these depths we observe more similar attention maps between supervised and self-supervised models. Quantitatively, as shown for instance with the Raw_4 and the Raw_6 methods, this is expressed with way better results for supervised models and with an improvement in average drop (AD) and average increase (AI) metrics for the self-supervised models. The improvement we observe in the AD and AI metrics for the self-supervised models is again proof that context matters and can further give confidence to a model before its prediction. Going back to the Raw_{12} qualitative result of supervised models, we observe that object information can be found inside the different heads of the last layer of the model. But while most of the heads focus on the context and as Raw_{12} is an average attention map obtained from all heads, the final map will contain mostly context information. To reveal the foreground information of an object that is being hidden in the Raw_{12}, we propose two new visualization methods based on the $\log$ and power operations. The two methods made the interpretability of the supervised models a lot easier and improved significantly the quantitative results of both supervised and self-supervised models. Furthermore, we propose Raw^* which is simply the visualization of the raw attention map from the last layer of the model when not taking into account the Softmax operation. This method improves a lot both qualitative and qualitative results for supervised and self-supervised models and shows us the amount of information Softmax operation can hide. The last attention-based methods we used in the experiments are Rollout [18] and TIBAV [19] methods. Rollout works better than Raw_{12} and in our study, we propose alternatives to the method to further investigate groups of layers of a model. TIBAV provides us with reasonable saliency maps for both supervised and self-supervised and diminishes the gap between the two types of learning. Quantitatively, TIBAV achieves similar classification metrics for supervised and self-supervised models and the best deletion (D) and insertion (I) scores.

Transformers are complicated and information is shared across multiple linear projections. Information could be found inside the queries Q, keys K and values V tensors as well. In our study, we choose to further investigate the K. We propose $\text{Keys}_{\text{norm}}$ and $\text{Raw}^* \times \text{Keys}_{\text{norm}}$, two new visualizations methods that provide saliency maps that contain information coming from K. Qualitatively, we observe reasonable maps and this is an insight that our method can be further optimized. Here, we prove that we must take into account information coming from other linear projections as well to better interpret and understand transformers. The quantitative evaluation of the two methods for both supervised and self-supervised models shows they are pretty good at AD and AI metrics but not good enough at D, I metrics.

The overall conclusion of our study on interpretability is that for both CNNs and transformers the saliency maps of the self-supervised models contain mostly information coming from the foreground of the object. On the other hand, information derived from the background of the object seems important for supervised models. Furthermore, we show that there is no proper method to interpret both CNNs and transformers. This is something we expected, having in mind the different inner mechanisms of the architectures. The mechanism of transformers is complicated and, as we show, information is spread inside the network. Although we try a different kind of methods to reveal this information, we strongly believe that there is room for improvement. Quantitatively, with the new methods we propose, we reach better AD, AI, D and I scores than the Raw_{12} and Rollout baselines. By comparing our methods with TIBAV, we achieve better AD and AI but worse D and I scores. The last comparison is not so fair because TIBAV is a class-specific method while the others are not.

4. MIM ON SELF-SUPERVISION

This chapter is devoted to the presentation of our work on self-supervised learning, in particular when using masked image modelling as a pre-text task to further improve and understand the representation learning process. We first explain thoroughly the methodology we follow in our experiments and highlight the most important contributions of our work. We then present the dataset, network and evaluation protocol we use. Here, we also provide you with valuable implementation details. We close this chapter with the experimental results and the conclusions.

4.1 Methodology

Inspired by the interpretability observations of chapter 3 and from the papers of iBOT [9] and AttMask [10], we follow a methodology with the main purpose to improve the self-supervision of MIM-based models. All models are pre-trained from scratch and interpreted as well, during and after the training process. We organize our methodology in three parts that are described in detail below.

Figure 18: Different masking strategies for a given input image (a). The strategy (b) is used by SimMIM [14], (b) by MAE [15], (d) by BEiT [16] and iBOT [9], (e) by AttMask [10] and (f) by MST [17]. Both (e),(f) are based on the attention map (g). The source of the images is [10].

Alternative masking strategies In subsection 2.1.4, we describe some of the masking strategies used in MIM-based self-supervised methods. We show some of them in Figure 18. Following a study on them and a series of conducted experiments, we cannot argue with AttMask [10] approach. This masking strategy claims that an attention-guided token masking strategy, which hides tokens that correspond to the salient regions of an attention map derived from the last layer of a model, offers several benefits over random masking when used in MIM-based self-supervised learning. Through the study, we found that the more challenging the reconstruction task for the MIM-based self-supervised method, the better the representation learning. Reconstructing regions of the object of one image is more challenging for a model than reconstructing regions of the background. Therefore, we believe AttMask [10] ends up with better evaluation scores than AttMask Low (MST [17]). We consider that a challenging task is to hide regions of an image that make a model confident before its prediction. A good question here could be “Which are the regions of an image that make a model confident before its prediction?”. The saliency map regions could be a possible answer here! In chapter 3, we studied a series of visualization methods from which we obtained different kinds of saliency maps that were evaluated qualitatively and quantitatively. The average drop (AD) and average increase (AI) are two classification metrics that give as a good indication of how confident is a model when seeing only the salient regions of the map before its prediction. From Figure 14 and

Table 10, we observe that raw attention maps derived from the last layer of model, which are used in mask generation in AttMask [10], do not have the best AD, AI metrics. Based on this, we believe that we can create a more challenging MIM objective, by generating masks from attention-based methods used in section 3.1. From subsection 3.4.2, we observe that a model is confident about its prediction when other than the foreground object sees some context information from the background as well. Based on this, we propose new masking strategies that hide from the student network the aforementioned regions of an image.

- *Mask generation from different layers:* Here, we use iBOT [9] as our baseline MIM-based model along with its released code. We first replace its masking strategy [16] with the respective of AttMask [10]. We use this strategy, then, in an extensive study for mask generation from different layers of the model. In this work, we call *AttMask-Raw*$_5$ the strategy which generates masks derived from the 5_{th} layer attention map etc. All the strategies are incorporated into iBOT and the models are trained from scratch and evaluated with k-NN and linear probing. The evaluation of these strategies is found in Figure 19.
- *Rollout [18] method for mask generation:* While AttMask [10] generates masks from raw attention maps, we propose to use the attention map derived from the Rollout [18] method for mask generation. As in section 3.1, we use Rollout to obtain an attention map from a specific group of layers as well to further generate different kinds of masks. The various kind of masks generated from this method, help us better understand the impact of the mask on the learning process. We call *AttMask-Rollout* the strategy that generates masks from the final rollout attention map and *AttMask-Rollout*$_{5-8}$ the one that generates masks from the attention map derived from the linear combination of the $5_{th}, 6_{th}, 7_{th}$ and 8_{th} layer of the model. The results of these strategies are in Table 11.
- *Pre-processing of attention maps for competitive MIM:* In section 3.1, we use the power and $\log$ to boost the next more high attended regions of an image to improve interpretability. Here, we use these two operations to introduce two new masking strategies. To be more specific, in both strategies we first pre-process the raw attention map of the last layer of the model with either the $\log$ or power operation and then we generate masks from the pre-processed maps. We name *AttMask-Log* the masking strategy that is based on $\log$ function and *AttMask-Pow* the one that uses the power operation respectively. In AttMask-Pow, the smaller the power value $\in (0, 1)$, the more context of an image will be hidden from the generated mask. For power values close to 0 the method generates similar masks with the AttMask-Log. The evaluation of the two proposed masking strategies is available in Table 12.
- *Multi-layer mask generation:* As we explained in subsubsection 2.1.4.1, iBOT [9] and AttMask [10] follow the multi-crop strategy of DINO [8]. Based on this strategy, from a given input image a set of different views $\in V$ is generated. There are two global views $\in V_g$ and several local views $\in V_l$ of smaller resolution. Through the teacher network, only the global views are passed while through the student both masked global views and unmasked local views. In iBOT, both global views are masked with the random block-wise [16] strategy before being passed through the student network and in AttMask, both global views are masked with the proposed attention-guided strategy. Here, we propose a multi-layer mask generation strategy, where the two global crops are masked with a different kind of strategy. We call *AttMask-Raw*$_{5,12}$ the strategy that masks the first crop with AttMask-Raw$_5$ and the second one with AttMask-Raw$_{12}$. We compare this method with the standard iBOT and AttMask

baselines. We further use a third global crop and propose AttMask-Raw$_{3,5,12}$, where we generate three different masks coming from Raw$_3$, Raw$_5$ and Raw$_{12}$, one for each crop. We set then the global crops to 3 for the iBOT and AttMask strategies as well and compare them with AttMask-Raw$_{3,5,12}$. All the results are found in Table 13.

Interpretability of MIM-based models Once the training of the iBOT [9] models with all the different kinds of masking strategies is finished, we interpret them. As an interpretability method, we choose to visualize the raw attention map of the [cls] token, because it is the easiest and most popular approach to interpret the prediction of a transformer. We obtain this map from the last layer of the model. We follow the same evaluation protocol as in subsection 3.3.3; therefore, we first evaluate the saliency maps qualitatively, given several input images derived from the ImageNet validation set. We then use the standard 1000 random images subset, derived from the ImageNet validation set, to evaluate the networks quantitatively with the average decrease (AD), average increase (AI), deletion (D) and insertion (I) metrics. Here, we have the chance to see the impact of each masking strategy on the saliency map qualitatively and quantitatively as well. Because we pre-train the models on different scales of ImageNet we further investigate the influence of the scale on the saliency maps. All the results are presented in subsection 4.4.2.

As we train the iBOT models from scratch, we interpret the networks in the intermediate stages of learning as well. In this way, we further study the evolution of the raw attention map per epoch, better understanding how quickly the models learn and where they mostly pay their attention during training. Here, we visualize again the raw attention map from the last layer of the model. We show part of the qualitative results in Figure 22.

Contrastive learning AttMask [10] uses the same loss function with iBOT [9], which is a weighted sum of L_{MIM} (2.19), L_{CLS} (2.20). Both are cross-entropy losses and L_{MIM} is calculated between predicted projections of patch tokens and target non-masked patch tokens projections. This dense distillation loss can be seen as a reconstruction loss of the hidden patch tokens and is based only on the global views of the multi-crop strategy. On the other hand, L_{CLS} global cross-entropy loss is calculated between teacher and student [cls] output representations and is based on both global and local views.

So far in our study, we investigate the impact of changing the masking strategy in a MIM-based self-supervised approach; therefore, we keep the standard loss of iBOT. Here, we study the influence of the addition of an extra 3_{rd} contrastive term on the previous loss. Based on the standard InfoNCE loss (2.11), we formulate a dense contrastive loss. This loss is based on unmasked global views. To be more specific, for a given image, we obtain the output patch token embeddings of the teacher network for the n_g global views. From the attention map of [cls] token, we know the highest attended tokens and lowest ones. Based on this, we order in a descending way $\vec{a}_{\text{[cls]}}$ and define the k first elements as *positives* and the k last elements as *negatives* for the n_g global views, where k is a user-defined hyper-parameter. The total number of positives T is equal to the total number of negatives and given by kn_g. By choosing a different anchor p_i each time derived from the positives p, we bring p_i closer to the rest of positives p_j and we push p_i apart from all the negatives n, when minimizing

$$L_{\text{InfoNCE}-\text{dense}} = -\frac{1}{T}\sum_{i=1}^{T}\sum_{\substack{j=1 \\ j\neq i}}^{T}\log\frac{\exp(p_i p_j/\tau)}{\exp(p_i p_j/\tau)+\sum_{n}\exp(p_i n/\tau)}. \tag{4.1}$$

$L_{\text{InfoNCE}-\text{dense}}$ is the dense contrastive loss we propose and τ a temperature hyperparameter that is defined from the user. With the addition of the contrastive term, we define iBOT objective

$$L_{\text{iBOT}-\text{contr}} = \lambda_1 L_{\text{MIM}} + \lambda_2 L_{\text{CLS}} + \lambda_3 L_{\text{InfoNCE}-\text{dense}}, \tag{4.2}$$

where $\lambda_1, \lambda_2, \lambda_3$ the weight coefficients of each loss, which are user defined hyperparameters.

In our methodology, we propose the extra contrastive term to bring high-attended tokens from different global views closer and push apart all the low-attended tokens from the high-attended ones. In this way, we make the model discriminate better between the foreground and background of an object. In our experiment, we examine if the 3_{rd} term has good or bad influence on the k-NN and linear probing evaluation scores. We first try this loss when following the AttMask strategy. Here we set $\lambda_1, \lambda_2 = 1$ and perform hyperparameter tuning to find the best k, λ_3 and τ parameters. We hold these hyperparameters and use the loss to experiment with other masking strategies as well. We present part of the results in subsection 4.4.3.

4.2 Contributions

In this subsection, we summarize the most important contributions of our study on MIM-based self-supervision, and we present them below:

1. We propose at least 7 new masking strategies to use on MIM-based self-supervision. Here we succeed to surpass all the previous state-of-the-art strategies on k-NN and liner probing evaluation scores when the models were trained on different scales of ImageNet and for a different number of epochs. We further show acceleration in the learning process of downstream tasks.
2. Through the study on the masking strategies, we come up with valuable insights on which regions of an image are most important to be hidden from the student network and define a more challenging MIM-based self-supervision pre-text task. We further highlight the impact of the training epochs, the scale of the dataset and the number of global views on the evaluation scores and how to narrow the gap between the masking strategies.
3. We also interpret these networks and show the impact a change in masking strategy and scale of a dataset bring to a saliency map. Here, we further investigate the evolution of saliency maps during the learning process.
4. Finally, we introduce a contrastive term on a MIM distillation-based objective and study the influence when used with different masking strategies.

4.3 Experimental setup

In this section, we present the experimental setup we follow in chapter 4. We provide information about the dataset, network and evaluation protocol we use and we show some valuable implementation details.

4.3.1 Dataset

In this chapter, we conduct all the experiments on ImageNet [12] dataset again. To be more specific, during training we use the *full* train set of Imagenet or a *subset*. For the subset, we select the first 20% of training samples per class. For the evaluation of the models, we use the full validation set of ImageNet. To interpret the trained networks, we use the standard subset of chapter 3.

4.3.2 Networks

All the masking strategies we explore are incorporated into iBOT [9]. The iBOT architecture we choose uses as transformer encoder a ViT-S/16 model as in [10]. Further architecture details can be found in subsection 4.3.4.

4.3.3 Evaluation protocol

As previously mentioned, we evaluate the networks on the ImageNet [12] validation set. Following DINO [8], iBOT [9] and AttMask [10], we use k*-NN* and *linear probing* for the evaluation. For k-NN, we first freeze the pre-trained model and extract the features of the training images. We then use a k-nearest neighbors classifier with $k = 20$ and summarize only the top-1 scores. In some of the experiments, we evaluate the models for $k = 10, 100, 200$ as well. For the linear probing evaluation, we first freeze the pre-trained model and then train a linear classifier using SGD with a batch size of 1024 for 100 epochs. The learning rate is set to 0.003 and we do not apply weight decay. Following [8–10], we give as input to the linear classifier the concatenation of the [cls] features from the last four layers of the pre-trained model. Again, we record only the top-1 accuracy scores.

To interpret the pre-trained from scratch iBOT models, we use the same evaluation protocol as in section 3.3. The raw attention maps derived from the last layer of the model are first evaluated qualitatively and then quantitatively on the 1000 ImageNet subset, by using the average decrease (AD), average increase (AI), deletion (D) and insertion (I) metrics.

4.3.4 Implementation details

The released iBOT code is our baseline for these experiments. Based on the implementation details provided in [10], we further develop the code to reproduce AttMask model, which is the model we mainly compete with. The final code with all the proposed masking strategies along with the model that performs contrastive learning are available:

`https://github.com/DimitrisReppas/MIM_on_self-supervision`

Training details We pre-train iBOT models on 20% or 100% of the ImageNet train set. The images are reshaped to $224 \times 224 \times 3$. The iBOT models we train, use ViT-S/16 as the backbone architecture. For the 20% of ImageNet experiments, we train the models for 100 epochs using distributed parallel processing with 16 GPUs. The global batch size is set to 256 and the batch per GPU to 16. We use AdamW as an optimizer and warm-up learning rate η for 10 epochs following the linear scaling rule $\eta = 5 \times 10^{-4} \times \mathtt{bs}/256$, where $\mathtt{bs}$ is the batch size and then decay using a cosine schedule. For weight decay, we use

a cosine schedule from 0.04 to 0.4. We also set student temperature to 0.1 and teacher momentum to 0.99. Following [8, 10], for the first 30 epochs, we use a linear warm-up for teacher temperature from 0.04 to 0.07. We follow a multi-crop strategy with 6 local crops and we experiment with 2 and 3 global crops. Global crop scales are sampled from $(s, 1)$ and local crop scales from $(0.05, s)$ and s is set to 0.25. As data augmentations, we use gaussian blur, colour jittering and solarization. Finally, we use a shared projection head for [cls] and patch tokens, of dimensionality 8192 and we do not perform weight normalization on the last layer of the MLP heads.

For the 100% of ImageNet experiments, we pre-train the models for 100 and 300 epochs. When the pre-training is for 100 epochs, we use again distributed parallel processing with 16 GPUs and a global batch size of 256. The setup is the same as on 20% of the ImageNet experiment, except for increasing the number of local crops to 10 and teacher momentum to 0.996. When the pre-training is for 300 epochs, again the setup is the same. The only difference is that we use distributed parallel processing with 32 GPUs, we set the global batch size to 800 and s to 0.32.

Masking probability and mask ratio Here, we follow the exact strategy of [10] in all the masking strategies we propose and incorporate into iBOT. To be more specific, first, each image is masked with probability $p = 0.5$. Then, from the total n patch tokens only the $k = \lfloor rn \rfloor$ highest-attended tokens will be masked. The mask ratio r is sampled uniformly per image as $r \sim U(a, b)$ with $[a, b] = [0.1, 0.5]$.

Details for the interpretability experiments To interpret the iBOT models that are trained with different masking strategies, we visualize the raw attention maps of the [cls] token from the last layer of the model. Here, we use the same code as in subsection 3.3.4 and follow the same image pre-processing and normalization techniques.

Details for the contrastive learning experiments For the experiments where we introduce the extra contrastive term, as thoroughly described in section 4.1, we set $\lambda_1, \lambda_2 = 1$. After a hyperparameter tuning, we set $k = 6, \tau = 0.07, \lambda_3 = 1$. Then, we use these parameters in the rest experiments.

4.4 Experimental results and discussion

In this section, we present and organize the experimental results of the chapter as follows. We first show k-NN and linear probing scores of iBOT models trained from scratch with different masking strategies. We then interpret the predictions of the models and provide qualitative and quantitative results. We finally present k-NN and linear probing scores when replacing the standard iBOT loss with a loss that contains a contrastive term as well. All the results are discussed thoroughly.

4.4.1 Masking strategies

In this subsection, we conduct experiments by using a variety of masking strategies that we incorporate into iBOT approach. The masking strategies we propose are compared with

baseline strategies and all are evaluated with k-NN and linear probing. All observations are discussed in detail to conclude which strategy is more beneficial for MIM-based self-supervised learning.

Mask generation from different layers In this experiment, we pre-train the iBOT model on the 20% ImageNet for 100 epochs while using the attention-guided token masking strategy (AttMask [10]) for different layers of the model. We summarize the results in two plots, one for the k-NN and one for the linear probing scores in Figure 19.

From the k-NN plot, we observe the AttMask-Raw$_5$ mask strategy performs better than the AttMask baseline, with a gain of 0.1%. The AttMask-Raw$_4$ performs equally good with the baseline and better than the rest of the methods.

From the linear probing plot, we see that both AttMask-Raw$_4$ and AttMask-Raw$_5$ mask strategies perform better than the AttMask baseline. The gain here for both mask strategies is 0.5%. The AttMask-Raw$_3$ achieves equally good accuracy with the baseline and better than the rest of the methods.

The previous two plots, show us that when using masks generated from attention maps derived from the 4_{th}, 5_{th} we perform equally good or even better than the AttMask baseline. This is the first insight that although hiding the object from the student network is proven to be effective, hiding the object and some context information may result in a more competitive MIM-based self-supervised approach.

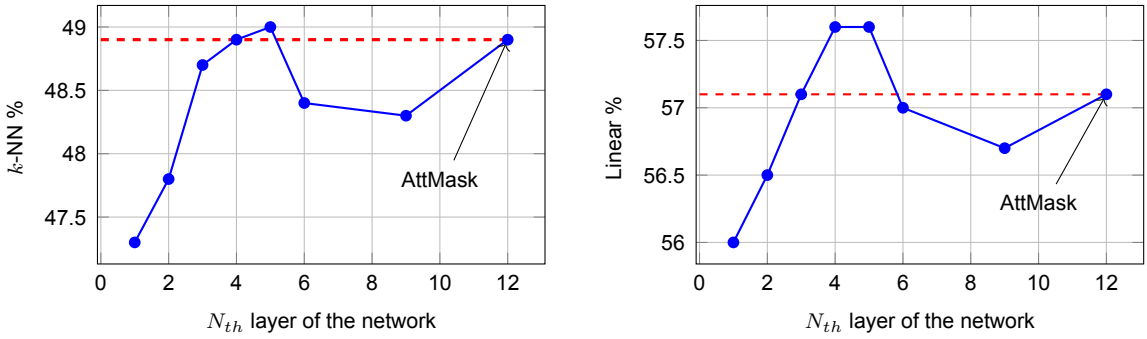


Figure 19: Linear probing and k-NN scores of AttMask for different layers.The AttMask for the 12_{th} layer is the baseline approach [10]. The models are trained on the 20% ImageNet and evaluated on the whole validation set of the dataset.

Rollout [18] method for mask generation Based on the previous observations, we want to further investigate the context information impact by taking advantage of the Rollout [18] method. The mask generated from the Rollout method is compared with three baseline mask strategies, which are the random block-wise [16] used in iBOT [9], the AttMask-Low [17] and the AttMask [10]. Here, we pre-train again the iBOT model on the 20% ImageNet for 100 epochs.

From a quick look in Table 11, we observe that from the three baselines, we achieve the best scores from AttMask and the worst scores from the AttMask-Low approach. In other words, during MIM-based self-supervised learning masking only regions from the background is the least effective approach while masking regions mostly on the object is the most effective one. The random block-wise [16] followed in iBOT [9] is in the middle.

In the AttMask strategy, masks are derived from the last layer of the model while in the AttMask-Rollout strategy the masks are obtained from the Rollout attention map. When comparing the two attention maps, as shown in Table 9, for the self-supervised approach we observe that the Rollout attention map contains salient regions on the object and less salient regions on the context than the attention map derived from the last layer of the model. This means that masks which hide only the object, is more probable to be generated from the AttMask-Rollout than the AttMask strategy. From Table 11, we see that the small context information the AttMask strategy hides from the student network gives the method a 0.3% gain in k-NN and a 0.1% gain in linear probing accuracy, compared to AttMask-Rollout. AttMask-Rollout$_{9-12}$ is a strategy that hides even less context; therefore, the gain when using AttMask is even bigger. On the other hand, when using the AttMask-Rollout$_{5-8}$ strategy we produce masks that hide greater context information than AttMask while still hiding the object as well. When comparing the two methods, the gain of using the AttMask-Rollout$_{5-8}$ is 0.2% in k-NN and 0.3% in linear probing accuracy. The worst mask strategy we tried was AttMask-Rollout$_{1-4}$, as this method generates masks that hide too much context information and not enough object information.

Table 11: Evaluation of Rollout-based masking strategies with k-NN and linear probing. The models are trained on the 20% ImageNet and evaluated on the whole validation set of the dataset.

iBOT Masking	Evaluation	
	k-NN	Linear
Random block-wise [16]	47.0	56.0
AttMask-Low [17]	44.0	53.2
AttMask [10]	48.9	57.1
AttMask-Rollout	48.6	57.0
AttMask-Rollout$_{9-12}$	48.5	56.7
AttMask-Rollout$_{5-8}$	**49.1**	**57.4**
AttMask-Rollout$_{1-4}$	47.9	56.7

The Table 11 shows as again similar insights to the previous experiment. First and foremost, the most important region to hide during MIM-based self-supervised learning is the object. If you hide some context information as well, the self-supervised approach will achieve better scores. On the other hand, if you hide too much context information and not enough object information, the performance will drop.

Pre-processing of attention maps for competitive MIM In this experiment, we pre-process the [cls] raw attention map from the last layer of the model with power and $\log$ functions to boost the saliency of some regions on the background. The mask generated from the pre-processed attention maps will hide more context information than the masks generated from the AttMask strategy. We conduct the experiment on the 20% ImageNet and we pre-train the iBOT models for 100 epochs.

Table 12 shows that pre-processing the attention maps from the last layer of the model with power function when using $power = 0.7, 0.5, 0.3$ has a positive impact on learning. In k-NN, AttMask-Pow(0.5) achieves a 0.4% gain over AttMask and in linear probing AttMask-Pow(0.3) reach a 0.6% gain over AttMask. On the contrary, the AttMask-Log strategy hides too much context information and not enough information coming from the object, resulting in a lower performance than the baseline.

Table 12: Linear probing and k-NN evaluation of masking strategies based on the pre-processing of the attention maps with power and $\log$ functions. The models are trained on the 20% ImageNet and evaluated on the whole validation set of the dataset.

iBOT Masking	Evaluation	
	k-NN	Linear
AttMask [10]	48.9	57.1
AttMask-Pow(0.7)	49.0	57.5
AttMask-Pow(0.5)	**49.3**	57.5
AttMask-Pow(0.3)	49.0	**57.7**
AttMask-Log	48.2	56.9

Multi-layer mask generation As previously mentioned, iBOT [9] and AttMask [10] follow the multi-crop strategy of DINO [8]. In our experiments, for all the proposed masking strategies we use 2 global crops as in DINO, iBOT and AttMask. In the experiment on the 20% ImageNet, the local crops are set to 6 as in AttMask. So far, we follow the strategy of iBOT and AttMask and mask the 2 global crops with the same kind of strategy. In this experiment, we mask the 2 global crops with 2 different strategies. Then, we add an extra global crop and mask it again with a different strategy.

For the 2 global crops experiment, we propose AttMask-Raw$_{5,12}$. In this strategy, the masks are derived from the attention maps of the 5_{th} and 12_{th} layer. Masking each of the 2 global crops with a different mask has a positive impact on the k-NN and linear probing scores, as shown in Table 13. AttMask-Raw$_{5,12}$ performs better than the 2 previous baselines and compared to AttMask, we achieve a 0.4% gain in k-NN and a 0.5% gain in linear probing.

The addition of the 3_{rd} global crop has a beneficial influence on all strategies, with a minimum 1.6%, maximum 2.5% gain in k-NN and a minimum 0.7%, maximum 1.4% gain in linear probing. The improvements in both scores come with an extra $\approx 47\%$ of the time for the 2 global crops experiment. Here, we propose AttMask-Raw$_{3,5,12}$ with the masks derived from the attention maps of the 3_{rd}, 5_{th} and 12_{th} layer of the model. The multi-layer mask strategy is again the best, with 0.4% gain in k-NN and equally good results with AttMask in linear probing.

In short, in this experiment, we see that using a multi-layer mask strategy and raising the number of global crops when a multi-crop strategy is followed, further improves a MIM-based self-supervised approach. Furthermore, it seems that with the addition of the extra crop, the gap between the masking strategies gets closed.

Table 13: Evaluation of the multi-layer masking and multi-crop strategies with k-NN and linear probing. The models are trained on the 20% ImageNet and evaluated on the whole validation set of the dataset.

Global crops	iBOT Masking	Evaluation	
		k-NN	Linear
2	Random block-wise [16]	47.0	56.0
	AttMask [10]	48.9	57.1
	AttMask-Raw$_{5,12}$	**49.3**	**57.6**
3	Random block-wise [16]	49.5	57.4
	AttMask [10]	50.5	**58.3**
	AttMask-Raw$_{3,5,12}$	**50.9**	**58.3**

Experiments on the full Imagenet In this experiment, we evaluate again the masking strategies we proposed, on the 100% ImageNet. First, we pre-train iBOT models for 100 epochs and then for 300 epochs as in [10].

For the 100 epochs experiment, we observe more or less the same insights, with the only difference that the score of the strategies comes even closer. So again, AttMask is better than AttMask-Rollout$_{9-12}$ and this time seems to be equally good with AttMask-Pow. The best strategy is AttMask-Raw$_{5,12}$ with a 0.1% gain in k-NN and a 0.2% gain in linear probing when compared to AttMask.

We pre-train the iBOT model for 300 epochs only for the AttMask and AttMask-Raw$_{5,12}$. Again, AttMask-Raw$_{5,12}$ is a bit better than AttMask with no gain in k-NN and a 0.3% gain in linear probing.

In summary, we show that in experiments on 100% ImageNet all strategies seem to get even closer and what to hide is not as important as in previous experiments. The AttMask-Raw$_{5,12}$ strategy is still the best strategy.

Table 14: Evaluation of masking strategies with k-NN and linear probing. The models are trained for 100 and 300 epochs respectively, on the 100% ImageNet and evaluated on the whole validation set of the dataset.

Pre-training epochs	iBOT Masking	Evaluation	
		k-NN	Linear
100	AttMask [10]	72.6	76.1
	AttMask-Rollout $_{9-12}$	72.4	76.0
	AttMask-Pow(0.5)	**72.7**	76.0
	AttMask-Pow(0.3)	72.6	76.1
	AttMask-Raw$_{5,12}$	**72.7**	**76.3**
300	AttMask [10]	**74.8**	77.2
	AttMask-Raw$_{5,12}$	**74.8**	**77.5**

A closer look on AttMask, AttMask-Raw$_{5,12}$ comparison Here, we explore in more detail the k-NN and linear probing results of the two strategies to gain a better understanding of their performance.

- *k-NN report:* In Table 15, we present in detail the k-NN evaluation of the two strategies for $k = 10, 20, 100, 200$. We show results when the models are pre-trained for 100 or 300 epochs, on the 20% or 100% ImageNet. From a quick look, AttMask-Raw$_{5,12}$ is the best strategy on this metric. If we look closer, this holds for the 100 epochs experiment for both 20% or 100% ImageNet. For the 300 epochs experiment on the 100% ImageNet, we observe that the two strategies are equally good. Only when $k = 10$ AttMask-Raw$_{5,12}$ performs slightly better than AttMask. It seems that the superiority of AttMask-Raw$_{5,12}$ for the 20% ImageNet experiments is progressively minimized when we use the full ImageNet and increase the number of train epochs.
- *Linear probing report:* In Figure 20 and Figure 21, we present plots of top-1 accuracy scores for the two strategies for a 100-epoch linear probing evaluation. The plot stops when each model reaches its maximum accuracy. In Figure 20, we show the scores for models pre-trained for 100 epochs on the 20% or 100% ImageNet. And in Figure 21, we see the respective results for models pre-trained for 300 epochs on the 100% ImageNet. First and foremost, with both strategies, the models reach their top accuracy before the 100_{th} epoch. For models pre-trained for 100 epochs

on the 20% ImageNet, AttMask-Raw$_{5,12}$ reaches the top accuracy of the AttMask in 55 fewer epochs. When pre-trained on 100% ImageNet for the same number of epochs, AttMask-Raw$_{5,12}$ reaches the top accuracy of the AttMask in 16 fewer epochs. For models pre-trained for 300 epochs on 100% ImageNet, the top score of the AttMask is achieved by AttMask-Raw$_{5,12}$ in 20 fewer epochs. Considering the previous observations, when choosing the AttMask-Raw$_{5,12}$ strategy to pre-train the iBOT model, we need to train less a classifier to use it for downstream tasks.

Table 15: Detailed k-NN evaluation report for AttMask [10] and AttMask-Raw$_{5,12}$ strategies.

Pre-training epochs	% ImageNet-1k	iBOT Masking	k-NN Evaluation			
			10	20	100	200
100	20	AttMask [10]	48.5	48.9	47.2	46.1
		AttMask-Raw$_{5,12}$	**48.8**	**49.3**	**47.6**	**46.5**
	100	AttMask [10]	72.7	72.6	70.7	69.5
		AttMask-Raw$_{5,12}$	**72.9**	**72.7**	**71.0**	**69.7**
300		AttMask [10]	74.8	**74.8**	**73.3**	**72.3**
		AttMask-Raw$_{5,12}$	**74.9**	**74.8**	**73.3**	**72.3**

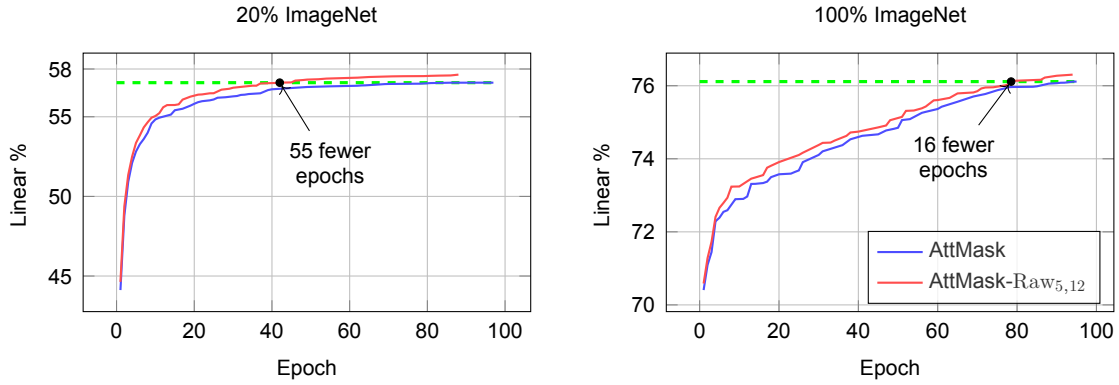


Figure 20: Top-1 accuracy scores for AttMask and AttMask-Raw$_{5,12}$ for a 100-epoch linear probing evaluation. The models are pre-trained for 100 epochs on the 20% (left) and on 100% ImageNet (right).

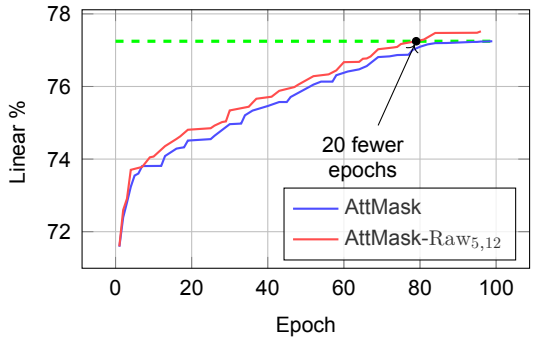


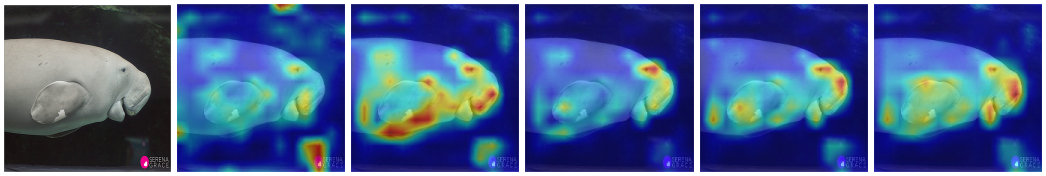
Figure 21: Top-1 accuracy scores for AttMask and AttMask-Raw$_{5,12}$ for a 100-epoch linear probing evaluation. The models are pre-trained for 300 epochs on the 100% ImageNet.

4.4.2 Interpretability of MIM-based models

In this subsection, we interpret the iBOT models trained with different masking strategies. To interpret them, we visualize the raw attention maps from the last layer of the models and evaluate them qualitatively and quantitatively. More precisely, we study the evolution of the attention map during training and the impact the masking strategy and the scale of a dataset have on interpretability.

The evolution of the attention map during training In Figure 22, we present the evolution of a saliency map during training. To be more specific, we visualize the raw attention map of the [cls] token, from the last layer of the iBOT model, when following the AttMask strategy. The model was pre-trained for 100 epochs on the 100% ImageNet and we show the attention maps for the $5_{th}, 25_{th}, 50_{th}, 75_{th}, 100_{th}$ training epoch, given an input image.

Looking closely at the evolution of the attention map, we observe that in the 5_{th} epoch there are scattered salient regions all over the image. After 20 epochs, the majority of the salient regions are found on the object while there are still some in the background. From the 50_{th} until the last epoch, we see no salient background regions and progressively more and more features of the object become part of the final saliency map.



(a) Input image (b) Epoch = 5 (c) Epoch = 25 (d) Epoch = 50 (e) Epoch = 75 (f) Epoch = 100

Figure 22: The evolution of the raw attention map during pre-training when AttMask [10] strategy is incorporated into iBOT [9]. The training lasts for 100 epochs and is on the 100% ImageNet.

The impact of masking and scale of a dataset on interpretability In this experiment, we interpret iBOT models trained for 100 epochs, on the 20% and 100% ImageNet, with three different masking strategies. AttMask, AttMask-Pow(0.5) and AttMask-Raw$_{5,12}$ are the three mask generation methods we choose. Here, we want to study the impact each mask strategy and the scale of the dataset have on the raw attention maps of each model. We visualize attention maps derived from the last layer of the models. These maps are evaluated qualitatively for randomly picked image samples derived from the ImageNet validation set. We then use the 1000 images ImageNet validation subset to evaluate the attention maps of the models quantitatively with the average decrease (AD), average increase (AI), deletion (D) and insertion (I) metrics.

From a quick look in Table 16, we observe that changing the masking strategy does not have a great impact on the attention maps and therefore all the maps look similar. On the other hand, the scale of the dataset has an influence on the map. It seems that when the model is trained on the 20% ImageNet, we observe spread out salient regions mainly on the object with a limited percentage of salient regions coming from the background. When the model is trained on the 100% ImageNet, we find small salient regions on specific features of the object.

The aforementioned observations are expressed quantitatively in Table 17. Here, we see again the low impact the change of the masking strategy has. It is hard to say, but it

seems that AttMask-Raw$_{5,12}$ has the best metrics when the model is trained on the 20% ImageNet and AttMask achieves the best when the model is trained on the full dataset. The influence of the scale of the dataset is expressed quantitatively mainly with the AD, D and I metrics. When the models are trained on the 100% ImageNet, we observe a clear improvement in these three metrics. The fact that the models reach a bit better AI results when trained on the 20% ImageNet, is something we didn't expect.

Table 16: Raw attention maps obtained from the last layer of iBOT when the model is trained with three different masking strategies on the 20% and 100% ImageNet.

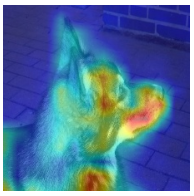
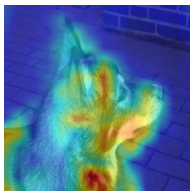
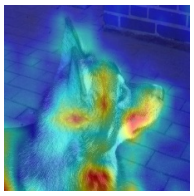
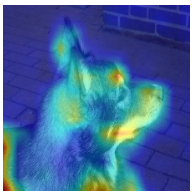
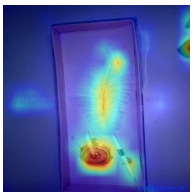
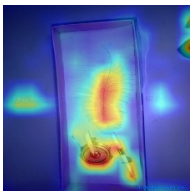
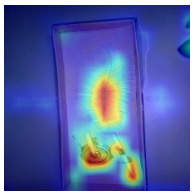
Sample	% ImageNet-1k	iBOT Masking		
		AttMask [10]	AttMask-Pow(0.5)	AttMask-Raw$_{5,12}$
 malamute	20			
	100			
 quill	20			
	100			
 teapot	20			
	100			

Table 17: Quantitative evaluation of raw attention maps, derived from iBOT models trained with three different masking strategies on the 20% and 100% ImageNet. Classification metrics on 1000 randomly picked images from the ImageNet validation set. AD/AI: average drop/increase I/D: insertion/deletion ↓ / ↑: lower / higher is better .

iBOT Masking	Metrics for models pre-trained on 20% and 100% ImageNet							
	AD↓		AI↑		D↓		I↑	
	20%	100%	20%	100%	20%	100%	20%	100%
AttMask [10]	0.61	**0.46**	**0.17**	**0.17**	0.07	**0.11**	**0.30**	**0.55**
AttMask-Pow(0.5)	0.61	0.49	**0.17**	0.15	**0.06**	0.10	**0.30**	0.54
AttMask-Raw$_{5,12}$	**0.58**	0.47	**0.17**	0.16	**0.06**	**0.11**	**0.30**	0.54

4.4.3 Contrastive learning

In these experiments, we introduce an alternative loss for training the iBOT [9] models. The proposed loss is a result of the standard iBOT loss plus an extra contrastive term, as detailed described in section 4.1. Here, we compare the k-NN and linear probing scores of the models when trained with the two different losses. We test both losses for models trained with AttMask [10], AttMask-Pow(0.5) and AttMask-Raw$_{5,12}$. All the models are pre-trained on the 20% ImageNet for 100 epochs.

The first experiments we conduct were on the iBOT model which followed the AttMask strategy. After a hyperparameter tuning for $k = 3, 6, 12$, for $\tau = 0.2, 0.14, 0.07, 0.03$ and $\lambda_3 = 0.0001, 0.01, 0.1, 0.5, 1, 5, 10$ we reach the top scores when $k = 6, \tau = 0.07, \lambda_3 = 1$. We therefore keep the best hyperparameters for AttMask in the experiments with the rest of the masking strategies. In Table 18, we present the results for the three strategies and from a quick look, there is no clear benefit of using the loss with the contrastive term. When using AttMask with our loss, we observe a bit better k-NN and equally good linear probing scores. Combining AttMask-Pow(0.5) with our loss gives equally good k-NN and a bit better linear probing scores. Using AttMask-Raw$_{5,12}$ with our loss has a bad influence on k-NN and a good one on linear probing scores. The usage of our loss comes with an extra $\approx 11\%$ of the experiment time with iBOT loss.

Table 18: Evaluation of two iBOT losses with k-NN and linear probing. The models are trained for 100 epochs with three different masking strategies on the 20% ImageNet and evaluated on the whole validation set of the dataset.

iBOT Masking	Loss	Evaluation	
		k-NN	Linear
AttMask [10]	iBOT Loss	48.9	**57.1**
	Our Loss	**49.1**	**57.1**
AttMask-Pow(0.5)	iBOT Loss	**49.3**	57.5
	Our Loss	**49.3**	**57.7**
AttMask-Raw$_{5,12}$	iBOT Loss	**49.3**	57.6
	Our Loss	49.1	**57.7**

4.5 Conclusion

In chapter 4, we focus on three different aspects of MIM self-supervision. In the first one, we study masking strategies used in the learning process and propose new ones to further improve accuracy. Then, we see the impact masking strategies and the scale of

a dataset have on interpretability. Finally, we introduce a new loss and show its influence on accuracy.

The effect of masking strategies on accuracy Here, we present and evaluate existing state-of-the-art masking strategies and propose new ones that perform even better. We further provide the reasons why a strategy achieves better results than another. First and foremost, we accept that attention-guided token masking is more beneficial than random masking. But we argue with AttMask [10] approach that claims mask generation from attention maps derived from the last layer of the model provides the best results. This approach provides masks that hide the foreground of an object in an image. We prove that generating masks that hide mainly the foreground but also a percentage of context information coming from the background can further improve evaluation scores and acceleration of the learning process in downstream tasks. We show this by improving the k-NN and linear probing scores of AttMask baseline when using our AttMask-Raw$_5$, AttMask-Pow, AttMask-Raw$_{5,12}$, AttMask-Raw$_{3,5,12}$ and deteriorating the baseline results when using our AttMask-Rollout, AttMask-Rollout$_{9-12}$. On the other hand, we show that hiding too much context information and not enough object information has a bad influence on learning and we prove it through AttMask-Raw$_2$, AttMask-Rollout$_{1-4}$, AttMask-Log evaluation results.

Another important insight of this study is that masking each global crop with a different mask further improves the results. The superiority of this multi-layer strategy is shown when compared with AttMask [10] and iBOT [9] baselines. To be more specific, we propose a new masking strategy we call AttMask-Raw$_{5,12}$ that generates a mask for the first global crop from Raw$_5$ and a mask for the second crop from Raw$_{12}$. This masking strategy achieves better results than the two baselines. In the next experiment, we add an extra global crop to all strategies and compare our multi-layer strategy with the previous baselines. Here we propose AttMask-Raw$_{3,5,12}$, a new masking strategy that generates masks from Raw$_3$, Raw$_5$ and Raw$_{12}$. This strategy proves again its superiority over iBOT and AttMask in k-NN and linear probing evaluation metrics.

The efficiency test of the masking strategies is being made on different scales of a dataset, on different numbers of pre-training epochs and on different numbers of global crops. It is worth mentioning, that as the values of the aforementioned parameters increase, we obtain better results for all masking strategies, but the gap between them is progressively narrowing. In other words, our proposed strategies do improve their results as the scale of the dataset, the number of epochs and global crops increase, but the gain over using other baseline strategies decreases. This means, that what to mask is not as important as before when the network is trained for too many epochs with more global crops on a greater scale of a dataset.

The effect of masking strategies and scale of a dataset on interpretability The second aspect of MIM self-supervision we study in chapter 4, is the impact each masking strategy and the scale of a dataset have on interpretability. Here, we observe that the influence of the scale of a dataset in comparison with the kind of masking is greater. In essence, when changing the masking strategy, we see a low impact on the saliency map. On the contrary, when the network is trained on a greater scale of the dataset, we observe smaller well-localized on the object saliency maps than when trained on smaller scales. This is quantitatively expressed with a better average decrease (AD), deletion (D) and insertion (I) metrics. At this stage of the work, we further investigate how the saliency map

progressively changes during the training procedure. We clearly show, that at the beginning of the training procedure the model is not able to recognize the important regions of an image. As the training continues, we prove that the model progressively starts to learn to focus more on the object and less on the background of an image.

The effect of new loss on accuracy In the last aspect of the MIM self-supervision study, we introduce an alternative loss for training the iBOT model. This loss is contrastive, is based on (2.11) and is used as an extra term in the standard iBOT loss. We propose this loss to help the model better discriminate between the foreground and background of an image with the objective to further improve accuracy scores. The new loss is tested with different masking strategies. For some masking strategies, we come up with some improvements in the results compared to the standard iBOT loss. The improvement comes with an extra $\approx 11\%$ of the experiment time with iBOT loss. Here, we think there is room for improvement and in section 5.2 we propose some possible ways to achieve it.

5. OVERALL CONCLUSION AND FUTURE WORK

In section 3.5 and in section 4.5, we provide you with a detailed description of our conclusions in this work. In this chapter, we want to present an overall conclusion we end up with and to show the reasons chapter 3 and chapter 4 are closely related. Then, we suggest some future work on both main chapters of this study.

5.1 Conclusion

This study is partially dedicated to the visual explanation of supervised and self-supervised learning and the investigation and improvement of a MIM-based self-supervised approach. Here, we want to highlight one more time the importance of interpretability and show how the two parts of our study are closely related. In this work we use interpretability insights of chapter 3 to further explore and improve MIM-based self-supervision. To be more specific, from chapter 3, other than the valuable information about supervised and self-supervised models and all the methods we use and propose to interpret them, we show that all the models in order to be confident for their predictions need to take mostly information from the foreground of an object in one image. Here, we discovered that we can boost more the confidence of a model if we reveal some extra context information from the background. This was the knowledge we took from the interpretability of the models, to further improve MIM-based self-supervised learning. In chapter 4, we studied many different masking strategies and we come το the conclusion that masking should be challenging, so we started to hide from the student network the regions of an image that make a model mostly confident before its prediction. Based on this, we proposed various kinds of masking strategies that proved to be beneficial to the MIM-based self-supervised approach.

5.2 Future work

Concerning future work and future directions in the interpretability domain we propose:

1. *Lack of a common visualization method for both CNNs and transformers:* In the experiments in subsection 3.4.1, we see that although we use CAM-based methods for the interpretability of the transformers as well, the results are not acceptable. Given an input image, the only attention-based method that is dependent with a certain class of the image is TIBAV. This method follows a strategy introduced for the interpretability of CNNs as well. TIBAV generates reasonable attribution maps, but their qualitative and quantitative results are not as good as those generated by CNNs. Consequently, class-specific visualization tools for transformers need to be further investigated. First and foremost, to achieve this, understanding better the inner mechanism of both CNNs and transformers and find some common properties between their architectures is necessary. Then, some of the strategies that have already been followed to make CNNs interpretable could be tested on transformers. An alternative future direction is to use TIBAV as baseline to further improve the method.
2. *A new method for the interpretability of transformers:* In subsection 3.4.1, we see that CAM-based methods suit well with CNNs. In the case of transformers, after long experimentation, we are not sure which method is optimal. Each method has

its strengths and weaknesses. Existing visualization methods for transformers take advantage of information coming from attention weights while some methods also follow strategies used in the interpretability of CNNs. Transformers are complicated and information is shared across multiple linear projections, such as queries, keys and values. Our experiments reveal that projections of keys can provide reasonable information. Here, an extension could be a new visualization method that benefits from other linear projections as well.

3. *More representative metrics:* We believe that the best way to interpret the prediction of a model is the qualitative evaluation given the whole dataset. But this is not possible, having in mind the scale of the datasets. This is the reason that we need representative metrics. In our study, we use four classification metrics. We have seen that only when all the metrics are good, the saliency map is reasonable as well. This means that the metrics are not good representatives of the saliency map when used separately. So the more metrics we use, the better the interpretability of models. On the other hand, using infinite number of metrics is not practical. We see two future directions for this challenge. Introducing a few extra metrics that measure other properties of the saliency map is one option. Finding some flaws in the existing ones and improving them would be another direction.

Concerning future work in the MIM-based self-supervised learning process we propose:

1. *Fairer comparison of MIM-based methods:* Although AttMask [10] approach and our methods are attention-guided masking strategies and not random block-wise as in iBOT [9], there is a lot of randomnesses in all methodologies. This yields a variance in the results when running the same experiment more than once. As a reminder, in the three approaches, each image is masked with probability $p = 0.5$. Then, from the total n patch tokens only $k = \lfloor rn \rfloor$ of them will be masked. The mask ratio r is sampled uniformly per image as $r \sim U(a, b)$ with $[a, b] = [0.1, 0.5]$. If we keep r fixed and execute the experiments for all the approaches, we think this could be a fairer comparison and the impact of the mask on the learning process would be clearer for sure. Finally, if we set $p = 1$, this could give further insights into the mask impact. Probably, the randomness the parameters p and r provide to the learning process boosts the accuracy scores. However, reducing the randomness may enable you to better understand the impact of your modification on the network at least during an ablation study. After the study, randomness could be reintroduced, if it has an positive impact on the accuracy.
2. *Further investigation on contrastive learning:*In section 4.1, we introduce a new loss term to the iBOT approach with the objective to further improve accuracy results. In particular, this is a dense contrastive term which helps the model better discriminate the foreground of an object from the background. The results, although promising, were not as good as we expected, but we believe this idea could be reimplemented in different ways. One way could be to use another type of contrastive loss function.

ABBREVIATIONS - ACRONYMS

DNN	Deep Neural Networks
CV	Computer Vision
CNN	Convolutional Neural Network
CAM	Class Activation Maps
MIM	Mask Image Modeling
AI	Artificial Intelligence
ANN	Artificial Neural Networks
ML	Machine Learning
DL	Deep Learning
FNN	Feedforward Neural Network
ReLU	Rectified Linear Unit
LSTM	Long Short Term Memory Networks
RNN	Recurrent Neural Networks
GAN	Generative Adversarial Networks
MLP	Multilayer Perceptrons
ResNet	Residual Neural Network
XCA	Cross-Covariance Attention
LPI	Local Patch Interaction
MoCo	Momentum Contrast
AttMask	Attention-Guided Masked Image Modeling
GAP	Global Average Pooling
TIBAV	Transformer Interpretability Beyond Attention Visualization
AD	Average Drop
AI	Average Increase
D	Deletion
I	Insertion
AUC	Area Under the Curve
k-NN	k Nearest Neighbours

REFERENCES

[1] Y. Mahdid, "Perceptron algorithm from scratch in python," 2020. https://yacinemahdid.com/static/7c425dc2a439bb4bcc6e627eb549b010/6c315/thumbnail.jpg.

[2] A. Mohanty, "Multi layer perceptron (mlp) models on real world banking data," 2019. https://miro.medium.com/max/700/1*-IPQlOd46dIsutIbUq1Zcw.png.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2020.

[5] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers and distillation through attention," 2020.

[6] A. El-Nouby, H. Touvron, M. Caron, P. Bojanowski, M. Douze, A. Joulin, I. Laptev, N. Neverova, G. Synnaeve, J. Verbeek, and H. Jegou, "Xcit: Cross-covariance image transformers," 2021.

[7] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," 2019.

[8] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," 2021.

[9] J. Zhou, C. Wei, H. Wang, W. Shen, C. Xie, A. Yuille, and T. Kong, "ibot: Image bert pre-training with online tokenizer," 2021.

[10] I. Kakogeorgiou, S. Gidaris, B. Psomas, Y. Avrithis, A. Bursuc, K. Karantzalos, and N. Komodakis, "What to hide from your students: Attention-guided masked image modeling," 2022.

[11] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[13] V. Petsiuk, A. Das, and K. Saenko, "Rise: Randomized input sampling for explanation of black-box models," 2018.

[14] Z. Xie, Z. Zhang, Y. Cao, Y. Lin, J. Bao, Z. Yao, Q. Dai, and H. Hu, "Simmim: A simple framework for masked image modeling," 2021.

[15] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," 2021.

[16] H. Bao, L. Dong, S. Piao, and F. Wei, "Beit: Bert pre-training of image transformers," 2021.

[17] Z. Li, Z. Chen, F. Yang, W. Li, Y. Zhu, C. Zhao, R. Deng, L. Wu, R. Zhao, M. Tang, and J. Wang, "Mst: Masked self-supervised transformer for visual representation," 2021.

[18] S. Abnar and W. H. Zuidema, "Quantifying attention flow in transformers," *CoRR*, vol. abs/2005.00928, 2020.

[19] H. Chefer, S. Gur, and L. Wolf, "Transformer interpretability beyond attention visualization," *CoRR*, vol. abs/2012.09838, 2020.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 1980.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[23] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.

[24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016.

[25] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 448–456, PMLR, 07–09 Jul 2015.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[27] H. Hu, J. Gu, Z. Zhang, J. Dai, and Y. Wei, "Relation networks for object detection," 2017.

[28] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," *CoRR*, vol. abs/2005.12872, 2020.

[29] A. Srinivas, T.-Y. Lin, N. Parmar, J. Shlens, P. Abbeel, and A. Vaswani, "Bottleneck transformers for visual recognition," 2021.

[30] B. Wu, C. Xu, X. Dai, A. Wan, P. Zhang, Z. Yan, M. Tomizuka, J. Gonzalez, K. Keutzer, and P. Vajda, "Visual transformers: Token-based image representation and processing for computer vision," 2020.

[31] P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens, "Stand-alone self-attention in vision models," 2019.

[32] H. Wang, Y. Zhu, B. Green, H. Adam, A. Yuille, and L.-C. Chen, "Axial-deeplab: Stand-alone axial-attention for panoptic segmentation," 2020.

[33] T. Ridnik, E. Ben-Baruch, A. Noy, and L. Zelnik-Manor, "Imagenet-21k pretraining for the masses," 2021.

[34] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," 2017.

[35] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-100 (canadian institute for advanced research),"

[36] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, (New York, NY, USA), p. 1096–1103, Association for Computing Machinery, 2008.

[37] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013.

[38] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models."

[39] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[40] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, "Adversarially learned inference," 2016.

[41] J. Donahue and K. Simonyan, "Large scale adversarial representation learning," 2019.

[42] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," 2018.

[43] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial feature learning," 2016.

[44] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," 2015.

[45] C. Doersch and A. Zisserman, "Multi-task self-supervised visual learning," 2017.

[46] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," 2016.

[47] G. Larsson, M. Maire, and G. Shakhnarovich, "Learning representations for automatic colorization," 2016.

[48] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," 2016.

[49] M. Noroozi and P. Favaro, “Unsupervised learning of visual representations by solving jigsaw puzzles,” 2016.

[50] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, “Photo-realistic single image super-resolution using a generative adversarial network,” 2016.

[51] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox, “Discriminative unsupervised feature learning with convolutional neural networks,” in *Advances in Neural Information Processing Systems* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, eds.), vol. 27, Curran Associates, Inc., 2014.

[52] S. Gidaris, P. Singh, and N. Komodakis, “Unsupervised representation learning by predicting image rotations,” 2018.

[53] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A simple framework for contrastive learning of visual representations,” 2020.

[54] A. v. d. Oord, Y. Li, and O. Vinyals, “Representation learning with contrastive predictive coding.”

[55] Y. Tian, D. Krishnan, and P. Isola, “Contrastive multiview coding,” 2019.

[56] Y. Tian, C. Sun, B. Poole, D. Krishnan, C. Schmid, and P. Isola, “What makes for good views for contrastive learning?,” 2020.

[57] S. Arora, H. Khandeparkar, M. Khodak, O. Plevrakis, and N. Saunshi, “A theoretical analysis of contrastive unsupervised representation learning,” 2019.

[58] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krähenbühl, “Sampling matters in deep embedding learning,” 2017.

[59] B. Harwood, V. K. B. G, G. Carneiro, I. Reid, and T. Drummond, “Smart mining for deep metric learning,” 2017.

[60] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko, “Bootstrap your own latent: A new approach to self-supervised learning,” 2020.

[61] X. Chen and K. He, “Exploring simple siamese representation learning,” 2020.

[62] A. Tarvainen and H. Valpola, “Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results,” 2017.

[63] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” 2018.

[64] X. Chen, H. Fan, R. Girshick, and K. He, “Improved baselines with momentum contrastive learning,” 2020.

[65] X. Chen, S. Xie, and K. He, “An empirical study of training self-supervised vision transformers,” 2021.

[66] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, “Unsupervised learning of visual features by contrasting cluster assignments,” 2020.

[67] F. Doshi-Velez and B. Kim, “Towards a rigorous science of interpretable machine learning,” 2017.

[68] Z. C. Lipton, “The mythos of model interpretability,” 2016.

[69] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, D. Pedreschi, and F. Giannotti, “A survey of methods for explaining black box models,” 2018.

[70] Y. Zhang, P. Tino, A. Leonardis, and K. Tang, “A survey on neural network interpretability,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 5, pp. 726–742, oct 2021.

[71] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, “Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization,” *CoRR*, vol. abs/1610.02391, 2016.

[72] A. Chattopadhyay, A. Sarkar, P. Howlader, and V. N. Balasubramanian, “Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks,” *CoRR*, vol. abs/1710.11063, 2017.

[73] R. Fu, Q. Hu, X. Dong, Y. Guo, Y. Gao, and B. Li, “Axiom-based grad-cam: Towards accurate visualization and explanation of cnns,” *CoRR*, vol. abs/2008.02312, 2020.

[74] H. Wang, M. Du, F. Yang, and Z. Zhang, "Score-cam: Improved visual explanations via score-weighted class activation mapping," *CoRR*, vol. abs/1910.01279, 2019.

[75] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," *CoRR*, vol. abs/2104.14294, 2021.

[76] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[77] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K. Müller, "Explaining nonlinear classification decisions with deep taylor decomposition," *Pattern Recognit.*, vol. 65, pp. 211–222, 2017.

[78] J. Gildenblat and contributors, "Pytorch library for cam methods." `https://github.com/jacobgil/pytorch-grad-cam`, 2021.

[79] J. Choe, S. J. Oh, S. Lee, S. Chun, Z. Akata, and H. Shim, "Evaluating weakly supervised object localization methods right," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. to appear.

[80] J. Zhang, Z. Lin, J. Brandt, X. Shen, and S. Sclaroff, "Top-down neural attention by excitation backprop," 2016.

[81] P. Dabkowski and Y. Gal, "Real time image saliency for black box classifiers," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.