Joana Luís Martins

Master of Science

Interpretability of deep neural networks at the model level

Dissertation submitted in partial fulfillment of the requirements for the degree of

Master of Science in
Big Data Analysis and Engineering

Adviser:	Ludwig Krippahl, Assistant Professor, NOVA University of Lisbon

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

November, 2021

Interpretability of deep neural networks at the model level

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

This thesis presented several challenges to me, but it was also a great learning opportunity. I am therefore indebted to my supervisor, Professor Ludwig Krippahl for suggesting this topic, and also for his patience in the many discussions during the work. Most of all, I would like to express my gratitude for helping me navigate the project, through his advices and guidance, and to thank him for providing me the computational resources for a significant part of the work presented in this thesis.

I would also like to thank Lucas Fischer, Nitin Shukla, Elisabetta Boella, Rita Gervásio, Thales Silva, for their moral support and friendship.

My greatest supporters throughout this thesis, as well as in other challenges in my life, have been my family. To my parents, and to my brother and my sister-in-law, I am deeply grateful for their constant supply of hope and encouragement.

Abstract

Deep neural networks (DNN) are very powerful tools but remain black boxes. Convolutional neural networks (CNN) are a type of DNN specialized for certain tasks like image classification, image segmentation and object detection, among others. These are the focus of this study.

Because of the risks involved in applying these networks to areas such as medical imaging and autonomous driving, it is important to understand the behaviour of CNNs, in particular, how they reach their predictions. This will also be beneficial for their further development.

Currently, most interpretability techniques are aimed at providing better understanding of a single network instance and often explain only the prediction of a given individual input example. However, for the same model (same architecture, task and dataset), one can have many different hypotheses, simply by changing the initial conditions (weights initialization). These sources of variability, between examples, for the same hypothesis (model instance), and between hypothesis for the same example, are not taken into account by the vast majority of current interpretability methods. This opens the question whether there is consistency between hypothesis in the attributes the interpretability diagnostics capture as being relevant for the predictions. In this work, tools and methods were developed to analyse these two forms of variability. They were applied to two interpretability diagnostics (saliency maps and sensitivity to occlusion) for several models, with different architectures and tasks. Furthermore, it was shown how they can be used to identify potential problems in the relevance of the attributes the interpretability diagnostics capture in some of the scenarios studied. The method also provides a means to assess possible strategies for mitigating this issue.

Keywords: deep neural networks, convolutional neural networks, interpretability techniques

Resumo

As redes neuronais profundas são ferramentas muito poderosas, mas permanecem caixas negras. As redes convolucionais são um tipo de redes neuronais profundas especialmente adaptado a tarefas como a classificação de imagens, a segmentação de imagens e a detecção de objectos, entre outras. Estas são o foco deste estudo.

Devido aos riscos associados à aplicação destas redes a áreas como a imagiologia médica e a condução autónoma, é importante compreender o comportamento das redes convolucionais, em particular, como chegam às suas previsões. Esta compreensão também será benéfica para o seu crescente desenvolvimento.

Actualmente, a maioria das técnicas de interpretabilidade estão vocacionadas para possibilitar uma maior compreensão de uma instância de rede em particular, e frequentemente oferecem uma explicação para um exemplo em particular tirado do conjunto de dados inicial. No entanto, para o mesmo modelo (mesma arquitectura, tarefa e conjunto de dados), podem ter-se muitas hipóteses diferentes, pela simples mudança das condições iniciais (inicialização dos pesos). Estas fontes de variabilidade, entre exemplos, para a mesma hipótese (instância do modelo), e entre hipóteses para o mesmo exemplo, não são tidas em conta na grande maioria das técnicas de interpretabilidade actuais. Isto abre uma questão sobre se há consistência entre hipóteses quanto aos atributos que os diagnósticos de interpretabilidade capturam como sendo relevantes para as previsões. Neste trabalho, desenvolveram-se ferramentas e métodos para analisar estas duas formas de variabilidade. Estas foram aplicadas a dois diagnósticos de interpretabilidade (mapas de saliência e sensibilidade à oclusão) para vários modelos, com diferentes arquitecturas e tarefas. Adicionalmente, mostrou-se como podem ser usadas para identificar potenciais problemas quanto à relevância dos atributos capturados pelos diagnósticos de interpretabilidade em alguns dos cenários estudados. Este método proporciona também uma forma de avaliar possíveis estratégias para mitigar o problema.

Palavras-chave: Palavras-chave (em Português) redes neuronais profundas, redes neuronais convolucionais, técnicas de interpretabilidade

Contents

List of Figures

Acronyms

AI	artificial intelligence.
CNN	Convolutional neural network.
DNN	Deep neural network.
FNN	Feedforward neural network.
GAN	Generative adversarial network.
ISOMAP	Isometric feature mapping.
LASSO	least absolute shrinkage and selection operator.
LIME	Local Interpretable Model-agnostic Explanations.
ML	machine learning.
MLP	Multi-layer perceptron.
PCA	Principal component analysis.
ReLU	Rectified linear unit.
VAE	Variational autoencoders.

Chapter 1

Introduction

Deep neural networks are a subfield of artificial intelligence (AI) which has produced powerful tools for diverse tasks such as image and speech recognition, automatic translation, among many others. Despite their capabilities, deep neural networks are considered to be black-boxes since an understanding of how they reach their (sometimes surprising) results is limited by their inherent lack of transparency. Unlike traditional machine learning algorithms, they are not provided with engineered features as input for a specific task. They apply nonlinear transformations to input data, producing new representations of them that are then used as features to compute an output. This nonlinearity and their complexity hinder our understanding of their learning process. As interpretability methods from machine learning are not directly applicable or are unsuited to neural networks, there is a growing interest in the field of interpretability/explainability of the latter.

The present work will survey deep neural networks interpretability techniques applicable to problems which involve non-trivial tasks with images, such as object detection or image classification. Based on this review, we will explore how to adapt these methods to deep neural networks models instead of specific network instances.

In the following, some context and background knowledge of this problem are presented. In section 1.1, we further motivate the need for employing current interpretability methods and for developing new ones. This is followed by section 1.2, where a review is made of the basics of deep neural networks: their architecture, training/learning processes, and some examples of specialized networks are presented. Finally, in section 1.3, the expected contribution of the present work to the field is summarised.

1.1 Motivation and problem statement

The evolution of artificial intelligence (AI) has enabled a growing body of applications. Neural networks in particular are currently employed or under study in areas as diverse as medicine, in automated diagnostic techniques, for instance through image processing (e.g. [1]); in finance, in areas such as the evaluation of mortgage risk [2]; in autonomous driving (e.g. [3, 4]); in judicial systems, in the assessment of the risk of recidivism in crime [5]; in image recognition (e.g. [6]) and speech recognition [7]; in machine translation [8]; etc.

This rapid evolution in the applications of neural networks has followed the growth in complexity of the algorithms employed, frequently at the expense of their transparency and interpretability, leading to a trade-off between model performance and interpretability [9].

The increased domain of neural networks applicability has also raised questions concerning the fairness [10], transparency, safety, privacy (of data collection) and ethical and legal implications of autonomous decision algorithms, given that they can have serious consequences in the lives of those that are the object of the decisions made [11]. In fact, there are already examples where the use of machine learning/deep learning has lead to controversial or biased outcomes (whether due to dataset or algorithm bias). Questions were raised by a study indicating that COMPAS, a software to assess the risk of criminal recidivism to aid parole and sentencing decisions, was racially biased and had weak predictive accuracy [12]. In another example, Amazon is said to have abandoned a project for automated analysis of job candidates when it uncovered that women were unfairly penalized [13] (see [14] for an in-depth review of bias in machine learning).

These concerns have led to research on bias in machine learning [10, 14] and, on the other hand, to the inclusion in the EU General Data Protection Regulation (GDPR) of several rules about the collection and use of personal data for automated decision-making. In particular, there is an obligation to provide the data subject with "meaningful information about the logic involved, as well as the significance and the envisaged consequences of such processing" for him [11, 15]. On the other hand, application of DNN in critical situations (such as medical applications and autonomous driving) also points to the need for improved interpretability of the algorithms, for software auditing and model validation.

Improving the interpretability of deep neural networks is also important from a fundamental research point of view. An increased understanding of their learning process could better guide the development of networks, ensuring they are not learning artefacts or are vulnerable to adversarial attacks. A telling example of such attacks was the discovery that by doing certain changes in an image, imperceptible to the human viewer, the correct results of an image classification network would change to an incorrect prediction [16].

Current efforts in the domain of neural network interpretability have focused mainly on interpreting trained instances of these networks (hypotheses) or the network response to a particular example or group of examples. However, it is not clear whether the insight gained with these methods extends to models. If, for example, one trains multiple instances of the same model, will the attributes captured by the interpretability methods regarding what is valued by the network to make a prediction, or about its functioning, be consistent among the instances?

It is important to assess the expected behaviour of a model, since, in principle, one will only use an instance of it in practice but would like to have an understanding of whether this will differ significantly if another instance were to be used.

1.2 Background Knowledge

1.2.1 Deep Neural Networks

Deep learning algorithms are similar to other machine learning algorithms in the sense that they are also composed of a dataset, a cost function, an optimization procedure and a model. However, unlike other machine learning algorithms, who learn the mapping from the features to the predictions, these do not need to be provided to neural networks. Deep learning algorithms take raw data (pictures, sequences of words,...) and learn the features relevant to the task during training, i.e. new representations of the data that are more useful for the desired purpose. Here resides their power.

Structure-wise, artificial neural networks are composed of a multitude of units (artificial neurons) organized in layers, connected between each other.

These units, or artificial neurons, were inspired by the way synapses occur in neurons in the brain. Stimuli collected from dendrites in the neuron are combined and only if they exceed some threshold to depolarize the membrane, will the stimulus propagate through the axon of the neuron until its synaptic terminals. Current artificial neural networks and neurons are no longer as close to the biological structures which inspired them as in the early days of artificial intelligence, but some similarities remain. The functioning of a generic artificial neuron is depicted in figure 1.1(a). The m inputs from an example $\boldsymbol{x}$ of the dataset (x_1 to x_m) are fed into the neuron, in the form of a weighted sum, with weight values indexed from ω_{k1} to ω_{km}. An additional bias term b is added to the previous sum. The first part of the computation yields the input signal of the neuron:

$$v_k = \sum_{j=1}^{m} x_j \omega_{kj} + b. \tag{1.1}$$

This affine transformation can also be expressed through the inner product of an input vector $\boldsymbol{x}^T = [1, x_1, x_2, ..., x_m]$ with a vector of weights $\boldsymbol{\omega}^T = [b, \omega_{k1}, \omega_{k2}, ..., \omega_{km}]$, $\boldsymbol{\omega}^T \cdot \boldsymbol{x}$, where the bias term was included through the addition of the extra term with fixed input value $x_0 = 1$ and the weight $\omega_{k0} = b$. The result of this weighted sum will be fed into the activation function as input.

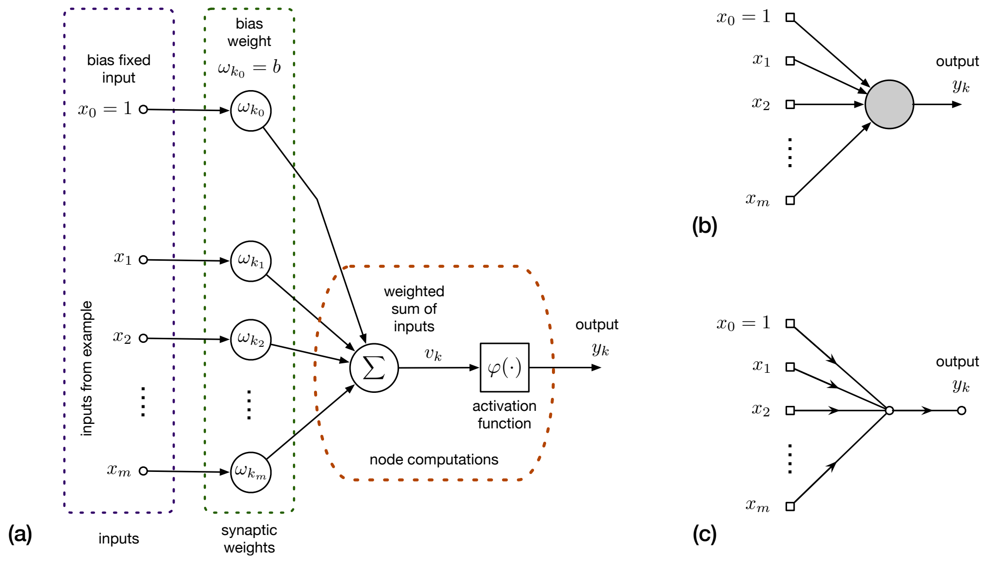


Figure 1.1: Artificial neuron in (a) block diagram (functional representation), (b) architectural diagram and in (c) signal flow graph. Adapted from [17].

In the early perceptron model, the Heaviside function was used as a threshold activation function (Figure (1.2(a))). As a result, even using more than one layer of neurons, the output would still be a linear combination of the inputs, unable to classify classes of examples which were not linearly separable correctly. That function was therefore replaced by nonlinear activation functions, such as those depicted in Figure (1.2(b)-(d)). In fact, the ReLU activation function, given by $\max(0, x)$, whose only source of nonlinearity is the discontinuity at $x = 0$ was enough to transform the inputs of neural networks into representations in a different space where they became linearly separable. The linearity of the ReLU for positive inputs also makes the calculation of gradients very easy, which, as will be seen ahead, are a part of the training process.

Nowadays, several different nonlinear functions are used for different purposes in neural networks, among which: sigmoid activations like the logistic function $\varphi(v) = 1/[1+\exp(-av)]$, where a is a parameter that controls the slope; and the hyperbolic tangent $\varphi(v) = \tanh v$ (see Figure (1.2(b) and (c))). The specific choice of nonlinear function depends on the desired task as will be described later on.

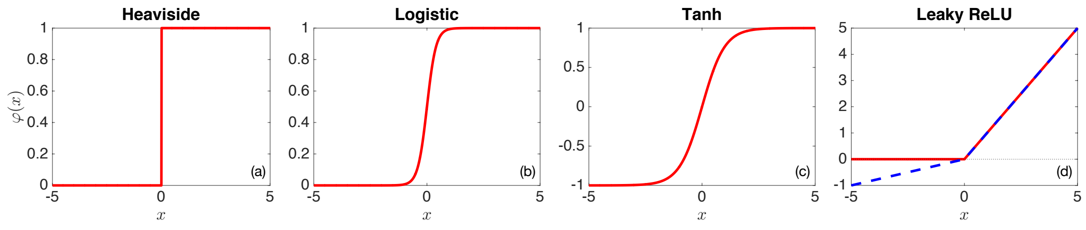


Figure 1.2: Typical activation functions.

While the hidden layer units are usually of the same kind (e.g. ReLU), output units are specifically chosen for the task to be performed. Different tasks correspond to different probability distributions we wish to learn from the data, and this is reflected in the determination of the error or loss function.

In neural networks, the loss function is typically obtained using conditional maximum likelihood estimation, which is equivalent to minimizing the Kullback–Leibler divergence or cross-entropy. In this way, in minimizing the loss, we are trying to make our model probability distribution as close (p_{model}) as close to the distribution we are extracting our samples from (p_{data}). The loss obtained from maximum likelihood estimation conditioned on our network parameters $\boldsymbol{\theta}$ can be expressed as:

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{data}} \log p_{model}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}). \tag{1.2}$$

Here $\boldsymbol{\theta}$ includes both the weights and biases to be learned. Loss functions typically contain additional terms for regularization purposes, such as weight decay, where a penalty term of the form $\lambda||\boldsymbol{\omega}||^2$ is added to prevent parameters from reaching very high values.

Linear activation functions with a loss function whose conditional probability model is a Gaussian function results in obtaining the mean of the distribution in the output. Since in linear regression it is assumed that the targets are normally distributed, the choice of a linear activation function is suitable for this task.

For binary classifiers, the output distribution should be of the Bernoulli type, since this models the probability of an event occurring, given two possible outcomes. In our model, we wish to have $P(y = 1|\boldsymbol{x})$, to define our loss function; all we need is a function that can give us this probability. A transformation given by the sigmoid function, which saturates at the values of 0 and 1 is appropriate since its range is $[0,1]$ as in probability density functions.

For multiclass classification, the goal is to identify the input example as pertaining to one of several classes. The distribution we expect to uncover is of the Bernoulli type since it models the probability of being in one of m states. The softmax units output a vector of scalars in $[0,1]^m$, where m is the number of classes, and whose sum is one. They are therefore suitable for this multi-class classification. Each component of the vector can then be seen as a probability of belonging to a given class k. The predicted class can then be extracted simply as the class for which the probability value was maximum.

Deep learning networks have multiple layers of neurons connected to each other in a hierarchical way. The most simple examples of such networks are the multilayer perceptrons (MLPs), also called feedforward neural networks (FNN). In feedforward networks, as the name indicates, information flows in a single direction, from the input to the output, without any feedback from the output introduced into the network again. Figure 1.3 depicts one of such networks. At the bottom is the input layer, where the examples from the external dataset are fed into the network. The output layer, at the top of the network, returns the prediction for the task at hand. The layers between them are called hidden layers since they cannot access the output directly. They are responsible for learning the best representation of the data in the dataset examples to map the input to the output in supervised learning.

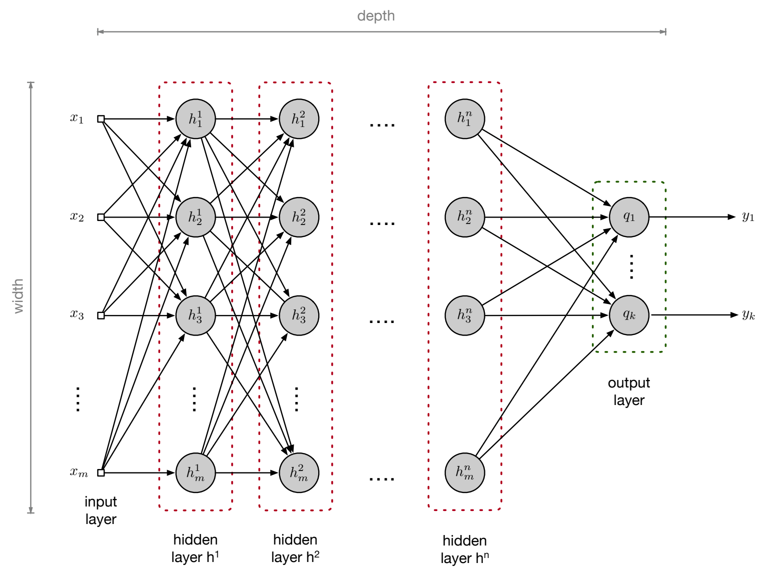


Figure 1.3: Neural network with m inputs, n hidden layers with m neurons and 1 output layer with k output values. The architecture of this network is abbreviated by $m - h_1 - h_2 - ... - h_n - q_k$. Adapted from [17].

Because the neurons of each hidden layer neurons are connected to all the neurons of the subsequent layer, this kind of network is said to be fully connected. When this is not the case, the network is partially connected.

For a general feedforward network, as illustrated in Figure 1.3, this flow of information through the layers can be thought of the application of successive transformations, which can be described as the composition of different vector-to-vector nonlinear functions, each representing a hidden layer.

The high connectivity of these networks and the existence of multiple hidden layers of neurons that introduce non-linearity through the network are behind their capacity to learn more complex problems than a single neuron. As more layers are added, the generalization - the ability to more accurately predict the correct output for previously unseen examples - and the invariance - insensitivity to perturbations - of the network increases [18]. These characteristics, however, pose several problems in terms of both the learning process and the interpretability of the network.

The transformation of the input into features in different representational spaces by the different layers, and the distributed non-linearity associated with it, makes it harder to understand the inner workings of the network and less transparent what it is learning. Even after training has ended and the weights have been set, it is not possible to understand what the network learned without additional tools to visualize/gain insight into the features being extracted and how they evolve during training.

1.2.2 Training Deep Neural Networks

The training of deep neural networks in supervised learning is typically done by minimization of an error or loss function, through gradient descent. The main idea of this type of algorithms is to change the weights iteratively in the opposite direction of the gradient of the error or objective function, relative to the weights, so as to direct it towards lower values:

$$\boldsymbol{\omega}^{(i+1)} = \boldsymbol{\omega}^{(i)} - \eta \nabla_{\boldsymbol{\omega}^{(i)}} \mathscr{E} \tag{1.3}$$

where i represents the iteration, $\mathscr{E}$ the loss function, $\boldsymbol{\omega}^{(i)}$ is the weight vector associated to a neuron and η the learning rate, which determines the size of the step to take when multiplied by the gradient term. In gradient descent methods, the loss function is therefore required to be a differentiable function.

During training, the error or loss function needs to be determined at every iteration in order to update the network parameters. Since the loss function is often of the form of a sum over terms relative to each example, the computational cost of this calculation grows linearly with their number. This can become prohibitive for current datasets, which can reach millions of examples. For this reason, in deep neural networks, stochastic gradient descent is used instead of simple gradient descent. In this case, mini-batches of examples (which can be on the order of hundreds of examples) are randomly extracted from the full training dataset, and the loss function is determined after the passing of each of these batches through the network. The network weights are updated until a sufficiently low value of the loss function has been achieved.

While gradient descent seems easy to implement at the output layer, one then needs a mechanism to distribute the error of the misclassifications to other layers in the network. This is done efficiently by using the back-propagation algorithm.

The back-propagation algorithm works in two steps. In general terms, in a first step, the input signal is propagated from the external input sources to the output layer (i.e. in the forward direction), undergoing multiple transformations as it traverses the different layers. The result of the output layer is then compared to the label for this example, and the cost function is evaluated. In the second step, the error committed is propagated in the backward direction, from the output layer to the first hidden layer and, in the process, the network weights are updated in a manner that decreases the cost function. In the following, this second step is described in detail (following [17]).

To understand how this works, it is useful to separate the output from the hidden layers. Propagating the cost function in the backward direction, the input of the output layer neurons is the cost function value, and it is therefore intuitive to determine the weight change by changing it in the direction opposite to that of the directional derivative of the cost function with respect to the weight of that neuron. We then need to find a way to do the same for the neurons in the hidden layers, which do not have direct access to the cost function. The popularity of the back-propagation algorithm lies in the efficient way in which it is able to update the weights of the hidden layers.

To facilitate our analysis, we first look into what happens in the output layer, where the loss function is calculated at the end of the forward pass. Since we are using a gradient descent method, the first step is to determine how to change the weights of the neurons in the output layer such that the loss function decreases.

The total error or loss function depends on all the outputs of the neurons in the output layer ($\boldsymbol{y}$), i.e., $\mathscr{E}(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{W})$, where $\boldsymbol{W}$ is the matrix of weights and the matrix element ω_{jk} corresponds to the weight that neuron j attributes to the input coming from neuron k (see Figure 1.4). We designate y_j as the output of one of such neurons, which can be expressed as

$$y_j(n) = \varphi_j\left(v_j(n)\right) \tag{1.4}$$

where $\varphi(\cdot)$ is the neuron activation function and v_j is the weighted sum of the inputs (y_k) it receives from the neurons in the previous layer which are connect to it:

$$v_j(n) = \sum_{k=0}^{m} \omega_{jk} y_k(n). \tag{1.5}$$

Note that the inputs from one layer are the outputs of the preceding layer, and therefore represented by y. To update the parameters ω_{jk} we then need to determine the derivative

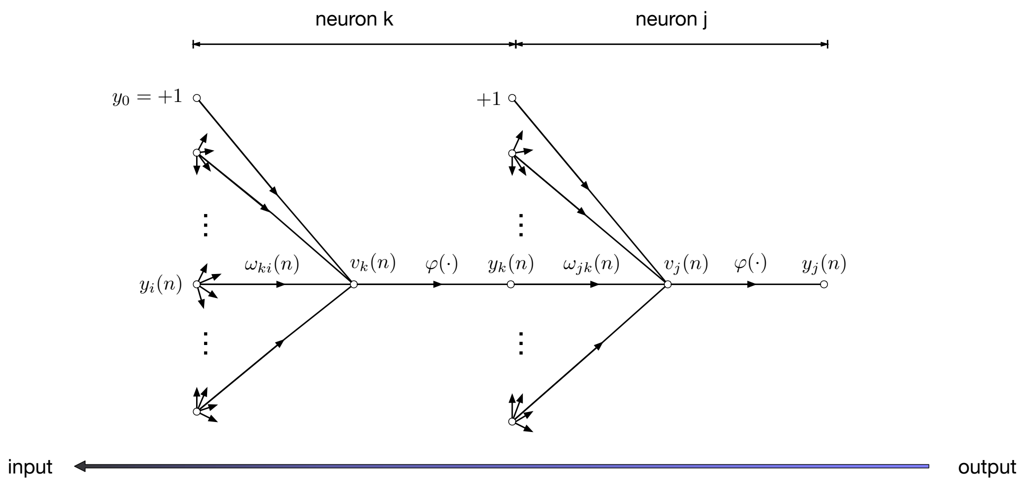


Figure 1.4: Simplified signal flow graph of two connected layers. Adapted from [17].

of the loss with respect to it:

$$\frac{\partial \mathscr{E}(n)}{\partial \omega_{jk}(n)} = \frac{\partial \mathscr{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial \omega_{jk}(n)} \tag{1.6}$$

also called sensitivity factor.

From equations (1.4) and (1.5) we obtain:

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)) \qquad\qquad \frac{\partial v_j(n)}{\partial \omega_{jk}(n)} = y_k(n)$$

and inserting these results in the sensitivity factor calculation we get:

$$\begin{aligned}\frac{\partial \mathscr{E}(n)}{\partial \omega_{jk}(n)} &= \tfrac{\partial \mathscr{E}(n)}{\partial y_j(n)} \varphi_j'(v_j(n))\, y_k(n) \\ &= \delta_j(n)\, y_k(n)\end{aligned} \tag{1.7}$$

where $\delta_j(n)$ is the local gradient of the error, which can be expressed as:

$$\delta_j(n) = \frac{\partial \mathscr{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\partial \mathscr{E}(n)}{\partial v_j(n)}. \tag{1.8}$$

For the particular case where the loss can be described as a sum of the squares of the difference between the output values of neurons in the last layer and the target values d_j, we have:

$$\mathscr{E} = \frac{1}{2}\sum_{j\in C} \|d_j - y_j\|^2 = \frac{1}{2}\sum_{j\in C} e_j^2 \tag{1.9}$$

where C is the set of output neurons. The local gradient can then be expressed as:

$$\delta_j(n) = e_j\, \varphi_j'(v_j(n)), \tag{1.10}$$

which only depends on quantities from neuron j.

The component ω_{jk} of the weight matrix can then be updated as:

$$\omega_{jk}(n+1) = \omega_{jk}(n) - \triangle\omega_{jk}(n). \tag{1.11}$$

with:

$$\triangle\omega_{jk}(n) = \eta\delta_j(n) y_k(n). \tag{1.12}$$

Let us now look into how the weights connecting hidden layer neurons can be updated. For the layer immediately above the output layer, the local gradient of a neuron k is calculated from equation (1.8):

$$\delta_k(n) = \frac{\partial \mathscr{E}(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial v_k(n)} = \frac{\partial \mathscr{E}(n)}{\partial y_k(n)} \varphi_k'(v_k(n)). \tag{1.13}$$

We can now express the error relative to neuron k as a function of its output (the input to the output layer y_j):

$$\frac{\partial \mathscr{E}(n)}{\partial y_k(n)} = \frac{\partial \mathscr{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial y_k(n)} = \underbrace{\frac{\partial \mathscr{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}}_{\delta_j(n)} \underbrace{\frac{\partial v_j(n)}{\partial y_k(n)}}_{\omega_{jk}} \tag{1.14}$$

and inserting it in equation (1.13) we get:

$$\delta_k(n) = \delta_j(n)\omega_{jk}\varphi_k'(v_k(n)) \tag{1.15}$$

which tells us that the local gradient of the error can be expressed as a function of the local gradient of the neuron in the next upper layer (in the direction of input to output) times the weight connecting the two neurons, and the derivative of a quantity dependent only on the current layer, the activation function of neuron k.

The same kind of reasoning used in equation (1.14) can be applied to the layers preceding layer k, making the calculation of the errors in the backward direction very efficient, as we reuse the previously calculated local gradients from the layer above.

The reason behind the efficiency of the back-propagation algorithm is, however, also a weakness. As the local gradients from upper layers are used in the updates of lower layers, if for some reason they become close to zero, this propagates backwards, with the weight updates becoming close to zero, leading to the death of neurons and convergence stopping.

Through the training process, the weights of the network are changed through a process that aims to minimize a loss function. In supervised learning, for instance, the objective function would describe how good the network is at obtaining correct target values for the inputs. The network adjusts them so as to obtain a good enough approximation to the real mapping between the inputs and the outputs, by transforming the inputs into features where this mapping is possible. The weights and bias, therefore, encode what the network has learned.

It is also important to note that supervised learning in deep neural networks is generally not described by a convex loss function, so there is no global minimum. The loss function can have multiple minima and saddle points, and though stochastic gradient descent with back-propagation will tend to move the network towards a lower point in the loss function manifold, there is no guarantee it will lead us to a global or local minimum. The learning process ends when a satisfactory low value of the loss function has been reached, or the network stops converging further due to the death of neurons. In addition, because the examples are presented to the network in random batches in stochastic gradient descent, training networks with same architecture multiple times will start from different initial conditions and yield different weight updates. For these reasons, different instances of the same network architecture, learning from the same dataset will yield different trained networks, i.e., different hypothesis for the same class of models.

From the perspective of interpretability of a model, this makes things harder as multiple instances can result in quite different hypothesis. Most interpretability methods proposed for neural networks focus on attempting to understand a single hypothesis, or sometimes even just its results on an example or neighbourhood of examples. Understanding the variability of networks for the same model is a challenge much less explored in the literature, but very important from a fundamental point of view. This is the focus of this work, in the context of image classification.

To deal with different specific tasks, such as this, speech recognition, and natural language processing, specialized versions of deep neural networks have been developed. In the following, an overview of a few of these specialized types of networks is given.

1.2.3 Convolutional Neural networks

Convolutional neural networks (CNN) are a variant of feedforward neural networks (FNN) first introduced by LeCun *et al* [19] in 1989, which are specially conceived for doing tasks on images, such as image classification.

Feedforward networks are capable of doing image classification but are less effective than CNNs. This is due to two main reasons [20]. Firstly, image inputs are formed by pixel grids, w pixels wide and h pixels high. The total number of these can be quite high even for very simple images. Since the input of a FNN needs to have as many neurons as the number of features, $w \times h$, this means that just in the connections between input and a first hidden layer of the same size, we would get $(w \times h)^2$ weights. The higher the number of weights to be tuned, the higher the computation cost of training the network, since more values need to be tuned. As an example, for a 30×30 pixels image, this yields 837000, including the biases. For many applications, images with much higher resolution are needed and since FNN can have many layers, which are fully connected, the number of weights to be determined increases dramatically [20]. Convolutional neural networks have layers that are not fully connected, do parameter sharing in the convolution layers and have sparse connectivity, thereby decreasing the computational cost involved.

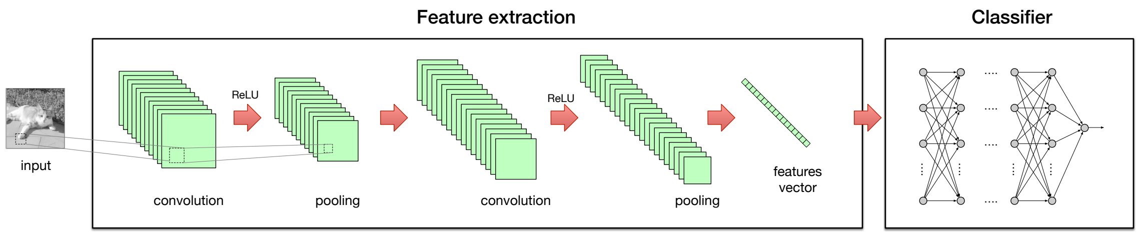


Figure 1.5: Convolutional network scheme. The set of convolution and pooling layers extracts features from the input image (here a 2D array since there is only one color channel) and the dense neural network at the end takes a feature input vector and does the image classification task.

However, the most significant difference between FNNs and CNNs is perhaps in the fact that they are specially adapted for grid-like input such as vectors or arrays, where the relative position of the array elements is important [20]. When FNNs take the pixel grid and transform it into a vector, information about the proximity of some pixels to others is lost. This makes them less efficient at detecting spatial features than CNNs.

CNNs are also structured in a different way than FNNs. In FNNs, the dense part of the network is responsible for both feature extraction and (as an example of a task) the classification, while in CNNs the two are performed by distinct parts of the network. While the classification part is done by a dense network, as in FNNs, the feature extraction is now performed by the convolution part, which uses multiple convolution layers to extract patterns in feature maps. In the following, we describe the structure and the functioning of each of these components in more detail.

The convolutional part of CNN is composed of two types of layers: convolution layers and pooling layers.

In convolution layers, a smaller matrix called kernel (or filter) slides over the image, and element-wise multiplication of the values in the kernel with the pixels values over which it is superimposed (receptive field) is done, followed by a sum of the results (see left panel of Figure (1.6)). In this way, it is performing discrete two-dimensional cross-correlation between the image I and the kernel matrix K (despite the name convolution being used interchangeably with cross-correlation in the field):

$$S(i,j) = (I \star K)(i,j) = \sum_m \sum_n I(i+m, j+m) K(m,n) \tag{1.16}$$

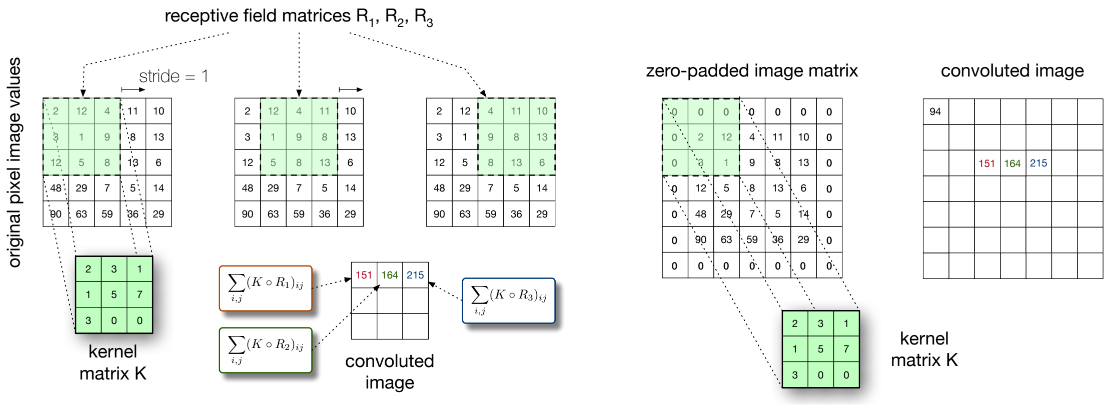
The values of these filters (kernel values) play the same role of the weights in FNNs and, together with the biases, are tuned during the training process. In the same way weights in FNNs measure the importance of the different inputs to a neuron, a change in the kernel unit values represents a change in the importance of the pixels in its receptive field. The result of these convolution operations is stored in the convoluted image matrix. These matrices map patterns in the image to features, and are therefore also called feature or activation maps. Parameter sharing results from the fact that the same kernel is used for the whole image for building a feature map.

A convolution layer performs convolutions with multiple kernels, whose feature maps are then stacked together. The convolution layer dimensions are then given by $w \times h \times d$, where d is the depth, given by the number of filters. Increasing the number of filters increases the capability of the layer to detect more patterns, but also the computational cost, as the total number of hidden units increases. These filters are normally square, and their size and the number of pixels by which they are moved in each step of the convolution operation are additional hyperparameters that can be tuned. Increasing the kernel size also leads to an increasing number of operations, as the number of hidden units to be tuned in the kernel also rises, but decreases the ability to discern finer details. The stride parameter roughly determines the spatial dimensions of the convoluted image, in an inversely proportional manner, and thereby affects the degree of detail in the convolution process as well.

If a square kernel of size $k \times k$ is used for convolution when the filter is at the edge, it cannot be centred on the border pixels, which would make the convoluted image smaller than the original, with size $4 \times [(k-1)/2]^2$. To prevent this decrease, a zero-padding can be done, i.e., adding pixels with zeros around the border so that the filter centre can be placed at the border pixels of the original window (see Figure (1.6)).

As in FNNs, the typical activation function used in convolutional layers is the ReLU or variants of this (e.g. Leaky ReLU).

One interesting property of convolution for image processing is that it is a linear operation that leads to equivariance. Translations in the original image will lead to corresponding translations in the convoluted image.



receptive field matrices R1, R2, R3
stride = 1
original pixel image values
kernel matrix K
$\sum_{i,j}(K \circ R_1)_{ij}$
$\sum_{i,j}(K \circ R_2)_{ij}$
$\sum_{i,j}(K \circ R_3)_{ij}$
convoluted image
zero-padded image matrix
convoluted image
kernel matrix K

Figure 1.6: Convolutional operation (left panel) and effect of zero-padding (right panel) in maintaining the same size of the pixel array as the original image.

The other type of layers present in the convolutional part of the network are pooling layers. These are placed after the convolution layers and downsample the image from the previous layer, therefore not requiring an activation function. The downsampling is done by sliding a window through the feature map (convoluted image) and replacing the pixels in its receptive field by a single value given by either the average (average pooling) or maximum value (max pooling) of the latter (Figure (1.7)). If pooling is done by taking the maximum pixel value, invariance to small translations is obtained, which can be useful for image tasks. The output is smaller in the spatial dimensions, but the depth is preserved. By reducing the convoluted image size, they decrease the hidden units in the next layers, making the network faster. With the application of successive

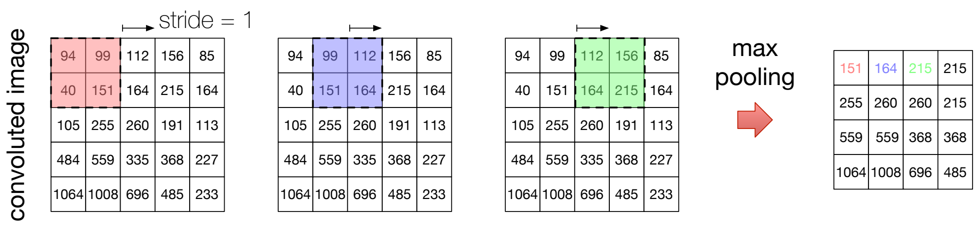


Figure 1.7: Max pooling operation with stride=1. The convoluted array values over regions (2x2 in this case) are aggregated. Here aggregation is done with the maximum function

layers of convolution and pooling, the network transforms the feature maps, increasing its depth during convolution and decreasing its spacial size during pooling. Finally, the feature map outcome of the feature extraction part is passed to the dense network (fully connected layers) that will take the learned features and classify them. Since the dense layers part needs the input to be a vector, the outcome of the final layer of the convolution part, which is an array, needs to be flattened, i.e., transformed into a vector.

The dense neural network responsible for the classification part works as any FNN for classification purposes. The input is the vector described above and is read by the first hidden layer, which must have as many neurons as elements in the former. This layer can be followed by several other fully connected layers up to the output layer, which will have as many neurons as the possible image classes being considered and where a softmax activation function is typically used.

So far we assumed that in the dense part of the network all the layers are fully connected. However, it is common to use some dropout layers between fully connected layers. These layers are not fully connected. A percentage of their neurons is chosen randomly and turned off. This procedure is repeated for each epoch. The selected neurons are dropped out in the sense they no longer are connected to the next layer neurons and thus do not participate in the training. The purpose of this operation is to increase the robustness of the network and decrease overfitting. During the training process, because of the different weights connecting neurons from one layer to another, some neurons might end up having much stronger or much weaker activations than others. This will cause the weights from adjacent neurons to lose or increase their influence on the local induced field for the next neurons and hence may be less tuned in the future. The dropout procedure prevents this, leading to a more robust network.

Dropout layers effectively average the behaviour of several networks (distinct in the neurons that are turned off). This fact can hinder the interpretation of networks that employ it, as they change their structure in a random way during training.

1.2.4 Autoencoders and Generative Adversarial Networks

Autoencoders are a type of generative neural network whose goal is to reproduce the input data in the output layer as closely as possible. They are composed of two parts: the encoder and the decoder, as can be seen in Figure 1.8. The encoder takes the input vector data $\boldsymbol{x}$ of size n and transforms it onto a representation with k components $f(\boldsymbol{x})$ (latent representation). The decoder then tries to take this representation and transform it back to the initial input vector, through a mapping $g(f(\boldsymbol{x})) \sim \boldsymbol{x}$. The training of the network is done in the same manner as in MLPs, and the loss function of autoencoders is a function of the error in reproducing the input example, $||\boldsymbol{x} - g(f(\boldsymbol{x}))||$, such as the mean square error. The learning is unsupervised as no label information is required.

If the number of components in the latent representation is smaller than the size of the initial vector, $k < n$, the autoencoder is said to be undercomplete. Such autoencoders can transform the input data onto a representation which contains its most important features in the latent space. It is said, in this case, that one is trying to pass the data through an *information bottleneck*. If, after training the autoencoder network, we separate the encoder part of the network, we can then use it to perform compression. On the other hand, the decoder can be used to generate new examples similar to those in the training data.

It is interesting to note that an undercomplete autoencoder with linear activation functions in the decoder does the equivalent of a principal component analysis (PCA)

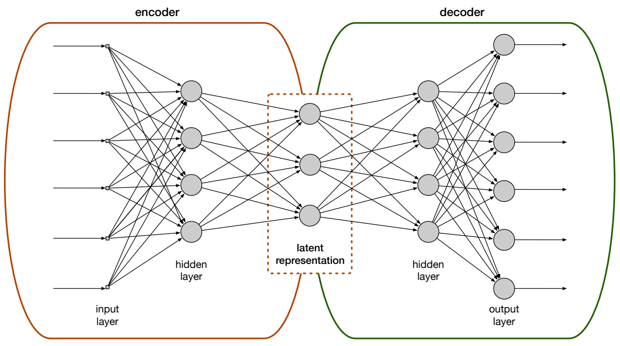


Figure 1.8: Example of autoencoder architecture. The information is passed from the input space to a latent space and then back to the input space. The central layer, of smaller width than the others, constitutes the *information bottleneck*.

dimensionality reduction in the encoder [21]. However, the use of nonlinear activation functions can generate powerful data representations. If the autoencoder is overcomplete, it is still possible to obtain a latent representation with a dimension smaller than that of the input data if sparsity (reducing the number of activations) or other regularization strategies are employed [21].

Variational autoencoders are a special class of autoencoders where the network tries to model the latent space as a specific probability distribution, usually, a mixture of Gaussian distributions, whose mean and standard deviation are learned.

Generative adversarial networks (GANs) [22] are another class of generative models, which can be used to generate samples for multiple tasks such as the generation of hyperesolution images (e.g. [23]), image to image translation (for e.g. converting satellite images to maps [24]), among others. Generative adversarial networks have some similarities to the decoder in variational autoencoders since they both try to generate new examples from a model probability distribution p_{model} which should be as close as possible as the real data distribution p_{data}. However, while variational autoencoders use an explicit p_{model} resulting from the training, in generative adversarial networks, a different approach is taken. Their training is also different.

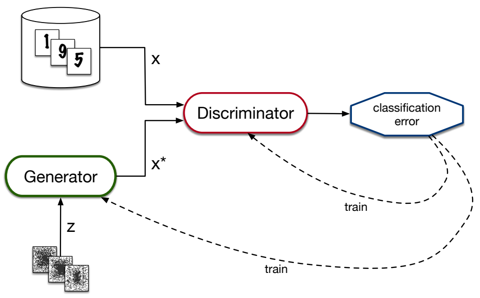


Figure 1.9: Generative adversarial network scheme. Adapted from [25].

A scheme depicting GANs is shown in Figure (1.9), where it is illustrated that these models are composed of two components, a generator and a discriminator. The generator tries to generate an example $\boldsymbol{x}^*$ starting from noise, as similar to the examples in the training data $\boldsymbol{x}$ as possible, and the discriminator acts as a binary classifier trying to distinguish if $\boldsymbol{x}^*$ and $\boldsymbol{x}$ are the real or fake sample. Training resembles a game where the discriminator and the generator try to beat their opponent: the discriminator by correctly classifying the fake and true samples, and the generator by producing better fake samples to fool the classifier [25]. There are now two loss functions, dependent on the parameters of both networks: $J^D(\boldsymbol{x}, \boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ and $J^G(\boldsymbol{x}, \boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$.

Training of this model amounts to trying to find an equilibrium where both the generator and the discriminator can no longer do better. Training of the networks is done simultaneously by cross-entropy minimization using gradient descent and back-propagation, as with other neural networks. It involves two steps. In the first discriminator is trained with the true and fake samples, keeping the generator network parameters frozen, and in the second step, the roles are reversed. The training continues until the networks are close to an equilibrium.

It is interesting to note that VAE and GANs can also be used in interpretability techniques. In [26], Usunova *et al.* assessed how different changes seen in MRIs could impact their classifier model by using meaningful perturbation technique. Removing different regions of the MRI image and replacing them with healthy tissue realistic-looking image regions generated by VAEs, allowed them to investigate how the regions with pathologies contributed to the classifier decision. On another example [27], Nguyen *et al.* used networks incorporating some of the principles from VAEs and GANs to generate realistic-looking images that maximally activate the network neurons. Applying this method (a variation of activation maximization [28, 29, 30]), one can better understand what types of objects / regions most influence the CNN classification.

1.3 Contribution to the topic

Despite the great achievements of neural networks in areas such as image and voice recognition, autonomous driving and even in the analysis of medical imaging for diagnostics, the progress in their interpretability has not been as fast. Though there has been a growing effort in understanding how they work, in many ways neural networks still remain black boxes, which can lead to problems, especially in applications with possible life-threatening consequences [9].

Many methods have been proposed in the interpretability of deep neural networks to address this issue, yet most of the current literature focuses on methods that provide explanations for an example or group of examples, or methods to interpret a trained network (hypothesis) behaviour and not that of the model (class of hypotheses). However, it is known that trained networks with the same architecture, task and dataset can give quite different solutions for different initial conditions (e.g. different weight initialization or the random selection of different batches). The application of different regularization methods (e.g. dropout, different penalties in the loss function), can also lead to different hypotheses. In addition, one can also have different families of hypotheses when, for the same dataset and task, two architectures are used that are equal in all but a single change, for e.g. the number of filters in a given layer. These sources of variability are not taken into consideration by the vast majority of current interpretability methods.

This work aims to study and develop methods to quantify the variability in the output of interpretability methods that can inform us whether they are robust in capturing attributes that are meaningful for the model and not just artifacts associated with a particular network.

Our approach is organized into two steps. In the first part, we start by training several networks for the same model or family of hypotheses to obtain a pool of hypotheses. In the second step, we apply two well-known interpretability methods which produce readily understandable and visually intuitive explanations to this pool: saliency maps and sensitivity to occlusion maps. We study the variability of their explanations, which in these two cases are in the form of heatmaps, from two perspectives, the variability between explanations of different examples obtained for the same network; and the variability between explanations for the same example obtained with the different network instances. For this study, we implement several tools. Firstly, we have implemented tools to train multiple network instances in a way that different architectures and datasets are easily exchanged so that multiple instances for different models can be easily trained. Secondly, scripts were implemented that extract the output of the two interpretability methods and determine quantitatively the dissimilarity (based on a distance metric) between all the explanations, in the two perspectives described above. Additional tools were implemented to take the distances between the explanations and determine if different networks behave similarly in terms of the interpretability attributes they capture for groups of examples; and if the explanations of different examples for each network are consistent from network to network.

With the tools described above, we were able to identify that, for certain models, the variability between networks for the same example can be greater that the variability between different examples for the same network. This constitutes a problem in the consistency of the diagnostic, since one would expect the opposite if the diagnostic was capturing relevant information regarding the attributes that are important for predictions of the network. In this way, we have developed a method to probe the consistency of the explanations of an interpretability diagnostic for different instances of the same model.

Chapter 2

State-of-the-art

In this chapter we review the state-of-the-art techniques for interpretability, explainability and visualization of convolutional neural networks, in the context of this dissertation. The techniques presented here are not an exhaustive listing of the interpretability methods in the literature but a selection of some that were either considered important in the field and/or particularly suitable for the goal of the project.

The methods were grouped into those which will require some adaptation/generalization for the purpose of this work, which are described in section 2.1; and those that can be directly applicable to the comparison of interpretability among hypotheses of a given CNN model, described in section 2.2.

2.1 Techniques that require adaptation for comparison of neural network model instances

Local interpretable model-agnostic explanations (LIME)

The local interpretable model-agnostic explanations (LIME) [31] is a method to produce explanations for predictions of classifier and regression models. It works by locally approximating them to a simpler surrogate model (e.g. a linear model), more easily interpretable and locally faithful to the original model under study.

For the surrogate model to be interpretable, its features are chosen such that they are inherently understandable by humans. In the case of images, for example, these interpretable features can be super-pixels [32], which are patches of contiguous pixels that can be obtained with suitable image segmentation algorithms (e.g. [32]). In datasets with text, the interpretable features could be words or expressions.

The surrogate model then acts in an interpretable representation space instead of the original feature space. This interpretable space is chosen to be the set of binary vectors which encode the presence or absence of interpretable elements (image patches in this case): $\boldsymbol{x}' \in \{0,1\}^{d'}$, where d' is the dimensionality of the space of interpretable representations.

Given the original model $f : \mathbb{R}^d \rightarrow \mathbb{R}$, LIME seeks to find the surrogate model $g : \{0,1\}^{d'} \rightarrow \mathbb{R}$ from a set of interpretable models G given by:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \, L(f, g, \pi_x) + \Omega(g) \tag{2.1}$$

where $L(f, g, \pi_{\boldsymbol{x}})$ is a loss that describes the lack of faithfulness; $\Omega(g)$ represents the complexity of g, which affects negatively its interpretability and can be limited by applying LASSO regularization [33] to select a certain number of features; and π_x is a locality function, which penalizes perturbed examples z according to their distance to the example being explained.

The faithfulness to the original model for the above examples can be described by a least squares loss function:

$$\xi(\boldsymbol{x}) = \underset{g \in G}{\operatorname{argmin}} \sum_{\boldsymbol{z}, \boldsymbol{z}' \in Z} \pi_x(\boldsymbol{z})(f(\boldsymbol{z}) - g(\boldsymbol{z}'))^2 \tag{2.2}$$

where the locality function limits the neighbourhood around the original prediction features where g is valid. To obtain a perturbed point $\boldsymbol{z}'$ close to the original example features $\boldsymbol{x}'$, a random number of features with positive values is removed, the choice of which is also uniformly random. Its locality weight is then obtained by transforming the features back to the initial space and using an exponential kernel of the L2 distance in the case of images or the cosine distance in the case of vectors of words.

Equation 2.1 describes in general the core idea of LIME. In [31], Ribeiro *et al.* set G to be the class of linear sparse models, whose instances can be expressed as $g(\boldsymbol{z}') = \boldsymbol{\omega}_g \cdot \boldsymbol{z}'$.

In the application examples presented by the authors [31], both for text and images datasets, it was found that the obtained linear model features and weights provided a humanly understandable explanation for the model prediction.

The authors [31] propose to give a more global perspective of the behaviour of the original model by selecting examples which offer a good coverage of the domain. The selection method, submodular picking, is based on a measure of importance of a given feature in the interpretable domain. However, the interpretable domain for images is not the same for each example, as the super-pixels will be different for distinct images, which renders the method inapplicable in this case. It should also be pointed out that whether the super-pixel patches that come out as the explanation for the model prediction are representative of the classes, or not, is a qualitative human judgement, not a quantitative metric.

For datasets like MNIST [34] and a synthetic dataset of geometrical figures in greyscale, the image patch could end up containing all the pixels in a digit/figure. However, for the latter one could instead define features such as “has 3 vertices”. For this and other more complex image datasets, one could then analyse the distribution of the surrogate model parameters for different model hypotheses. Another issue is that this is a method applicable to individual examples. To study its variability one would need to choose super-pixels that are the same for different images, such as square regions of pixels [35] and measure the variability in the weights of the linear surrogate model $\boldsymbol{\omega}_g$.

Testing with vector activating concepts (TVAC)

Testing with vector activating concepts (TVAC) [36] is a technique which aims to provide interpretability to networks by measuring the sensitivity of the model layers to a vector that points towards a humanly interpretable concept. Like LIME, this method aims to provide a bridge between the latent representation space and the space of humanly understandable concepts. However there are differences in the way this is achieved.

In this work, a humanly interpretable concept is defined by two sets of examples (which may be exterior to the training dataset), positive P_C and negative N, which are/are not representative of the concept. Like LIME, TVAC also involves the training linear classifiers, but here with the goal of identifying the direction normal to the boundary between the region where examples representative of the concept are, and the region of random examples that do not represent this concept. For a given layer, represented as a vector to vector function $f_l : \mathbb{R}^n \to \mathbb{R}^m$, where n is the number of inputs and m the number of outputs, these regions can be defined as $\{f_l(\boldsymbol{x}) : \boldsymbol{x} \in P_C\}$ and $\{f_l(\boldsymbol{x}) : \boldsymbol{x} \in N\}$. The unit vector normal to the hyperplane that separates these two sets is then the concept activation vector $v_C^l \in \mathbb{R}^m$. In this way, even though the vector lives in the space of the activations, it is interpretable by what it represents by construction: positive or negative association with concept C.

The sensitivity of the example $\boldsymbol{x}$ from class K to a concept C at layer l can be determined by taking the directional derivative of the layer activations in the direction of the previously obtained concept activation vector:

$$S_{C,K,l}(\boldsymbol{x}) = \nabla_{h_{l,k}}(f_l(\boldsymbol{x})) \cdot \boldsymbol{v}_C^l \tag{2.3}$$

where $h_{l,k}$ is the function that represents the remaining part of the network, upwards from layer l, and whose output is the log odds of $\boldsymbol{x}$.

With the sensitivity measure defined for a data point, the importance of the concept activation vector (CAV) to a whole class K can be obtained with TVAC$_{C,K,l}$, which is simply the fraction of elements of K which have a positive sensitivity to the CAV.

The authors [36] show that their results are able to detect bias in datasets (e.g. association of ping-pong balls with a particular race), consistent with previous work [37]. Kim *et al* [36] also showed examples where concepts for which a model shows high sensitivity are present on images obtained with the deep dream model [38], which generates images through activation maximization. Finally, TVAC can also be used to sort images based on cosine similarity with the CAVs, giving an ordering to their importance to the classification outcome [36].

One advantage of this method is that it is global in the sense that a whole class can be analysed, though this could be somewhat time consuming for big datasets. It also has the advantage of allowing the user to represent the concept to be studied by examples outside the training set. In this way, when comparing different trained hypotheses, one can avoid using examples which may have been previously seen by one network and not by the others.

One caveat with the TVAC approach is that any random sets of examples can generate a CAV vector. To ensure that the CAVs obtained are meaningful the authors then need to do a series of training runs and perform statistical significance tests. This is an extra step which involves additional computational time.

To analyse the variability of the method for different model instances, one can measure how either the TVAC or the actual CAV vector change between them for the same concept and between different concepts for the same network.

Linear probes

Alain and Bengio [39] explored how the discriminative power of a neural network evolves before and during training using a technique based on inserting probes along its layers. The probes were inserted at each different layer along the network and consisted in linear classifiers with a softmax activation function, and the different target classes were one-hot encoded. They were each trained with cross entropy minimization. The gradient of the backpropagation of the error was prevented from entering the probes, so that they did not interfere with the network training.

Several scenarios illustrating the potential of using probes to analyse the changes in discriminative power of neural networks were used. In one of the studied cases the authors [39] used the MNIST convolutional model available in Tensorflow [40] and added probes before and after each convolution, pooling and fully connected layers. For each epoch, they recorded the test error for all the probes. Interestingly, they observed that with the randomly initialized filters and before training even begun, the test error in the linear classifier probes decreased after the initial convolutional layer (32 filters, ReLU, maxpooling), increasing afterwards. This suggests the randomly initialized first layer already produced a transformation into a representation easier for classification. After 10 epochs of training, the error decreased at all but one probes, with only a minor increase in this one. The authors also showed how probes could be used to diagnose the distinct evolution in the discriminative power of network layers, to identify training issues and testing corrections.

It could be interesting to use linear probes to assess the evolution of the discriminative power of the layers of different instances of a neural network model. From the distribution of the training error at each probe for a set of model hypotheses, one could analyse whether there are significant changes among them and thus check if these different instances of the model developed more efficient filters or fully connected layers. This would provide a measure of variability between instances for a given sample of examples. To analyse the variability for the same network instance but different examples, one could do train the probes inserted on a given network for different samples of the test dataset and measure how the training error at each probe changes.

Deep inside convolutional networks

Saliency map techniques are aimed at understanding the influence of different parts of the input on the classification. One of the techniques used to achieve this is to generate an image which maximizes the score for a given class C, and thus, should be reminiscent of the object or concept represented by that class [28]. More formally, one aims to produce the image I^* that satisfies [28, 29]:

$$\underset{I}{\operatorname{argmax}}\, S_C(I) - \lambda \|I\|_2^2 \tag{2.4}$$

where $S_C(I)$ is the class C score function and the last term is an L2 regularization term controlled by the parameter λ. In general, the output of the softmax layer in CNNs represents the probability P_C that the image belongs to class C. This probability is related to the score as $P_C = \exp S_C(I)/\sum_j \exp S_j(I)$, where the index j runs over the possible classes. This technique was first proposed by Erhan *et al* [28] but first applied to CNNs by Simonyan [29].

A seed image I_s (the zero image if zero-centering is used) is provided to the trained network, and its score $S_C(I_s)$ is used as an initial condition. Image I^* is then obtained by backpropagation of the score function for class C, $S_C(I_s)$, doing the derivatives relative to the image and not to the weights of the network, which are kept fixed. The image mean of the training set is then added to the result. While the contours and some marks in the synthesized image do resemble their label meaning, the information extracted is of a qualitative nature.

Simonyan [29] propose an additional method to indicate which pixels in an image I_0 are most relevant for its class score $S_C(I_0)$. To estimate their relevance, a linear approximation to the class score at the image I_0 point is done: $S_C \approx \omega^T I + b$, where I is the vector resulting from the flattening of the image array and ω is given by: $\omega = (\partial S_C/\partial I)|_{I_0}$. The importance of the pixels to the score of the image for a given class C is then estimated by the weights ω in the linear model (the derivative of the score function with respect to the image pixels).

To obtain a class saliency map, which represents the pixel support for the score S_C, all that remains is to map the weights into the saliency map matrix $M \in \mathbb{R}^{m\times n}$, obtained by mapping the flattened pixels indexes back to a pair index corresponding to the coordinates in the image array. Designating this mapping function by $h(i,j)$, the saliency map matrix is then $M_{ij} = |\omega_{h(i,j)}|$. For colored images, for a given pixel, the maximum weight across color channels is used: $M_{ij} = \max_c |\omega_{h(i,j,c)}|$, where c is the index of the color channels.

The class saliency map is a type of explanation for a given image, an example from the dataset. To analyse variability a dissimilarity metric such as a distance between the maps reshaped as vectors can be used. These distances will be determined for different image examples using the same network and for different networks for the same example. The analysis of variability based on these distances will be detailed in chapter 3.

DeepLIFT

DeepLIFT [41] is a backpropagation method designed to overcome some of the issues present in some of the backpropagation methods based on gradients. Many methods of attributing relevance to pixels are based on determining the gradient of the score with respect to input pixels, in a backward pass. With DeepLIFT, the main idea is to measure the contribution of an input or a neuron activation relative to its reference value, to the difference of the target output relative to its own reference value [41]. The reference values are picked according to domain-specific knowledge, and need only to be set to the input values, as the reference for all the neurons above the input layer can be determined through propagating the activations. As with other backpropagation methods, it requires a single backpropagation pass to obtain the results.

The main advantages of DeepLIFT [41] is that the calculation of importance values relative to reference values allows it to overcome several counter-intuitive or misleading results from other methods in particular situations.

In essence, the method considers that the importance of the target value relative to its reference value is a sum of the contributions from all the inputs that contribute to it: $\sum_{i=1}^{n} C_{\Delta_{x_i}\Delta_t} = \Delta t$, which is designated the summation-to-delta rule. It assigns a contribution from each of the inputs to the difference of t from its reference value (Δt). To continue to split the importance of each neuron to Δt in the backward pass, multipliers are used. The multiplier $m_{\Delta x \Delta t}$ is defined as: $m_{\Delta x \Delta t} = \frac{C_{\Delta x \Delta t}}{\Delta x}$.

To determine the multipliers for any hidden layer in the network, the chain rule of multipliers is used. Assuming an input layer with neurons $x_1, x_2, ..., x_n$, a hidden layer with neurons $y_1, y_2, ..., y_n$ and the output neuron t, the chain rule for multipliers states that: $m_{\Delta x_i \Delta t} = \sum_j m_{\Delta x_i \Delta y_i} m_{\Delta y_i \Delta_t}$. This is similar to what is done in the training of a network during backpropagation of errors, where the chain rule of derivatives also allows us to propagate and distribute a quantity along the neurons in the network, starting from the output and moving towards the input. Here, we are distributing the importance of the deviation of inputs from their reference value to the resulting deviation of the outputs from their own reference values.

Compared to other backpropagation-based methods, DeepLIFT rules for distributing importance in the backward-pass solve several issues that methods based on gradients, gradients x input and layerwise backpropagation do not.

One such problems is that of saturation of the contribution of inputs to the output, which occurs when changes in the inputs do not lead to a change in the output and the gradient becomes zero, even though the inputs are not. Because the gradient becomes zero, all the above mentioned methods assign zero importance to the inputs, which is counter-intuitive.

The second problem is thresholding, which can arise, for instance, when the output is only bigger than zero after a given input threshold. In gradient methods and gradient $\times$ Δinput methods, this leads to a discontinuity in the gradient, which in turn leads to artifacts in the results.

These two issues are absent from two backpropagation methods: DeepLIFT and the integrated gradients method. However, there is a third issue that integrated gradients cannot address but DeepLIFT can. When two inputs are not zero but cancel each other out in one of the neurons in the network (for e.g. in min/AND network), the importance/relevance will be assigned to only one of them, which is not very meaningful since both contribute to the result, which represents a relation among them. Because DeepLIFT can split the relative activation values along the network into positive and negative contributions which are propagated separately, it can deal with this situation and will assign equal importance contributions to both inputs.

Both DeepLIFT and the attribution methods described in the previous section produce a heatmap per example. To analyse the variability of the diagnostic applied to a model, one could take obtain the heatmaps for the same network instance for different image examples and for different image examples using the same network and flatten them into vectors. This would yield a set of vectors whose pairwise distances can be calculated. With these distances one could, as described in chapter 3 obtain measures of variability that allow us to understand the robustness of the diagnostic.

Network dissection

Network dissection is a framework developed by Bau *et al* [42] to quantify the interpretability of convolution layers and individual units in them. It allows us to evaluate whether different individual convolution filters are detectors for semantic visual concepts, or, in the terminology of the authors, if they become aligned with the visual semantic objects [42]. If the semantics of the images passing through the network can be disentangled, this approach allows their interpretation to be factorized [42].

To assess this alignment, a densely labelled dataset is required, such that each image can have several segmented regions at the pixel level. The authors [42] created this dataset, named Broden, by gathering multiple labeled segmented datasets: ADE [43], OpenSurfaces [44], Pascal-Context [45], Pascal-Part [46] and Describable Textures Dataset [47]. These datasets contain annotation of objects, object parts and materials at the pixel level, and of textures and scenes for full images. Each pixel also has an additional color annotation. Only images for which there is at least one labelled concept were used and labels of semantically close concepts or synonyms were merged. In this manner, for each image $\boldsymbol{x}$, an annotation mask $L_c(\boldsymbol{x})$ could be defined which indicates which pixels of that image are associated with each concept c.

The method is applicable to any convolution filter in the network and involves a single forward pass in the network. Its core idea is to obtain binary activation masks corresponding to the pixels with highest activations over the whole dataset and, assess whether they align with the annotation mask for a given concept, using the intersection over union score metric.

Starting from a trained network with k convolution layers, each image $\boldsymbol{x}$ in the dataset makes a forward pass in the network during which the activations of each convolution filter $A_k(\boldsymbol{x})$ are recorded. For each pixel location a distribution of activations $p(a_k)$ over the whole dataset is obtained, where a_k is the activation for a specific pixel location. From this distribution a quantile T_k is determined from a threshold τ, such that $p(a_k > T_k) = \tau$, which effectively determines the maximally activated pixels over the whole dataset for a given filter k. In their work, Bau *et al* [42] set the threshold to $\tau = 0.004$. This threshold defines a binary segmentation mask $M_k(\boldsymbol{x})$ for each image $\boldsymbol{x}$ in the dataset and each filter k, with the value one on the locations where $p(a_k > T_k) \geq \tau$ and zero elsewhere. Convolution filters smaller than the original image were up-sampled through interpolation.

To be able to determine whether a given filter k aligns with a concept c, i.e., responds to it with its maximal activations, the intersection over unit score metric is used:

$$IoU_{k,c} = \frac{\sum |M_x(\boldsymbol{x}) \cap L_c(\boldsymbol{x})|}{\sum |M_x(\boldsymbol{x}) \cup L_c(\boldsymbol{x})|} \tag{2.5}$$

where the sum is over the images of the whole dataset that have at least one annotation mask for the concept c and $|\cdot|$ is the cardinality of the set. This score represents the accuracy of filter k in detecting the semantic concept c, and a threshold of $IoU_{k,c} > 0.04$ was set for the filter to be considered a detector for it. Since each convolution filter k can be aligned with several different concepts, the one with the highest score is attributed to it.

The $IoU_{k,c}$ score can be used to establish a quantitative metric for the interpretability of a layer which can be compared across networks: the number of unique concept detectors existent in that layer. Bau and coworkers [42] studied the effects of different initializations and regularization strategies on the number of unique detectors found in the network. To test the effect of initialization, they used the AlexNet [6] as a baseline network model and trained it with three new different initial random weights to obtain different initial conditions. It was found that neither the number of unique concept detectors nor the total number of concept detectors changed significantly. However, it would be interesting to see if this conclusion holds for a much higher number of different initial conditions (network model hypotheses).

One limitation of this work is that it is designed to consider only concept detectors that are composed of a single convolution filter. It is known from previous works [48, 49] that multiple filters can be involved in encoding a single concept, which is not considered here. Fong *et al.* [50] addressed this issue and generalized the work of Bau and coworkers [42], proposing a new framework to study interpretability: Net2Vec.

Net2Vec

Net2Vec [50] is a framework based on the above described dissection technique but that generalizes it, starting from a different perspective: that multiple concepts can be coded in a single filter and multiple filters may be needed to encode a given concept.

To generalize the network dissection technique, a two-step method is followed. In the first step, as in network dissection, the activations of a special probe dataset (Broden) are saved during a forward pass on the pre-trained network. In the second step, not only are these activations used to calculate the intersection over union score but their weighted sum is also used to learn an embedding for semantic concepts in two tasks: segmentation and classification of images. This weighting method provides not only a way to compare the alignment of combinations of different numbers of filters with a concept but also to obtain embeddings that can be used to perform analysis similarly to what is done with word embeddings, including applying clustering and dimensionality reduction techniques.

On the segmentation task, the method for calculating the intersection over union score is the same as in Network dissection [42], except for a few differences, the biggest of which is that the segmentation masks are now calculated using only the training images of the probe dataset and the IoU_{set} score is determined over the subset of validation from the same probe dataset. To compare how the single filters fare in relation to combinations of filters in encoding concepts, weights for linear combination of filters are learned for each task: segmentation and classification. In the case of classification, a bias is also learned. For both tasks, the weights are learned through stochastic gradient descent using a per-pixel cross-entropy loss.

It is important to note that the process of learning the weights for a chosen layer implies taking the pre-trained AlexNet network and ending it right after that layer with a single fully connected layer and an output layer.

Using the learned weights of either tasks, it is possible to compare both the IoU_{set} score and the classification accuracy for different selections of filters: all of the filters in one layer; a single filter, by putting all other weights to zero; or a subset of filters, by relearning selected weights and with the remaining ones at zero.

By comparing the above two metrics for all the filters in each of the five convolution layers of the AlexNet, it was found that, averaging over all the concepts for each category of the Broden dataset, the linear combination of all filters always outperformed that of a single filter [50]. In addition, when looking at concepts individually, for around 90% of the concepts, the linear combination of filters resulted in better performance on the same metrics [50]. These results show that networks rely on distributed encodings for most of the concepts.

By measuring the number of times a filter was selected as the best filter for a concept and plotting its distribution, the authors [50] also showed that, though the majority of filters are not the top filter for any concept, many filters encode more than one concept.

Another powerful application of this weight learning technique lies in the possibility of, using either the weights ($|w_k|/\|\boldsymbol{w}\|_1$) or the IoU score for a given concept c ($IoU_{set}(c; M_{k^*}, val)/IoU_{set}(c; M(\cdot, \boldsymbol{w}), val)$) , to obtain an embedding of each concept in the probe dataset, where each axis represents a filter. Using these encodings, it is possible, as the authors [50] show, to apply dimensionality reduction techniques to plot all the concepts and analyse semantic connections between them, such as proximity of similar concepts for the segmentation task, and of concepts that are part of another one through compositionality, in the classification task case.

Finally, learned embeddings for a set of concepts can be compared between different networks. A cosine distance matrix is determined for each network according to $d(W) = W \cdot W^T$, where W is the matrix whose C rows are the normalized concept embeddings for each concept. Once $d(W)$ is determined for two different networks, the similarity between their learned representations for the concepts in C can be determined as the Euclidean distance between the two matrices. This was done in the paper and can be visualized either in a matrix or projecting the different network embeddings in 2D with absolute metric multidimensional scaling. The possibility to compare embeddings can thus be used to compare multiple networks, and in this way, lends itself to direct application in the analysis of multiple instances of the same neural network model, which is at the core of the work of this dissertation.

However, one limitation of this diagnostic is the requirement of having annotation masks for the different concepts whose encoding in filters one would like to study. For many datasets this is not readily available and can be cumbersome to obtain.

Deconvolution

Zweiler and Fergus [51] proposed to use deconvolutional networks [52] as a tool to visualize the stimuli that most activated a feature map in the image pixel space for any convolution layer in the network.

Contrary to other techniques, it does not involve backpropagation of gradients. Instead, a series of operations designed to undo the sequences of convolution, rectifying and pooling operations is performed by attaching the deconvolutional network to a particular convolution layer.

The network is trained normally and, when examining new images (e.g. from the validation set), the only modification to the forward pass of the network is that the positions of the maxima from max-pooling in the rectified feature map are stored for each image.

The deconvolution operation starts from the pooled rectified feature map of a convolution layer. Since the pooling operation is not invertible, what is done to reverse it is an upsampling. A new map is created with the dimensions of the feature maps before the pooling operation. To fill it, each maximum associated with the pooling operation is repeated in all the receptive field that originated it when the kernel was overlapped with them. This can be done thanks to the fact that the positions of these maxima in the rectified feature map were recorded in the forward pass as described above.

After the unpooling, a normal rectification operation is applied. While it does not invert the ReLU of the forward pass, it ensures the feature maps remain positive.

Finally, it remains to do an up-convolution of the activation map, which is achieved doing a convolution of the rectified maps with the transpose of the initial convolution matrix. If the convolution layer to be examined is not the first, the procedure of unpooling, rectifying and upconvoluting is repeated until the image pixel space is reached.

This technique contributes to the interpretability of the network by showing in the image pixel space, which is more understandable to us than that of the features, the stimuli that most activate a given filter.

The authors [51] applied this deconvolution network probes to the different filters in a network with the architecture of [6] with slight modifications and trained it on the dataset ImageNet 2012 [53]. By depicting the nine strongest activations for each filter, they were able to gain insight into the structures the network was responding to, at each level of the feature extraction part of the network. Interestingly, while for the lower layers it was observed that the activation maps showed great similarity among themselves, higher layers appeared to highlight certain higher level features strongly and show higher invariance.

The highest activations of a filter could be used to compare filters based on their response, in a similar way to what is described in [54]. To compare filters from the layers of different networks, one could identify the strongest activation pixels and the images where they occur for one filter, and then pass them through the other. The same would be done with the filters switched. If they both had high activation values for the same pixels in the same images, this would suggest they are similar in the patterns they detect. From the similarity of the normalized activations of both filters for the pixels and images that maximally activate each one could derive a metric of the similarity of the filters, which would then be used in the study of the variability of the layers of different instances of the same model.

In their work, Zweiler and Fergus [51], also studied the sensitivity to occlusion of networks regarding different regions of an image. This allowed them to check which locations of the image were more important to the network prediction. The technique consists in covering a square region of an image with a grey square and determining the value of the probability of the class the network predicted with this occlusion. This is repeated for different positions of the grey square, and the predicted class probability values for the altered images are assigned to the location of the occluded region. Together, these values form a heatmap of the changes in the probability that the network was giving to the predicted class with the original image. This mapping gives insight into the regions most relevant for a network prediction.

The variability of the sensitivity to occlusion maps is associated with a single network instance and a specific example (image). However, its variability between examples for the same network, and between networks for the same example can be studied using the distances between the occlusion maps in these two perspectives. This will be more detailed in chapter 3.

2.2 Techniques with direct / almost direct application to comparison of neural network model instances

Visualization of the hidden activity of neural networks

Rauber *et al.* [55] explore the use of different types of visualizations to better understand the representations of data learned by intermediate layers, and the relationships among different neurons in the context of a classification task.

Let us consider a neural network as a function $f : \mathbb{R}^m \longrightarrow (0,1)^d$ with a binary vector outcome that describes whether the input belongs to each of the d different classes and m the dimensionality of the input. The activations of a layer ℓ for a given input can then be represented as a vector, where each component corresponds to the output of a neuron of that layer in response to the input: $\boldsymbol{a}_{\boldsymbol{x}}^{(\ell)} = \{a_1^{(\ell)}(\boldsymbol{x}), a_2^{(\ell)}(\boldsymbol{x}), ..., a_q^{(\ell)}(\boldsymbol{x})\}$, where q is the number of neurons in this layer.

To visualize the representations learned by a given intermediate layer, the authors [55] use a dimensionality reduction technique called t-SNE to project the corresponding activation vectors (one per input datapoint) into two-dimensional space. This technique attempts to preserve the high-dimensional structure of the data, namely neighbourhoods and clusters. To assess the quality of the projections, the neighbourhood hit (NH) metric was used, which is the fraction of the n neighbours of a point which belong to the same class as it. Each datapoint was colored according to its predicted class.

This projection technique was applied by the authors [55] to both multilayer perceptron networks (MLPs) and convolutional neural networks (CNNs), using three different datasets: the MNIST dataset of handwritten digits [34], the SVHN dataset of house number digits [56] and the CIFAR-10 dataset [57], which contains images belonging to ten different classes.

It was found [55] that the more spatial spread out between each other were the classes after training, the better the performance of the network. In addition, datasets which already showed reasonably separated class clusters before training were easier to classify [55].

Another important result from the analysis of projections was that semantic information could be extracted from these visualizations [55]. As an example, in the SVHN dataset projection of the last layer before the output, it was found that datapoints belonging to each class were clustered, but also that these clusters appeared divided into two smaller ones. Upon inspection it was found that one belonged to light-colored digits in a dark background, and the other one to dark-colored digits in a light background.

Rauber *et al.* [55] also demonstrated how multiple projections could be combined in a single graphic as trajectories to show either inter-epoch evolution (with training progress) or inter-layer evolution (resulting from the flow of the data through the network), with the brightness of the color indicating the training iteration or layer number. In the first case, it could be seen that the further along the training, the further away from the projection center the different classes become, spreading out in different directions according to their class. In the second, the clusters associated with the different predicted classes were observed as the datapoints flowed through the layers.

The authors [55] also proposed a novel technique to visualize relationships among neurons in the same layer. In this case, each vector in the high-dimensional activation space contained the activations of a single neuron over a subset of a dataset, i.e., $\boldsymbol{a}_j^{(\ell)} = \{a_j^{(\ell)}(\boldsymbol{x}_1), a_j^{(\ell)}(\boldsymbol{x}_2), ..., a_j^{(\ell)}(\boldsymbol{x}_s)\}$ for neuron j and s datapoints in the subset. For this projection, the absolute metric multidimensional scaling (MDS) [58] was used. This technique displays the points in two-dimensional space according to their similarity. In this case, the similarity between two neurons i and j was measured as the distance $d_{i,j} = 1-\rho_{i,j}$, where $\rho_{i,j}$ is the Pearson correlation coefficient. By using this measure, the goal is to identify neurons that participate together in the classification of a certain class, for example. This could be assessed by using a gradient of color to depict how high the discriminative power of a neuron for a certain class was compared to other classes (determined using extremely randomized trees [59]). Analysing these projections before and after training showed that the neurons that had the highest discriminating power for a particular class were near each other after training, whereas before training they were spread out in the projection [55]. Additional insight could also be gained in other types of visualization, such as a neuron projection where these were colored according to the class for which they had the most discriminative power (discriminative neuron map) or analysing neuron maps in combination with activation projections.

These studies suggest a great potential in using visualization to extract semantic knowledge. The projections of the activation vectors of a layer for a dataset sample could be applied to different network instances to qualitatively assess how well are able to discriminate between different classes. However, it is not clear how a quantitative measure of dissimilarity between such maps could be obtained in order to measure their variability.

Convergent Learning: Do different neural networks learn the same representations?

Li *et al* [60] examined the similarity between learned representations from neural networks with the same architecture but different initial conditions for training using correlation between units in a layer as a metric. The architecture used is based on the AlexNet network [6] and trained on the Imagenet dataset [53]. In their work [60], they try to assess whether a correspondence between units in the different networks can be established; and, if so, whether it corresponds to learning the same features.

In the analysis of Li *et al* [60], they consider a vector $X_{l,i}^n$ with all the activations of neuron i from network n in layer l (or in the case of convolution layers the activations for one of the filter positions) to represent a unit in a layer. They then study how close are the representations of a layer in two networks with the same architecture but different initial weights (networks n_1 and n_2) by constructing the between-net correlation matrix using the vectors $X_{l,i}^{n_1}$ and $X_{l,j}^{n_2}$ from all the units in that layer. The between-net correlation matrix determined for networks 1 and 2 maps the correlation between the activation of unit in i (row i) from the first network to the unit j (column j) in the second one, regardless of their magnitude.

To try to understand whether the layer in the two networks learns the same representation the authors [60] look for permutations of units that maximize the between net correlation. Two types of alignments by permutation are studied. In the first, the permutation is done by simply looking in each row for the column which has the maximum correlation value. This corresponds to bipartite semi-matching. In the second type, which corresponds to bipartite matching, each unit in both networks can only be associated with one unit from the other at the most. The authors [60] then look for the permutation that maximizes the sum of the correlations values.

From the comparison of both types of matching for the same units, the authors [60] find that there are some units for which a one-to-one matching is possible between both networks, but not all. When the correlations of both types of matching are different, the one associated with bipartite matching is much lower, which the authors [60] speculate may be a result of a concept being represented by different number of units in the two layers, such that the extra units in one end up associated with units in the other to which they are not very correlated to.

In addition to one-to-one alignments, the authors [60] also tried to find many-to-many mappings, using spectral clustering. Results indicate that these exist as well, and that the clusters in each of the networks that represent the same concept do not necessarily have to have the same number of units [60].

Singular vector canonical correlation analysis (SVCCA)

Singular vector canonical correlation analysis (SVCCA) is a method proposed by Raghu *et al.* [61] to compare representations learned by two sets of neurons. These can be from different layers in the same network or from layers of networks with different architectures.

This technique looks at the neurons of a neural network as vectors where each element is the output of the neuron in response to an image from the input dataset. In this perspective, the activation vector of each neuron holds all its output from processing a dataset, and the layers formed by a set of neurons correspond to the subspace spanned by the activation vectors of its neurons. These activation vectors then contain a representation of the dataset by the corresponding neurons.

SVCCA involves two steps: singular value decomposition followed by canonical correlation analysis, and is applied to two sets of neuron activation vectors. The role of the singular value decomposition is to find the most important directions in terms of maintaining the maximum variability of the data. It takes the k singular value vectors which retain the most variability and projects the original two sets of vectors onto these directions. Subsequently, canonical correlation analysis is applied on the two new sets of vectors to linear transform them into orthogonal pairs of vectors that are maximally correlated.

One of the possibilities opened up by this technique is to study the intrinsic dimension of a network layer. To do so, the authors [61] took two instances of a convolution network with different initializations trained on the CIFAR-10 dataset. In one of them, the layers under study (one pooling layer and two fully connected layers) were limited to the top k SVCCA neuron activation vectors, whereas in the other network the k neurons either with the highest activations or picked randomly were kept. Comparing the accuracy of these networks varying the number of directions chosen, revealed that this number is much smaller than the number of neurons on the original network. This suggests the intrinsic dimension of the layer is smaller than its number of neurons. On the other hand, when the k SVCCA vectors were projected back onto m of the original neuron directions, it was found that, to achieve the same accuracy as using the SVCCA directions, the number of original directions (m) needed to obtain the same accuracy was quite higher than k, suggesting that the SVCCA directions are distributed among more than one neuron.

The authors also propose a new metric to compare the alignment of two layers. From the application of SVCCA to two sets of neurons, one for each layer, one can obtain, in addition to the aligned directions, the correlation value corresponding to each of the two layers ρ_i, where the index i identifies the layer. The proposed metric to measure the overall alignment of the representations from the two layers is designated by SVCCA similarity $\bar{\rho}$ and is given by $\bar{\rho} = [1/\min(m_1, m_2)]\sum_i \rho_i$, where m_1, m_2 are the sizes of the two layers.

Raghu *et al.* [61] used SVCCA similarity to compare all the layers of each of the networks (convnet and Resnet) at different instants during training with the same layers (of the same network) at the end of training. In both networks it was observed that learning occurs generally bottom-up, with the bottom layers reaching their final similarity values earlier than top layers. This observation suggested that freeze training could be used to save computing time during training.

Importantly, the SVCCA provides a metric to compare the alignment of the different sets of layers, either with layers from the same network of from a different one.

Insights on representational similarities in neural networks with canonical correlation

Morcos *et al.* [62] build upon the singular vector canonical correlation analysis (SVCCA) method developed by Raghu *et al.* [61] and propose a new method, projection weighted canonical correlation analysis (PWCCA), which replaces the singular value decomposition step by a different technique, that decreases the signal to noise ratio in the canonical correlation analysis (CCA) representations.

Canonical correlation analysis was applied to layers of a convolutional neural network in the same manner as [61], to compare the representation of a layer at each iteration to its representation at the final time-step (with $t_{final} > t_{convergence}$). From the observation of the correlation coefficients obtained, it was found that, even though many were close to their final value at convergence time, some continued to change. In addition, the representations of two sets with the 100 most stable and least stable CCA directions at an early time were compared through CCA with their representations at $t_{final}/2$, by measuring the CCA distance between them: $d_{SVCCA}(L1, L2) = 1 - \bar{\rho}$ (as defined above). This revealed that the set of stable directions remained so after convergence, while the unstable set diverged. These two results suggested the unstable directions could be mostly noise, and unimportant for the network performance. This motivated the proposal of a new metric for comparing layer representations: $d_{PWCCA}(L1, L2) = 1 - \sum_{i=1}^{c} \alpha_i \rho^{(i)}$, a pseudo-distance between layers that weights the correlation coefficients $\rho^{(i)}$ by the projection of the CCA vectors from layer i, $\boldsymbol{h}_i$, onto the original directions of that layer $\boldsymbol{z}_i$ yielding $\alpha_i = (\boldsymbol{h}_i \cdot \boldsymbol{z}_i / \sum_{j=1}^{c} \boldsymbol{h}_j \cdot \boldsymbol{z}_j)$.

This metric was shown to be more robust to noise than d_{SVCCA} in experiments between neural network representations subject to the same signal and a noise whose amplitude was controlled.

Since SVCCA and PWCCA look for affine transformations that maximally align directions, these techniques are more powerful than the approach of Li *et al*, which only looks for permutations between layer units. For instance, SVCCA and PWCCA, can look for correlations between layer subspaces spanned by sets of basis vectors rotated in relation to one another. Both SVCCA and PWCCA also provide pseudo-distance metrics which could be used to apply absolute metric multidimensional scaling to visualize the similarity among different hypotheses of the same network model. These pseudo-distance metrics provide a quantitative way of comparing layers between different hypothesis (network instances), which could be used to study their variability.

Chapter 3

Variability of interpretability explanations in model instances

3.1 Introduction

There are several interpretability diagnostics for convolutional neural networks which are local, i.e., they provide an explanation for an example. Saliency maps and occlusion maps are two well known local interpretability diagnostics, among others. Both try to provide a measure in pixel space of the importance of features to the network prediction, though through different routes. For these diagnostics, as well as others, it is important to understand whether they provide some information about the model (architecture + classification task) or there is too much variability in the explanations from instance to instance. If one trains under equivalent conditions, except for the random initialization, hundreds of instances of the same model and dataset and ask for explanations of a set of examples, will they vary significantly? Is this variability sensitive to the type of problem, the depth of the networks or other parameters? How is this variability expressed across different examples and network instances?

To understand this, one must first devise a method to measure it, to take the explanations for different examples and different networks and try to extract information about their variation at a model level.

In section 3.2, we detail our approach to the problem of assessing variability among different instances of a model for the same example (image) and between different examples for the same network instance. The tools developed and the methods used in the following sections are detailed. In section 3.3 we describe the models and datasets used in our study. This is followed by section 3.4 where it is shown how the variability analysis methods developed can identify problems in the consistency of an interpretability diagnostic. A more detailed exploration of the characteristics of this variability is described in section 3.5 for each of the diagnostics using the Cartoon dataset with a number of different models. Finally, preliminary results on the variability analysis of a VGG11 model [63] with a subset of the CIFAR10 dataset [57] are presented in section 3.6.

3.2 Methods and tools

Many interpretability diagnostics produce an explanation for a given example or groups of examples, i.e., they are local, so different examples can have different explanations within the same network. They will also vary from network to network for the same example. Even if the classification task and the architecture of the CNN are the same, i.e. the model is the same, the random initialization will generate differences from one network (model instance) to another, which will in turn lead to different explanations for the same example.

The problem then arises of how much can an explanation produced for a specific example and specific instance tell us about the behaviour of the model (architecture + task). In other words, if one runs N instances of the same model and apply an interpretability diagnostic, one will obtain N explanations for a single example. Will these explanations be close, or will their variability be such that the diagnostic does not reveal attributes that are general to the model, or extracts attributes which are artifacts that are not meaningful to the network predictions?

Intuitively, one would expect the variability between explanations of different examples for the same network to be bigger than the variability between explanations of the same example, produced with different networks (model instances) with the same accuracy. However, this is not always the case, as we will see in the next section. This constitutes a problem in extracting meaningful information from an interpretability diagnostic that exhibits this behaviour. In the following, we describe a tool we implemented and methods we developed to explore both the variability between explanations for different examples produced by the same network, which we will refer to as *variability per network*; and the variability between explanations produced with different networks, for the same example, which will be referred to as *variability per example* or *variability per image*, since this work concerns CNNs trained to classify images.

These tool and methods are quite general in their applicability, as long as one can identify a metric for the differences in the outcome of the interpretability diagnostic outputs. In this work, we will describe them in connection to their application to the saliency and occlusion sensitivity diagnostics. These diagnostics provide very intuitive visual explanations and are part of a larger group of diagnostics whose output is a heatmap (e.g., DeepLIFT and other relevance attribution methods), making their usage particularly useful in the demonstration of the developed methods.

In the case of saliency, the values in the heatmap pixels represent how much their variation affects the probability of the predicted class, whereas the occlusion diagnostic shows how the probability of the predicted class of an image changes when different regions are covered by a square of grey color.

The methodology used in this study of variability comprises three steps. First, multiple instances of the same model are generated and trained until they reach the same approximate accuracy. This is followed by the application of the interpretability methods and the measurement of the variability of the explanations in the two perspectives discussed above, per network and per image (see Figure 3.1 for examples of saliency and sensitivity to occlusion maps). To quantify differences between two explanations, the Euclidean distance between two heatmaps is used in the application cases described here. Finally, in the last step, using all the pairwise distances obtained for all the heatmaps produced using the sample of networks and the test dataset subsample, we analyse the variability of these at different levels of granularity.

A set of scripts was written to generate and train the necessary model instances for different architectures and datasets in an easy manner. A module is used as an input file such that all the parameters that change from one experiment to another are concentrated in a single place. The main module imports the source code of the input module whose name was passed as an argument (using the package *importlib*) and reads its parameters. It then checks for the completion status to understand if this is a restart run, obtains the appropriate train and validation dataset generators and loops through the network instance index, where a call to the training model function is done. Once the experiment is complete, the status file and a file where the run time is saved are updated. In the module *models_and_training.py*, all the functions required to create and train a network, as well as those relative to early stopping and the recording of metrics are implemented. The restart file contains the functions required to initiate and update both the status file and the file where the run time is recorded. This main set of scripts is independent of the dataset and model architecture used. The creation of the network instances and other tasks that are specific to a dataset or model architecture are handled through calls to general functions in the modules *model_methods.py* and *data_methods.py* that use some of the parameters of the input file that are passed as arguments to choose the appropriate specific functions for each dataset to call. The architecture of the network is in a separate file whose source code is imported from the module *model_methods.py*. In this way, to generate new networks for experiments all one needs to do is to create an architecture module, which only contains a function to create the structure of the untrained network and a fit function; and a dataset module, that contains the specific functions for the dataset in question. The interface between these two specific files and the main code is done through the general functions in the modules *model_methods.py* and *data_methods.py*. It should be noted that the function that created the model in the *model_methods.py* module imports the source code of the specific architecture module, so it does not need to be changed when supplying different network architectures.

For the second step, scripts were created for calculating the per network and per image variability, where only the heatmap generating function is different. The saliency map

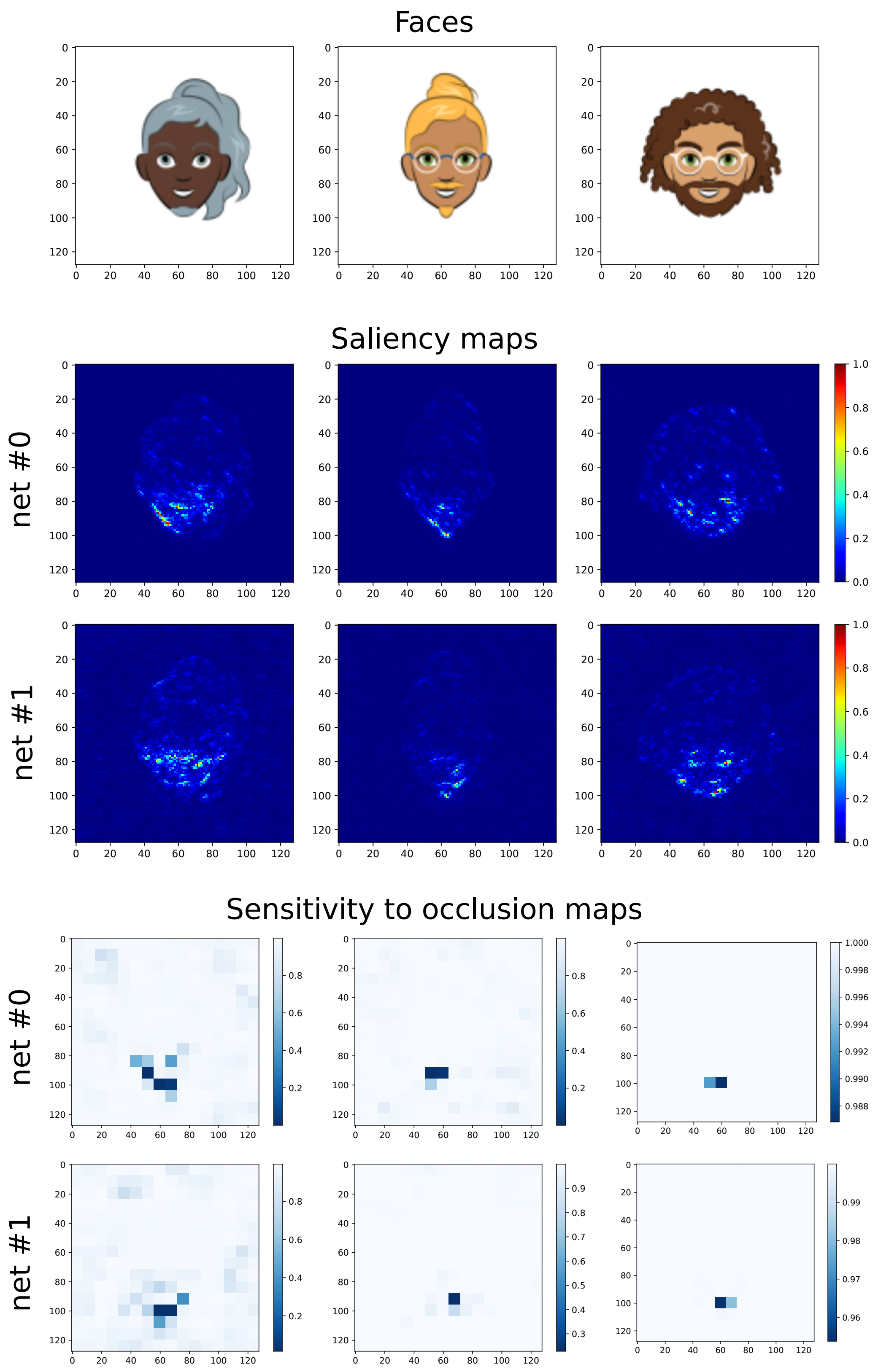


Figure 3.1: Examples of saliency and occlusion maps obtained for three of the one thousand random faces taken from the Cartoon test subset using two of the 200 networks with three convolution layers trained for classifying facial hair.

calculation technique was based on [64]). The variability calculation scripts have two versions, where the calculations performed are exactly the same but one is an optimized version that requires more memory. In these optimized versions all the network instances and heatmaps are created previous to the variability calculations which avoids some repetition of calculations.

It is important to note that, both the training of the networks and the calculation of the variabilities are quite computationally demanding. In particular, in the main part of the work, 200 network instances were trained for each combination of one of three convolutional neural network architectures and and four different tasks. This amounts to 2400 networks created and trained. In addition for each of the twelve combinations of architecture + classification task both the variability per network and the variability per image were calculated for a subset of 1000 test images. Since these calculations consist in extracting pairwise distances between heatmaps, they grow with the square of the number of images. Furthermore, the calculation of an occlusion sensitivity map grows with the square of the number of cells, when, as we did, a stride equal to the occlusion window side size was used. This limits the resolution which we can use. Because of the amount of computing hours required to complete the 24 variability calculations (variability per network and per image for each of the 12 combinations of architecture + task), running them in a laptop sequentially took too long. The final variability calculations, once the diagnostics were finalized, were run in Amazon Web Service instances to take advantage of launching them in parallel and took several hundreds of CPU hours.

In the following, we detail the methods used to obtain the pairwise heatmap distances, and to use the distribution of these distances to extract information on the variability of the networks.

Each heatmap matrix can be reshaped into a vector so one can compute distances between pairs of saliency or occlusion maps by taking a distance such as the Euclidean between the two vectors. This generates a great number of pairwise distances and we propose two ways to approach the data: looking at it by comparing the distributions of distances between all the images, for each network. We refer in the text to this approach as an analysis inside a network or per network. We also analyse the distributions of the distances between maps generated by different networks for each of sample images. We refer in the text to this approach as an analysis between networks or per image.

Different analysis are then possible with these pairwise distance distributions.

At a more aggregate level and to begin the exploration of the variability in the outcome of the diagnostic, one can take the average of pairwise distances per network in one case, and per image on the other, where we exclude the pairs of equal networks/images. This average of pairwise distances amounts to the calculation of the cohesion of clusters of points, in the first case considering a cluster for each network and in the latter, a cluster for each image.

The variability of the pairwise distances between networks for the same image (per image) is calculated by averaging the euclidean distances between the saliency maps for the same image i obtained from the set J of different networks:

$$C^i = \frac{1}{|J|(|J|-1)} \sum_{j \in J} \sum_{k \neq j, k \in J} d(m_j^i, m_k^i) \tag{3.1}$$

where j and k are indexes that run from 0 to $|J|-1$, m_j^i and m_k^i are one-dimensional vectors obtained from the reshaping of saliency map matrices of image i, and d is the Euclidean distance between two vectors.

The variability inside a network (per network) is calculated averaging the distances between the saliency maps obtained from this network for the subset of images:

$$C_j = \frac{1}{|I|(|I|-1)} \sum_{i \in I} \sum_{h \neq i, h \in I} d(m_j^i, m_j^h) \tag{3.2}$$

The distribution of these averages can also be plotted and analysed to see how the average of the pairwise distances of per network (for all the images, in each network) compares to the distribution of the averages of pairwise distances per image (among the different networks). This can give a first clue of how the variabilities in these two perspectives compare.

These variabilities, C^i and C_j, can also be averaged over the set of images / networks to obtain an aggregate variability between networks / images.

One can wonder if, in the case that the interpretability heatmaps such as saliency maps have a high number of cells, and there is noise for some images and not others, such differences can cumulatively lead to significantly different variabilities. In the case of saliency maps (which are normalized to one) and for the Cartoon dataset and the 3 architectures considered, this was tested with two thresholds for noise considered: pixels with saliency above 0.025 and 0.05. The first threshold is enough to eliminate most of the noise while the second does the same for images with higher noise but can remove some significant low saliency regions from the face area as well when applied to low noise images. For the lower value of 0.025, it was found that the average relative difference of the variabilities C^i and C_j and the relative difference of the aggregate variabilities with low threshold relative to using no threshold was below 5%. In the case of the higher threshold, the average relative difference of variabilities in both cases relative to using no threshold was always below 7%. It was also found that the relative difference of the ratio of aggregate variability between networks over the aggregate variability between images with threshold vs without was below 2% in both scenarios.

Another question that can arise is the effect of small translations of the images in the results of the variabilities between networks. While this was not tested and could potentially lead to differences in the location of relevant saliency regions, this would be indicative of inconsistencies of the explanations between different networks and therefore are not a problem, but part of what one is trying to examine.

At a more granular level, one can investigate whether different networks have similarly shaped distributions of pairwise distances (for pairs of maps from different images) or not. To investigate this, we start by producing kernel density estimations (KDEs) of the different distributions, one for each network, and plotting them. One can then observe whether the KDEs possess similar shapes for an architecture + task or there are several types of distributions (unimodal, multimodal, ...).

Distributions that are similar to a single Gaussian, for example, indicate the distances between the pairs of maps are similar in value. This is, however, not enough to infer similarity between the behaviour of the two networks, since it does not guarantee that the distances between each of the pairs of image maps are correlated for the different pairs. Close values could correspond to different pairs of image maps but still give the same distribution.

Distributions that are multimodal, for instance, could indicate that the image maps for the network are somehow organized in a more complex manner. A two mode distribution would indicate that the pairwise distances are close to either one of two different values. One example that could lead to such a distribution would be if the maps are clustered in two groups, with similar cohesions but apart from each other. Each cluster could have similar inside distances and pairwise distances between one member of each cluster would have a higher value. This is however, just an example of how differently shaped distributions could indicate a different pattern in how the saliency/occlusion maps are organized for different networks.

In an analogous way, one can take the pairwise distances from the maps generated from the same image by different networks and investigate whether there are groups of images whose maps are similarly distanced among different networks.

3.3 Datasets and models

3.3.1 Cartoon dataset and convolutional neural networks used with it

In the study detailed in this chapter, the cartoon dataset [65] was used in experiments with different convolutional neural network (CNN) model architectures and classification tasks. For faster results, the images were downsampled from 500×500 to 128×128 pixels. This dataset was chosen because it allows creating different experiments easily where simple CNNs can achieve high precision. A set of three reference model architectures were used and their parameters are detailed in tables 3.1, 3.2 and 3.3. For each of these models, two hundred instances were trained until an early stopping condition was reached: the training accuracy reached at least 95% and during the next 2 iterations there was no improvement greater than 2 % relative to the previous best training accuracy. The tasks chosen for this set of experiments were the classification of images of the cartoon faces according to the following 3 predefined categorical variables: face shape, facial hair, glasses, each with 7, 15 and 12 possible attributes respectively, described by an integer; and a custom category created by the author of faces with both reading glasses and certain facial hair styles (glasses of styles 0,2,3,4,5,6,7, among 12 possible and facial hair of one

Layer	activation	kernel size	stride	padding	# filters / # units	output shape
Input	-	-	-	-	-	128×128
Convolution	ReLU	3×3	(1,1)	same	4	$128 \times 128 \times 4$
MaxPooling	-	2×2	(2,2)	valid	-	$64 \times 64 \times 4$
Convolution	ReLU	3×3	(1,1)	same	8	$64 \times 64 \times 8$
MaxPooling	-	2×2	(2,2)	valid	-	$32 \times 32 \times 8$
Convolution	ReLU	3×3	(1,1)	same	16	$32 \times 32 \times 16$
MaxPooling	-	2×2	(2,2)	valid	-	$16 \times 16 \times 16$
Flatten	-	-	-	-	-	4096
Dense	ReLU	-	-	-	64	64
Dense	Softmax	-	-	-	# categories	# categories

# trainable parameters in feature extraction	1576
# trainable parameters in dense part	$262208 + 65 \times \#\text{categories}$

Table 3.1: Model 1 architecture parameters.

of the following types: 2,3,4,6,7,8,9,10,11,12,13, from the 15 possible).

In each set of experiments, the same classification tasks were performed but by a different set of models. Three different architectures were chosen, with three, four and five convolution layers, respectively. The images are coded in RGB values between 0 and 255 and were normalized to float values between -1 and 0. This choice of normalization was done so that white now corresponds to 0 in the three color channels. Since this is also the value used in padding, it was hoped that the absence of a sharp difference of values at the border of the images would somewhat mimic having reflective borders and decrease the artifacts observed in feature maps with the previous architectures.

Layer	activation	kernel size	stride	padding	# filters / # units	output shape
Input	-	-	-	-	-	128×128
Convolution	ReLU	3×3	(1,1)	same	4	$128 \times 128 \times 4$
MaxPooling	-	2×2	(2,2)	valid	-	$64 \times 64 \times 4$
Convolution	ReLU	3×3	(1,1)	same	8	$64 \times 64 \times 8$
MaxPooling	-	2×2	(2,2)	valid	-	$32 \times 32 \times 8$
Convolution	ReLU	3×3	(1,1)	same	16	$32 \times 32 \times 16$
MaxPooling	-	2×2	(2,2)	valid	-	$16 \times 16 \times 16$
Convolution	ReLU	3×3	(1,1)	same	32	$16 \times 16 \times 32$
MaxPooling	-	2×2	(2,2)	valid	-	$8 \times 8 \times 32$
Flatten	-	-	-	-	-	2048
Dense	ReLU	-	-	-	64	64
Dense	Softmax	-	-	-	# categories	# categories

# trainable parameters in feature extraction	6216
# trainable parameters in dense part	$131136 + 65 \times \#\text{categories}$

Table 3.2: Model 2 architecture parameters.

Layer	activation	kernel size	stride	padding	# filters / # units	output shape
Input	-	-	-	-	-	128×128
Convolution	ReLU	3×3	(1,1)	same	4	$128 \times 128 \times 4$
MaxPooling	-	2×2	(2,2)	valid	-	$64 \times 64 \times 4$
Convolution	ReLU	3×3	(1,1)	same	8	$64 \times 64 \times 8$
MaxPooling	-	2×2	(2,2)	valid	-	$32 \times 32 \times 8$
Convolution	ReLU	3×3	(1,1)	same	16	$32 \times 32 \times 16$
MaxPooling	-	2×2	(2,2)	valid	-	$16 \times 16 \times 16$
Convolution	ReLU	3×3	(1,1)	same	32	$16 \times 16 \times 32$
MaxPooling	-	2×2	(2,2)	valid	-	$8 \times 8 \times 32$
Convolution	ReLU	3×3	(1,1)	same	64	$8 \times 8 \times 64$
MaxPooling	-	2×2	(2,2)	valid	-	$4 \times 4 \times 64$
Flatten	-	-	-	-	-	1024
Dense	ReLU	-	-	-	64	64
Dense	Softmax	-	-	-	# categories	# categories

# trainable parameters in feature extraction	24712
# trainable parameters in dense part	$65600 + 65 \times \#\text{categories}$

Table 3.3: Model 3 architecture parameters.

Layer	activation	kernel size	stride	padding	# filters / # units	output shape
Input	-	-	-	-	-	32×32
Convolution	ReLU	3×3	(1,1)	same	64	$32 \times 32 \times 64$
MaxPooling	-	2×2	(2,2)	valid	-	$16 \times 16 \times 64$
Convolution	ReLU	3×3	(1,1)	same	128	$16 \times 16 \times 128$
MaxPooling	-	2×2	(2,2)	valid	-	$8 \times 8 \times 128$
Convolution	ReLU	3×3	(1,1)	same	256	$8 \times 8 \times 256$
Convolution	ReLU	3×3	(1,1)	same	256	$8 \times 8 \times 256$
MaxPooling	-	2×2	(2,2)	valid	-	$4 \times 4 \times 256$
Convolution	ReLU	3×3	(1,1)	same	512	$4 \times 4 \times 512$
Convolution	ReLU	3×3	(1,1)	same	512	$4 \times 4 \times 512$
MaxPooling	-	2×2	(2,2)	valid	-	$2 \times 2 \times 512$
Convolution	ReLU	3×3	(1,1)	same	512	$2 \times 2 \times 512$
Convolution	ReLU	3×3	(1,1)	same	512	$2 \times 2 \times 512$
MaxPooling	-	2×2	(2,2)	valid	-	$1 \times 1 \times 512$
Flatten	-	-	-	-	-	512
Dense	ReLU	-	-	-	4096	4096
Dense	ReLU	-	-	-	4096	4096
Dense	ReLU	-	-	-	1000	1000
Dense	Softmax	-	-	-	# categories	# categories

# trainable parameters in feature extraction	9220480
# trainable parameters in dense part	$22979560 + 1001 \times \#$categories

Table 3.4: VGG11 model parameters.

3.3.2 CIFAR05 dataset and VGG11

Applying our variability study to a single dataset could be limiting. In order to make our study more broad, the variability tools developed for our study were also applied to the convolutional neural network model VGG11 [63] and to a dataset we refer to as CIFAR05. The CIFAR10 dataset [57] contains photo-like images of ten categories (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck), and is therefore less synthetic than the Cartoon dataset. However, since for the simplest version of VGG11, whose parameters are described in table 3.4, the achieved accuracy during training was close to 80%, well below the accuracy achieved with the simpler CNNs described in the section before with the Cartoon dataset, we opted to eliminate the five categories for which the test error was greater. We refer to this dataset with only the categories automobile, frog, horse, ship, truck as CIFAR05.

As in the case before, two hundred instances of the model VGG11 were trained with an early stopping condition that required that the training accuracy reached at least 95% and that during the following 2 iterations there was no improvement greater than 2 % relative to the previous best training accuracy.

3.4 Variability analysis overview

The effect that networks trained with different random initialization have on the variability of saliency maps and occlusion maps interpretability diagnostics was investigated. This is important both to try to understand what they can tell us about the differences in network behaviour and if one can infer something about the interpretation of CNN models (network architecture + classification task) based on the analysis of a single instance.

In Figure 3.2 the variability between networks for the same image and for each network between images is plotted. Here we quantify the variability by taking the distribution of distances between saliency maps for networks/images for the same image/network, doing the average of this distribution, and then taking the average of these averages over the images/networks, as described in the methods section. This aggregate quantity gives an estimate on how distant are the saliency maps of different networks for a given example compared to the distance between maps for different examples for the same network.

Saliency maps and occlusion maps attempt to determine which parts of the image are more relevant for the prediction it makes given a certain example, though through different approaches. Intuitively then, one would expect that the distance between maps of different examples for the same network would, on average, be higher than the distances between maps of different networks for the same image. If all model instances show approximately the same high accuracy, it is expected that the attributes they value (here in pixel space) for classifying an example as a certain class should be more similar than attributes that are important to classify a different example, that could even be of a different class.

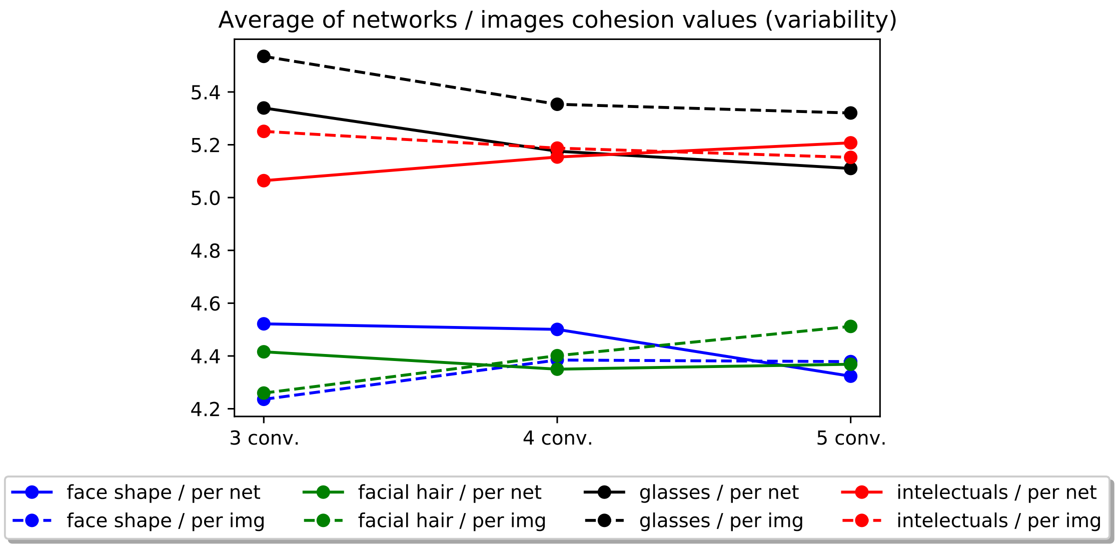


Figure 3.2: Variability of the saliency maps between networks for the same examples (per image) and between images within networks (per network).

However, when one examines Figure 3.2, it can be seen that, for different tasks and different architectures the two variabilities are either of the same order, or higher for maps of different networks for the same image (dashed lines). The latter case can be seen for all architectures considered for the task of classification of glasses and for the two deeper CNNs for the case of classifying facial hair and distinguishing intellectuals from non-intellectuals.

This constitutes a potential problem, since it can mean that the different network instances can be picking out attributes that are not very relevant for distinguishing different examples, or that the diagnostic is not very effective at capturing the importance of different regions of the image for the prediction.

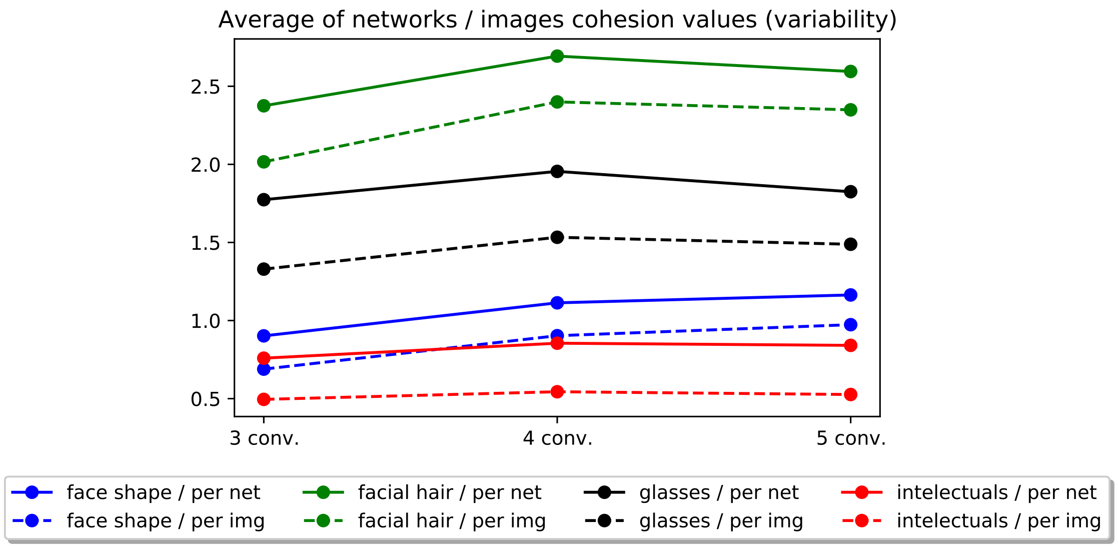


Figure 3.3: Variability of the occlusion maps between networks for the same examples (per image) and between images within networks (per network).

In Figure 3.3, it can be observed that both variabilities are of the same order, though in this case the variability between different examples for the same network is higher than the variability between occlusion maps for the same image from different networks. This could be due to the occlusion diagnostic being better at identifying attributes that distinguish different examples or an artifact of the size of the occlusion window used. For images of 128x128 pixels, a window size of 8x8 pixels was used due to the computational cost of decreasing its size.

To test whether a smaller window size would alter the relative positions of the average variability per image and per network, occlusion maps for the same networks of one of the models (glasses classification and 3 convolution layers) but a smaller random sample were produced for window sizes of 8x8 and 4x4, using 200 of the original 1000 images sample. The decrease in size of the sample was necessary since the computational cost of decreasing the window size goes with the square of the number of cells in each direction and the occlusion diagnostic is already, by itself, more computationally demanding than the saliency one because for each image, for each position of the window a new prediction needs to be done.

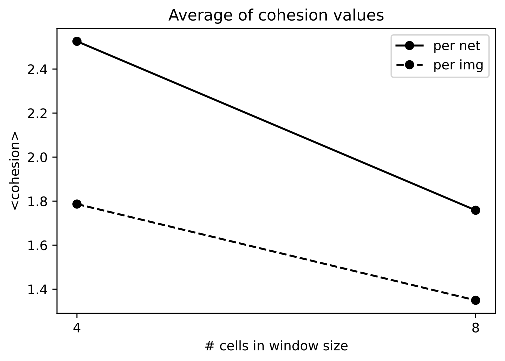


Figure 3.4: Variability of the occlusion maps between networks for the same examples (per image) and between images within networks (per network) for the 3 convolution layer architecture and the task of classification of glasses. The abcissa of the points is the number of cells in the side of the square occlusion window

In Figure 3.4, it can be seen that for a smaller window, the differences between occlusion maps from different networks for the same image reach higher values, which leads to an increase in its average value. However, the average value of the distances between occlusion maps from different images for the same network also increased. The variability of the occlusion maps between different images, for the same network network, remained higher than the variability in the occlusion maps for the same image, obtained from different networks. This result was obtained for a single network architecture and task. Further studies should be conducted in future work to better understand the impact of its size.

One can wonder whether the observed behaviour is affected by the class distribution within the sample, since, for some of the tasks, the class distribution is far from even, such as in the case for facial hair and distinguishing intellectuals from non-intellectuals. To investigate this, the same variability calculations were done for samples where each class is represented evenly (40 points per class), and are presented in Figures 3.5 and 3.6.

While the variabilities remain similar for the occlusion diagnostic, in the saliency diagnostic case they are now much more similar for the glasses classification and that of intellectuals vs non-intellectuals. For all but the facial hair case, in the saliency maps there is at least one architecture per task for which the variability is higher between networks for the same image than for different image maps produced by the same network. The face shape classification case remains almost the same as expected, since it already has an approximately even distribution of examples between its classes.

These differences between the saliency and occlusion diagnostics indicate that the variability between networks for the same image and between images for the same network is much more affected by the the distribution of classes than occlusion.

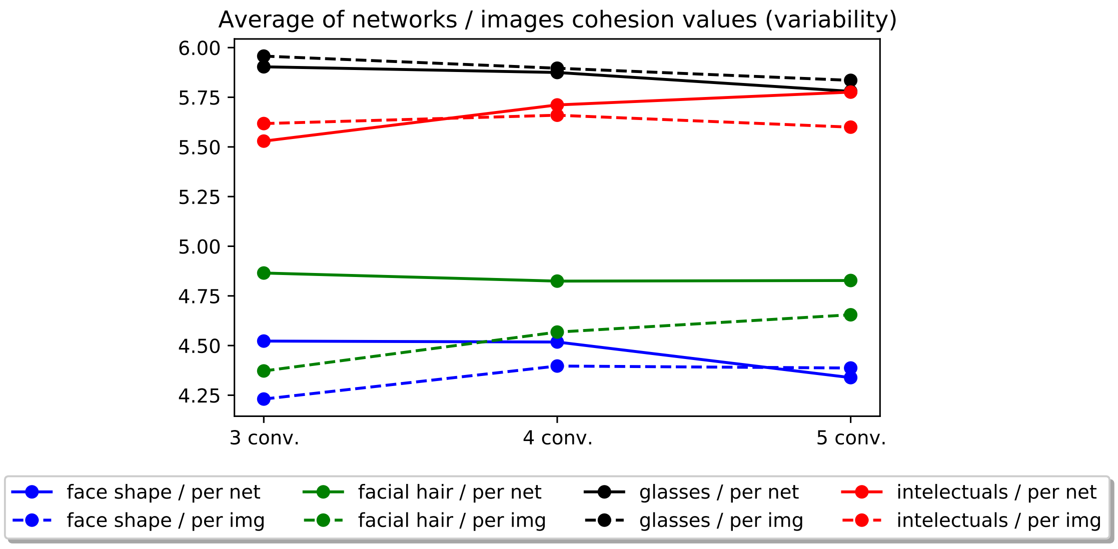


Figure 3.5: Variability of the saliency maps between networks for the same examples (per image) and between images within networks (per network) for a sample with even number of images among classes.

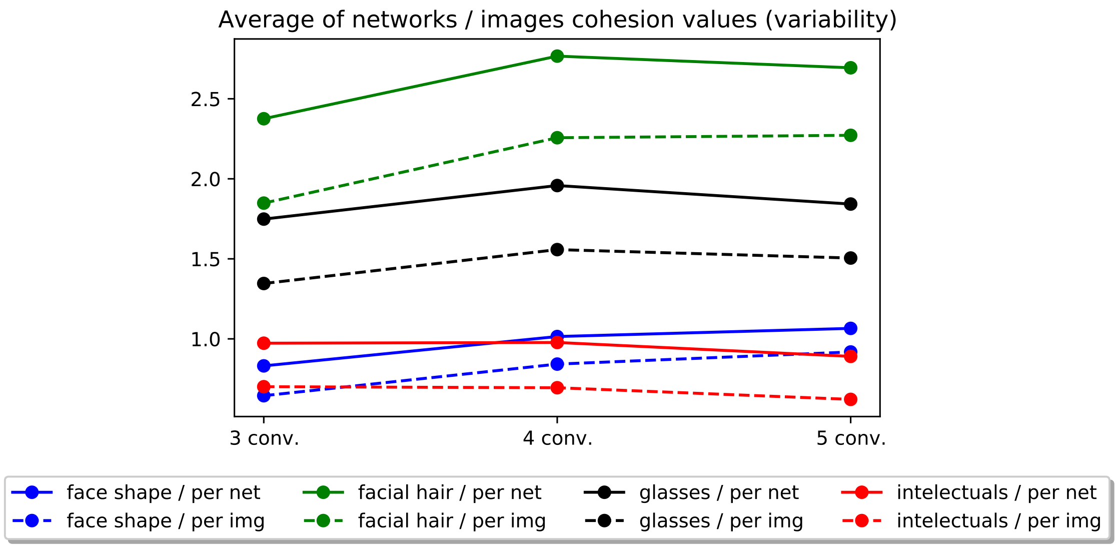


Figure 3.6: Variability of the occlusion maps between networks for the same examples (per image) and between images within networks (per network) for a sample with even number of images among classes.

3.5 Detailed variability analysis for the Cartoon dataset

3.5.1 Saliency for a random sample of 1000 images

3.5.1.1 Variability per network

The measure used to quantify variability in the previous section was an aggregate one. It does not tell us how the distances between maps are distributed for each network, between the images; or for each image, between the networks. It also does not show which shape do these distributions have and whether it is the same for all networks/images. To more easily explore this, we start by determining and plotting the kernel density estimation functions for each of these distributions.

The distributions of the distances between saliency maps from different images (all the 1000 images in the sample) for each network are plotted in Figures 3.7 and 3.8. Each line in a plot corresponds to one network. We would like to understand if the networks are distinct in terms of the attributes they pick up from different examples according to the saliency diagnostic. Do all networks have the same distribution of saliency maps distances? Are there groups of saliency maps distanced differently from one network to another?

In Figures 3.7 and 3.8, it can be seen that different tasks and different architectures lead to different patterns. In the case of the task of glasses classification, for example, a significant fraction of the distributions appear to be multimodal. The existence of more than one mode in the KDE distribution of a network indicates differently distanced groups of maps by that network. To verify this, we test for the modality of the KDE curves using the Hartigan Dip test of unimodality implemented in the Python package [66] in the different scenarios. For the cases where they are unimodal distributions, we also calculate their variances. These results are presented in Table 3.5.

For the face shape classification task, it was observed that the majority of the KDE curves are unimodal. It is interesting to note that, for all tasks, the average variance of the unimodal distributions increases from the three to the four convolution layer model. This indicates an increased spread in the distance values. A possible explanation for this could be that the networks with four layers capture more diverse features in the saliency maps, leading to more differences in the maps and therefore larger differences between them. However, this is just a hypothesis and would require further testing.

For the tasks of classification of facial hair, glasses and for the case of classification of intellectuals vs non-intellectuals with a five convolution layer model, there appears to be a significant number of multimodal KDE distributions. Their existence indicates that, for the corresponding networks, there is some kind of grouping or non-uniform spacing between saliency maps from different images.

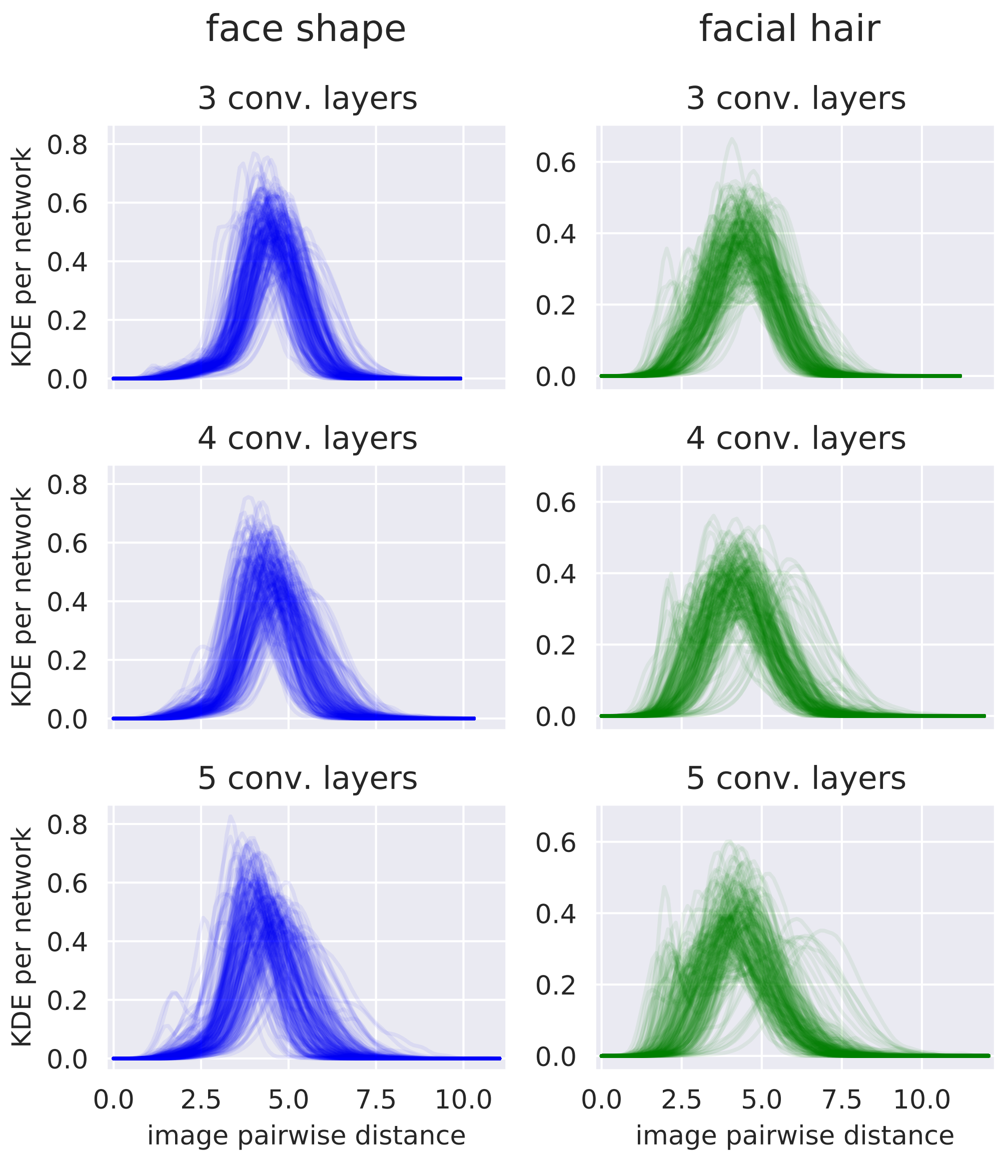


Figure 3.7: KDEs for each of the 200 network instances for the tasks of classification of facial shapes and facial hair. Each line represents the KDE of the distribution of pairwise distances between images for a single network. Plots in the same row are associated with the same model architecture.

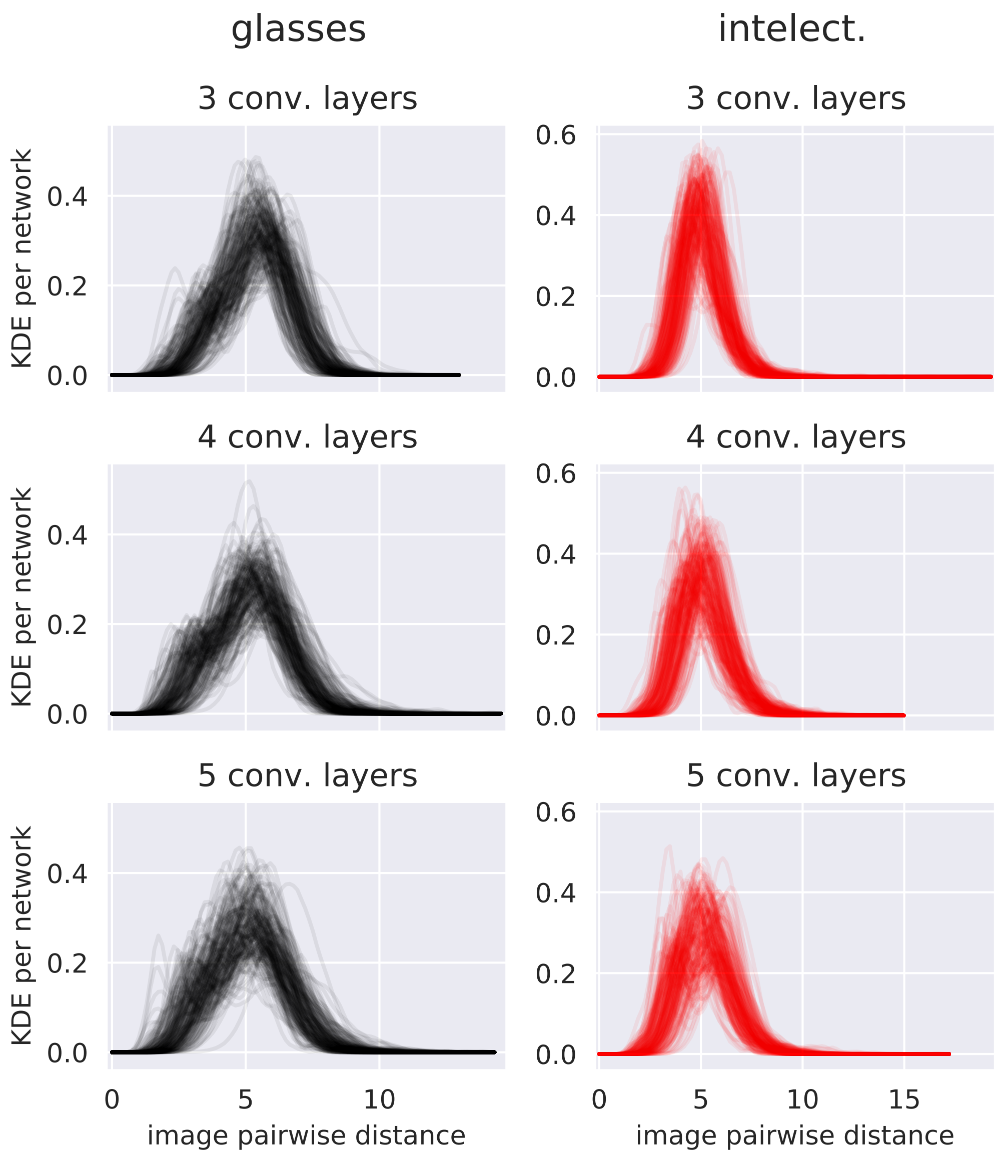


Figure 3.8: KDEs for each of the 200 network instances for the tasks of classification of glasses and intellectuals vs non-intellectuals. Each line represents the KDE of the distribution of pairwise distances between images for a single network. Plots in the same row are associated with the same model architecture.

Classification task	# of conv. layers	# unimodal KDEs	% unimodal KDEs	Average variance of distributions
face shape	3	188	94	0.69
	4	196	98	0.74
	5	192	96	0.71
facial hair	3	143	71.5	0.91
	4	153	76.5	1.02
	5	142	71	1.07
glasses	3	135	67.5	1.37
	4	89	44.5	1.58
	5	100	50	1.58
intelect.	3	189	94.5	1.06
	4	172	86	1.34
	5	146	73	1.43

Table 3.5: Modality and variances of unimodal distributions for the KDEs per network for the different classification tasks.

Classification task	# of convolution layers	Average pairwise correlation
face shape	3	0.426
	4	0.345
	5	0.334
facial hair	3	0.653
	4	0.605
	5	0.588
glasses	3	0.543
	4	0.591
	5	0.562
intelect.	3	0.452
	4	0.508
	5	0.514

Table 3.6: Average correlation between pairs of networks in the MDS projection. The correlation is between each pair of network vectors, whose elements are all the saliency maps pairwise distances for the different pairs of images, for the same network, as in Figures 3.10, 3.11,3.12 and 3.13.

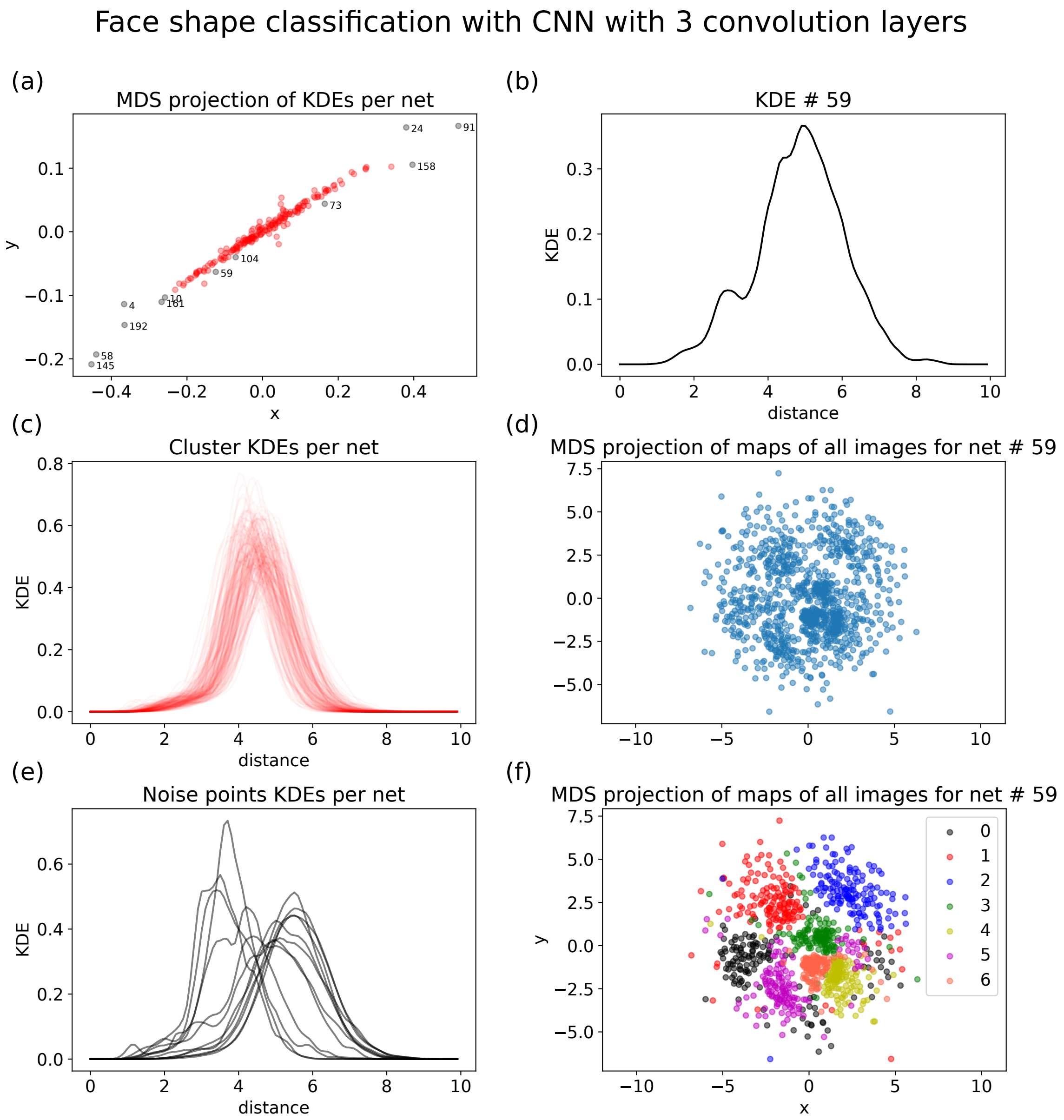


Figure 3.9: (a) MDS 2D projection of KDE curves; each point represents a network and they are colored red if they belong to the cluster and black if they are noise points. (b) KDE for network 59. (c) KDE curves corresponding to the points in the cluster. (d) MDS 2D projection of saliency maps obtained from network #59. (e) KDE curves corresponding to noise points the panel (a). (f) same MDS 2D projection as (d) but colored according to face shape seven classes.

Figure 3.9 presents the analysis of the KDE curves of the distribution of distances between saliency maps for each network in more detail for the task of face shape classification using the model with three convolution layers. This corresponds to the top left panel of Figure 3.7. In panel (a), the MDS projection of the correlation of the KDE distribution curves can be observed, with each point corresponding to a network. Clustering was performed with DBSCAN [67] using the elbow method to find the ϵ parameter ($\epsilon = 0.014$). It can be observed that, as expected from the overlap in KDE distribution curves seen in the top left panel of Figure 3.7, these distribution curves are indeed very similar, forming a single cluster (red dots) apart from a few noise points (black dots). The KDE distribution curves in the cluster are plotted in panel (c) and those from the noise points in panel (e). These outliers are observed to be a few multimodal distributions, or unimodal distributions with the peak at a different position than the KDEs of the main cluster.

To investigate the structure of one of these outlier KDE curves, we plot it in Figure 3.9 (b). Since it is multimodal, some structure in the space of the saliency maps for this network is expected. If, instead of the distribution of distances as in the KDE curve, we take the actual distances between all the saliency maps for this network, we can use them to display this saliency map space in 2D with a MDS embedding, seen in Figure 3.9 (d). Some structure is indeed clear with the presence of some higher density regions. Could these regions be associated with saliency maps from images with the same face shape class? Panel (f) of the same figure, where the points are colored according to their class, confirms this.

Since the clustering analysis of the KDE distribution curves per network for the face shape classification with the three convolution layers model (panel (a) of Figure 3.9) did not provide additional information, it was not repeated for the remaining models and tasks.

Given the significant degree of similarity in the KDE distribution curves of the distances between saliency maps of different images for the same network for the different tasks, one may wonder whether there is also similarity in the way the saliency maps are distributed in space for the different networks, i.e., whether the similarities are solely on the distribution of distances or the distances between the same pairs of images are correlated between the networks. This could suggest that the saliency attributes the networks capture for the same images are similar between networks. To test this hypothesis, we need to take the vectors with all the pairwise distances between images, for each of the networks, and determine the correlation between them. Since correlation can be used as a similarity measure, we can use it as before to do a MDS embedding and visualize if there are regions of higher density corresponding to highly correlated networks. This analysis was done and its results are presented in Figures 3.10, 3.11, 3.12 and 3.13, where each point represents a network. In all the cases (different tasks and model architectures) no internal structure with groups is observed, and the points are approximately uniformly distributed. The average correlation between each pair of points in the MDS projection (considering all the distances between saliency maps) was also determined and is presented in Table 3.6. The average correlation values are non-negligible. This, together with the absence of groups in the networks MDS projection (considering all distances between saliency maps; see bottom panels in Figures 3.10, 3.11, 3.12 and 3.13), could mean that there is a significant degree of similarity between networks. However, further investigation is needed to verify this.

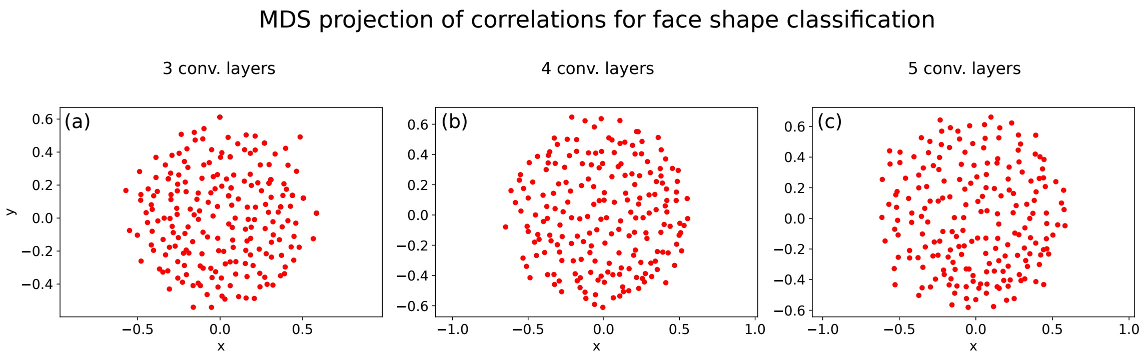


Figure 3.10: MDS 2D projection for face shape classification where each point corresponds to a network and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between pairs of saliency maps. Panel letters identify the CNN model architectures: (a) three, (b) four and (c) five convolution layers.

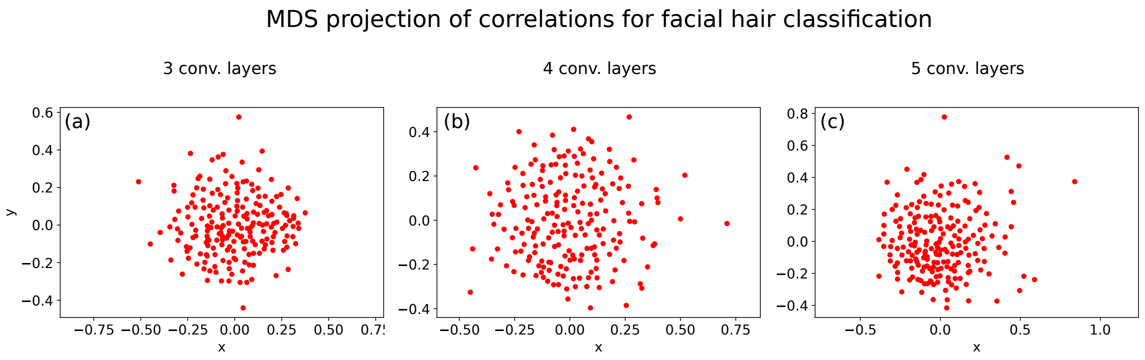


Figure 3.11: MDS 2D projection for facial hair classification where each point corresponds to a network and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between pairs of saliency maps. Panel letters identify the CNN model architectures: (a) three, (b) four and (c) five convolution layers.

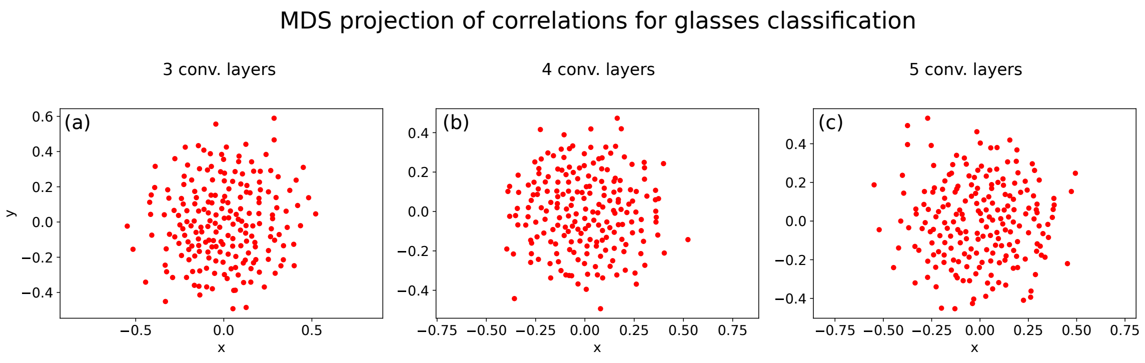


Figure 3.12: MDS 2D projection for glasses classification where each point corresponds to a network and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between pairs of saliency maps. Panel letters identify the CNN model architectures: (a) three, (b) four and (c) five convolution layers.

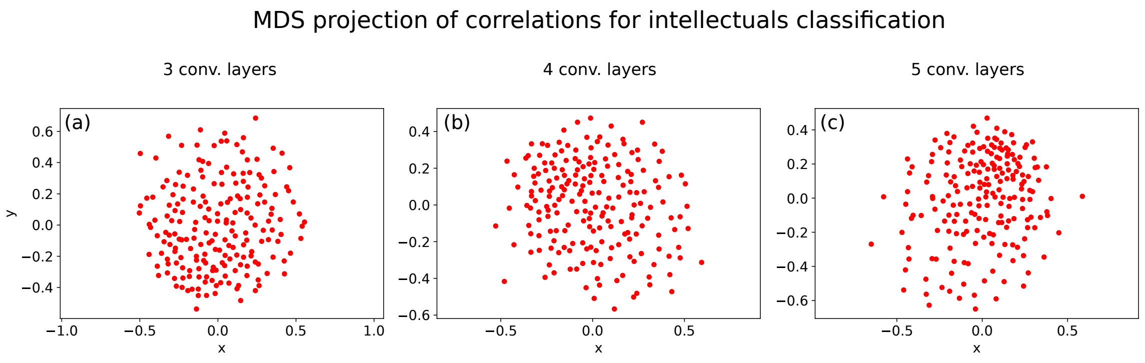


Figure 3.13: MDS 2D projection for intellectuals classification where each point corresponds to a network and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between pairs of saliency maps. Panel letters identify the CNN model architectures: (a) three, (b) four and (c) five convolution layers.

3.5.1.2 Variability per image

The previous analysis was focused on the variability between examples for the same network. We now focus on the variability between networks for the same image. We follow a similar exploratory approach as the above.

One can then ask what is the shape of the distributions of distances between maps generated by different networks for the same image. Are there groups of images with different distribution shapes? As a first step to investigate this, the KDE of the distributions of distances between maps obtained from different networks for the same image were obtained and are plotted in Figures 3.14 and 3.15. In each of these figures, each line in a plot corresponds to one image and the density curve describes the distribution of the distances between saliency maps for that image obtained from different networks.

For all the tasks and each of the architectures, the distributions appear mostly unimodal. This was confirmed with the Hartigan-dip test as before. The number of unimodal distributions and their variances are presented in Table 3.7. In the face shape case, the distributions corresponding to different images appear very similar as they are quite overlapped. In the remaining tasks, even though the distributions of distances remain unimodal, there is more variation in peak position. In the classification of glasses, groups of unimodal distributions with different peak positions are easily distinguishable, but the number of layers does not appear to change the pattern nor significantly affect the variances. In the other tasks, classification of facial hair and intellectuals vs non-intellectuals, the variability of the peak positions of the distributions appears to change significantly with increasing number of convolution layers. The fact that the peak position of these distributions is shifted for some groups of images, indicates that for some images the saliency maps are more similar between networks (less distant) than others.

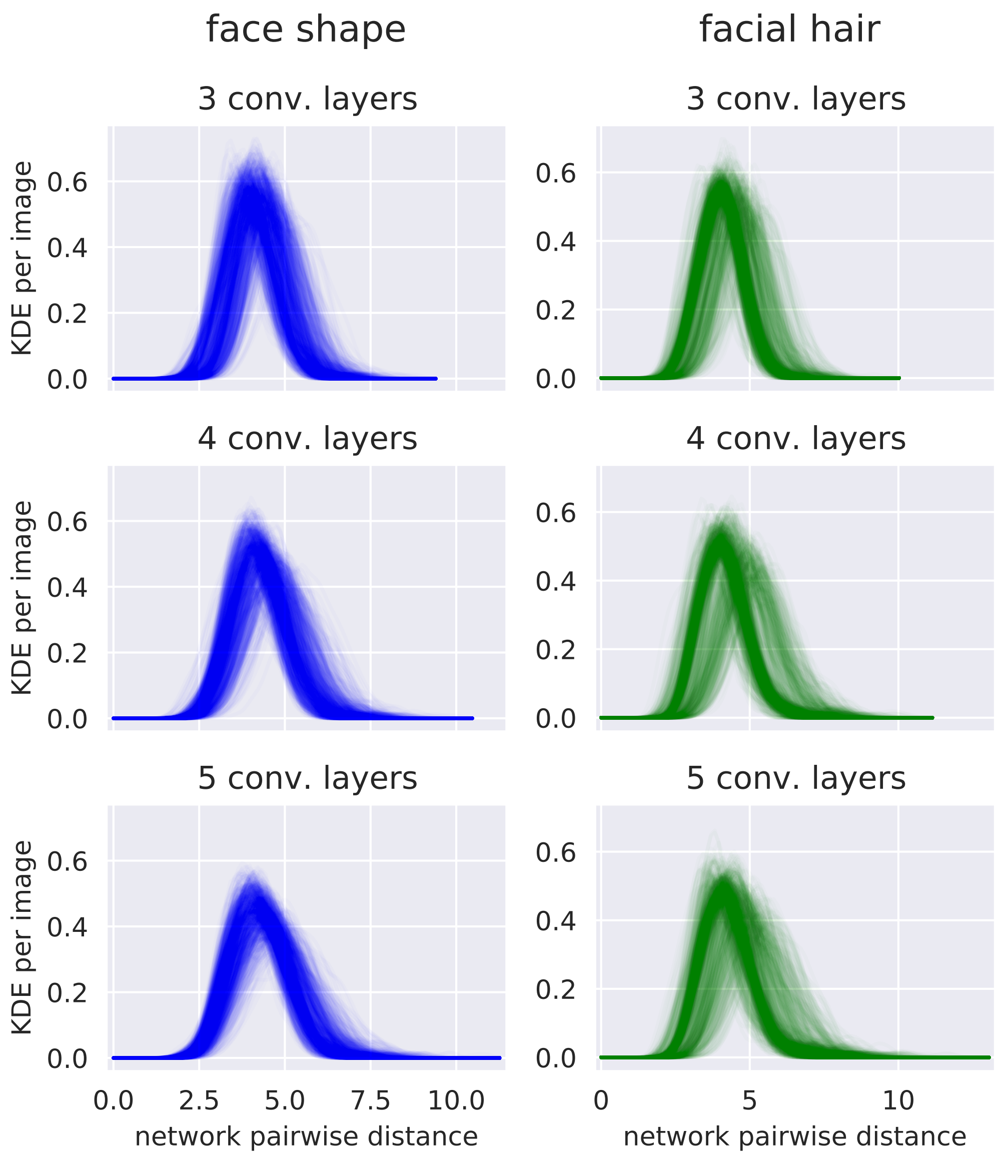


Figure 3.14: KDEs for each of the 1000 random images for the tasks of classification of facial shapes and facial hair. Each line represents the KDE of the distribution of pairwise distances between networks for a single image. Plots in the same row are associated with the same model architecture.

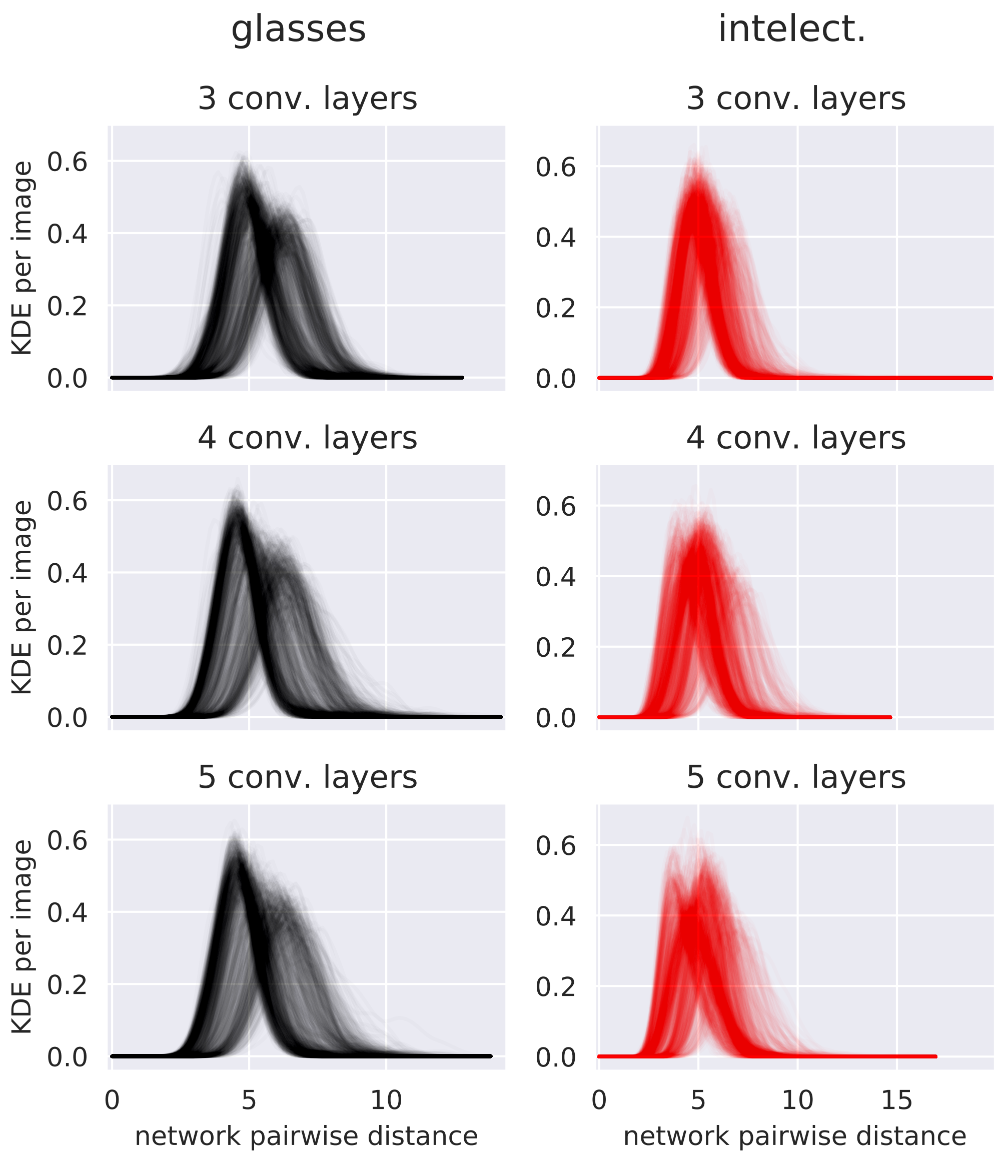


Figure 3.15: KDEs for each of the 1000 random images for the tasks of classification of glasses and intelect. Each line represents the KDE of the distribution of pairwise distances between networks for a single image. Plots in the same row are associated with the same model architecture.

For all the tasks but the classification of facial hair, the KDE distributions widen with increasing number of convolution layers, consistent with the observed values for the variances in Table 3.7.

The fact that different images share similar distributions of distances does not tell us how these pairwise distances are distributed among the different networks. For different images, are the relationships between distances of various pairs of networks kept the

Classification task	# of conv. layers	# unimodal KDEs	% unimodal KDEs	Average variance of distributions
face shape	3	1000	100	0.51
	4	997	99.7	0.65
	5	995	99.5	0.74
facial hair	3	999	99.9	0.54
	4	992	99.2	0.69
	5	942	94.2	0.81
glasses	3	1000	100	0.79
	4	999	99.9	0.77
	5	998	99.8	0.79
intelect.	3	999	99.9	0.69
	4	992	99.2	0.79
	5	995	99.5	0.91

Table 3.7: Modality and variances of unimodal distributions for the KDEs per image for the different classification tasks.

same; i.e., if two networks have a bigger distance between them than another pair, will the same happen for a different image. We do not expect so, as the images are different, and saliency features captured by the networks should not be the same for all of them.

To verify this, one must go beyond the distribution of distances and actually look, for each image, at the distances between each pair of networks. Using the vector with all these network pairwise distances for each image, we can calculate the correlation between images and use it as a measure of similarity. In Figures 3.16, 3.17, 3.18 the MDS projection of the images according to their correlation similarity is plotted. Each point corresponds to an image saliency map and they are colored according to their class label. As expected, we do see some groups of images clustered.

For the face shape task, the MDS projection of the images shows a clear structure and the images are mostly clustered according to their class label. This indicates that the distances between networks for the images of the same class are smaller than for images from different classes. The clusters also appear to become more compact for the five convolution layer. This clear separation between classes based on the vector of distances between networks for the same image, together with the non-negligible average correlation value between networks as represented by the vectors with the distances between the saliency maps obtained from them (table 3.6), could mean that the networks agree on the most important saliency features but noise or second-order features in the saliency maps are different for different networks, degrading the correlation between them.

For the tasks of facial hair and glasses classification, it is clear that the vast majority of networks have the saliency maps from one particular class closer together than with maps of images from other classes. This class corresponds to the majority of images in the sample and dataset for that category, no beard in the case of facial hair classification (label 14) and no glasses (label 11) in the task of glasses classification. However, and unlike the face shape task, the remainder classes are not so well separated in the MDS projection of the saliency maps of the images, even though some class grouping is still observed. This suggests that the saliency features from the other classes are not as similar inside a class and dissimilar with respect to maps from other classes for all the networks. One hypothesis for the class grouping being less pronounced would be that the networks pick up some common saliency attributes for saliency maps of the same class as well as other attributes which are not the same between networks and not as exclusive to a particular class. It is also interesting to note that the compactness of clusters of images from different classes changes with the number of layers, and not always in the same manner. The clusters of saliency maps of some classes (for e.g. class 0 in facial hair classification) are more compact in the models with lower number of convolution layers while for others it is the opposite (for e.g. class 12 in facial hair classification). The internal structure of the cluster with the majority of points also changes between different architectures. It would be interesting to investigate whether this corresponds to some feature being captured by some networks and not by others, if there is a semantic meaning to the subclusters observed.

In the case of the classification of intellectuals (class 1) vs non-intellectuals (class 0), it can be observed in Figure 3.19 that the different networks separate well a large portion of the population of non-intellectuals, in terms of the saliency map features. However, a second cluster of saliency maps for the non-intellectuals appears close and mixed with the points corresponding to saliency maps from intellectuals. This task is somewhat complex since the distinction between intellectuals and non-intellectuals is based on the type of facial hair and glasses. The intellectuals have combinations of some of the classes from these two categories. However, a certain type of facial hair or glasses can occur in both populations. It would be interesting to understand what distinguishes the two main clusters of saliency maps of the non-intellectual population, the one clearly separated and the one that appears close and mixed with the intellectual saliency maps clusters.

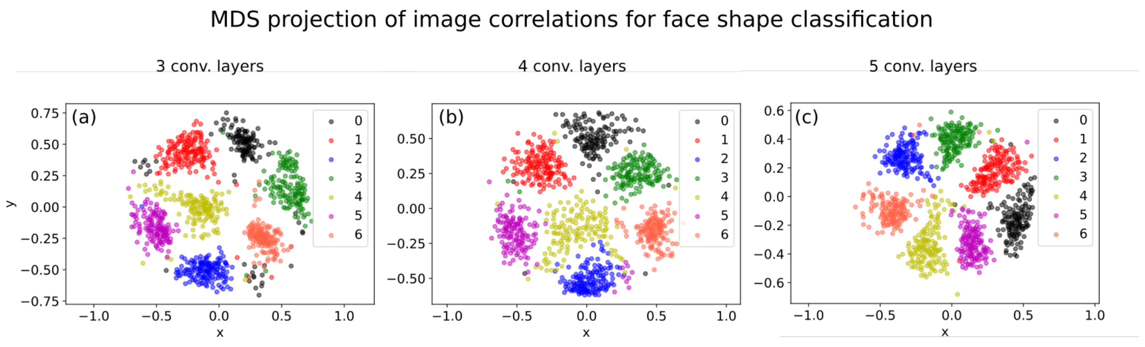


Figure 3.16: MDS 2D projection for face shape classification where each point corresponds to an image and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between pairs of networks. Points are colored according to class label. Panel letters identify the CNN model architectures: (a) three, (b) four and (c) five convolution layers.

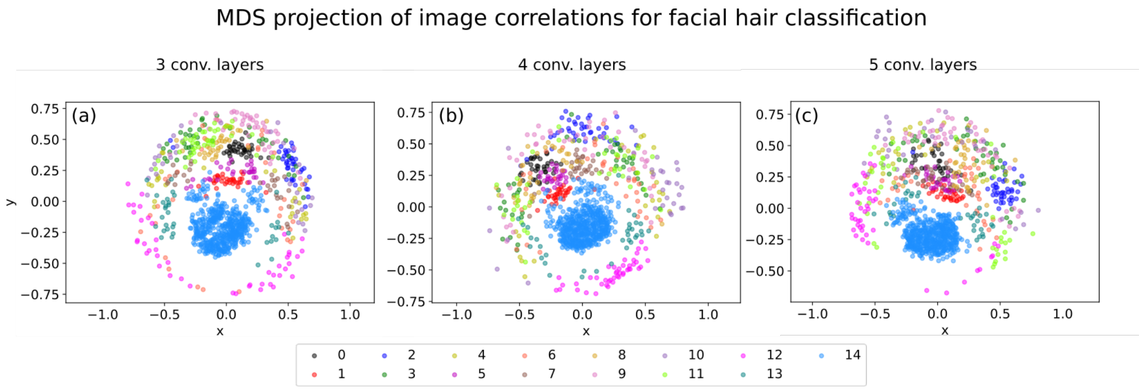


Figure 3.17: MDS 2D projection for facial hair classification where each point corresponds to an image and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between pairs of networks. Points are colored according to class label. Panel letters identify the CNN model architectures: (a) three, (b) four and (c) five convolution layers.

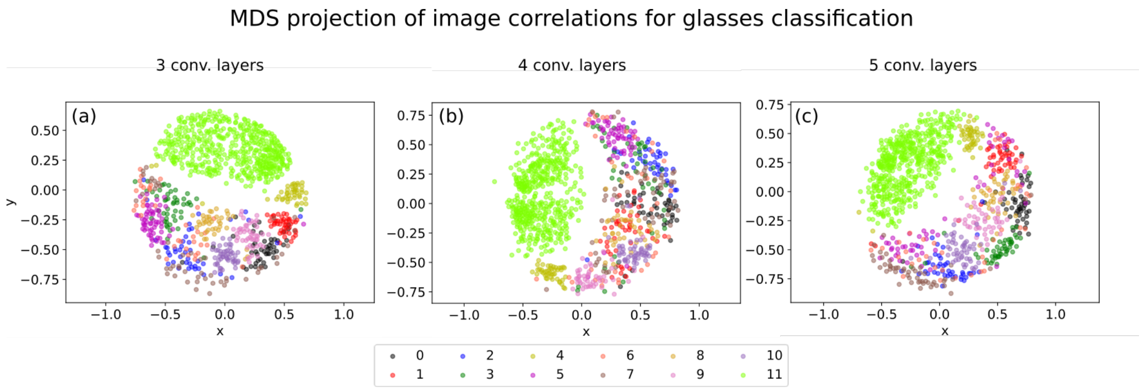


Figure 3.18: MDS 2D projection for glasses classification where each point corresponds to an image and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between pairs of networks. Points are colored according to class label. Panel letters identify the CNN model architectures: (a) three, (b) four and (c) five convolution layers.

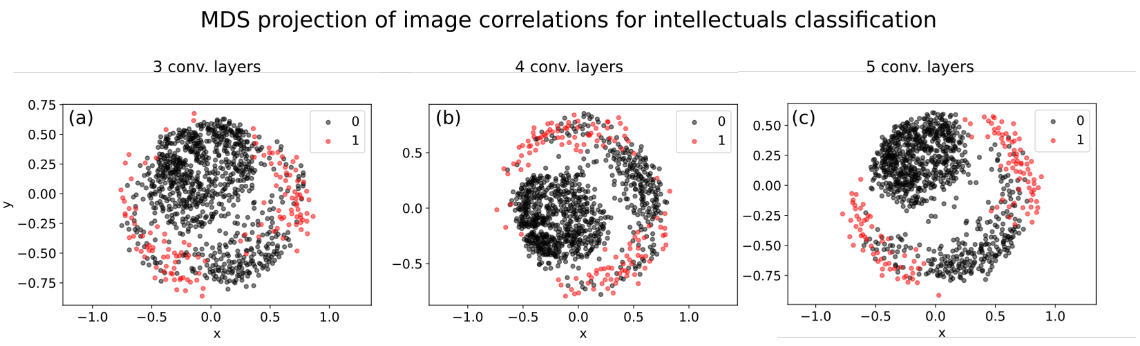


Figure 3.19: MDS 2D projection for intellectuals classification where each point corresponds to an image and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between pairs of networks. Points are colored according to class label. Panel letters identify the CNN model architectures: (a) three, (b) four and (c) five convolution layers.

3.5.2 Occlusion for a random sample of 1000 images

For the study of the variability for the same network between different examples and for the same image between different networks, we started the exploratory analysis in a manner analog to what was used in the case of saliency.

3.5.2.1 Variability per network

In this way, we start by looking at the distribution of distances between examples for the same network. Again, we do this by determining the KDE curve for the distribution of these distances for each network. The KDE distributions are then displayed in the same plot for comparison, and the procedure is repeated for each task, and within it, for each model architecture. The results are displayed in Figures 3.20 and 3.21.

When analysing the KDE distribution plots, one thing that immediately stands out is that for the tasks of classification of face shape, glasses and intellectuals vs non-intellectuals, there is a peak at a distance value very close to zero for practically all the curves. This indicates that there are groups of occlusion maps that are nearly identical. This could be the result of an insufficient resolution in the occlusion window, or the regions that significantly influence the prediction being quite limited in space and consistent for the same task, for example, the two sides of the cheeks in the face shape case. The KDE distributions are not unimodal for these tasks and other peaks are observed in varying number. In the case of glasses classification several peaks are observed, suggesting structuring in the space of the occlusion maps, with different groups of maps appearing for different networks. The overlap is not as high as in the case of the KDEs for the face shape and intellectuals vs non-intellectuals classification, which could mean the grouping of the occlusion maps is more consistent for the latter tasks than for the glasses classification, where the structure changes from one KDE to another.

In contrast with the other tasks, the KDE distribution curves for the case of classification of facial hair appear approximately unimodal for the vast majority and centered at a distance close to two. This suggests that the occlusion diagnostic in this task is able to detect more differences between the examples in terms of how the occlusion of different regions of the image affects the class prediction.

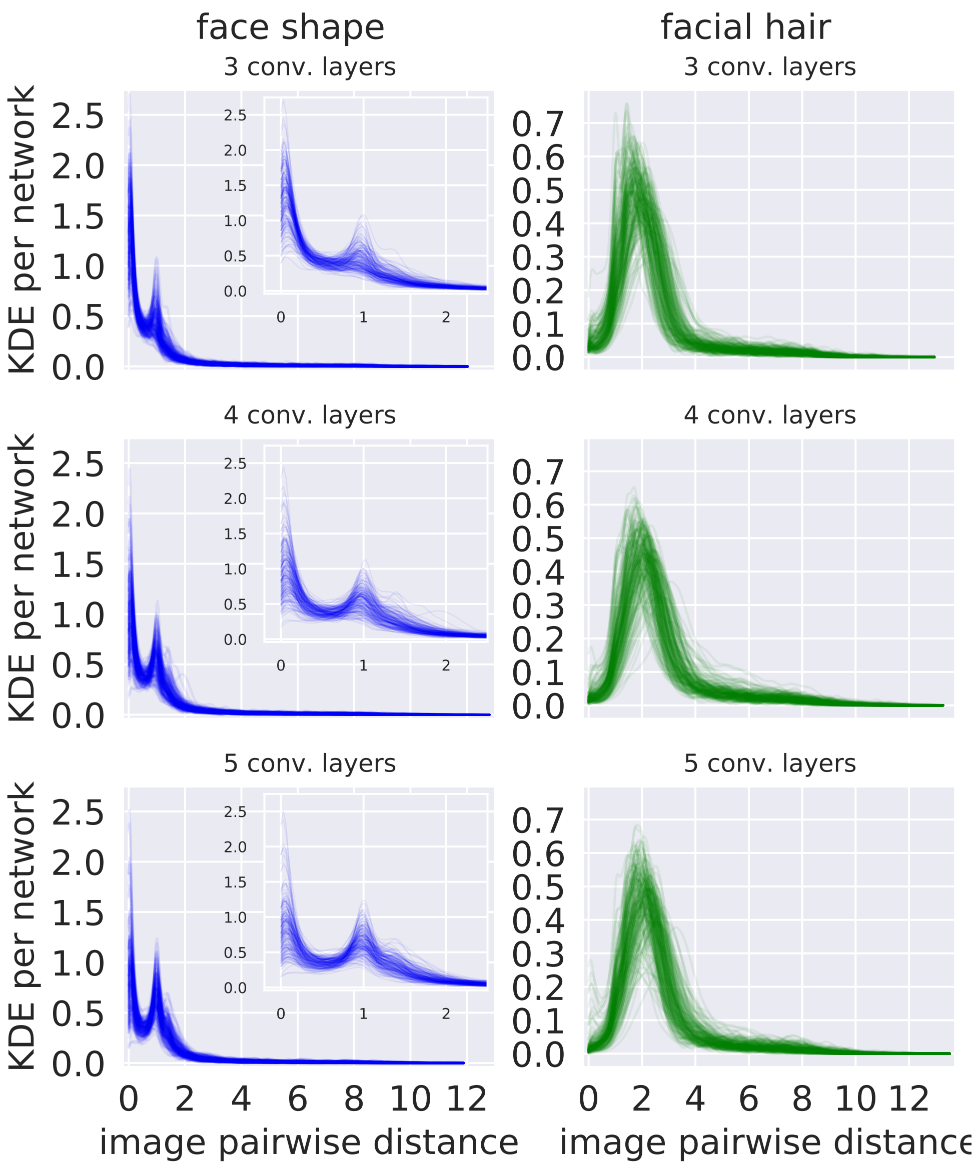


Figure 3.20: KDEs for each of the 200 network instances for the tasks of classification of facial shapes and facial hair. Each line represents the KDE of the distribution of pairwise distances between images for a single network. Plots in the same row are associated with the same model architecture.

It is interesting to note in the KDE distribution curve plots for face shape classification

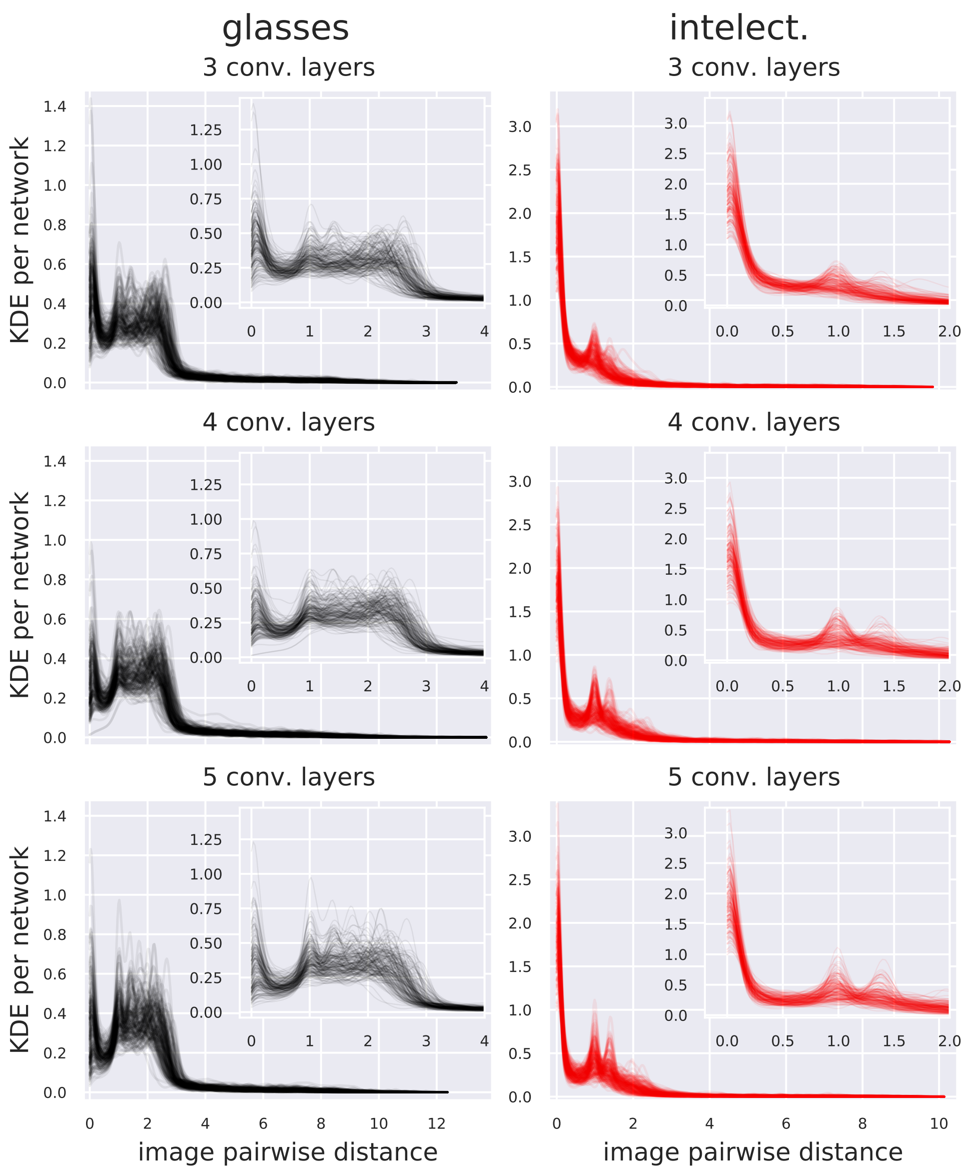


Figure 3.21: KDEs for each of the 200 network instances for the tasks of classification of glasses and intellectuals vs non-intellectuals. Each line represents the KDE of the distribution of pairwise distances between images for a single network. Plots in the same row are associated with the same model architecture.

Classification task	# of convolution layers	Average pairwise correlation
face shape	3	0.451
	4	0.491
	5	0.474
facial hair	3	0.396
	4	0.383
	5	0.392
glasses	3	0.512
	4	0.452
	5	0.425
intelect.	3	0.555
	4	0.548
	5	0.557

Table 3.8: Average correlation between all pairs of networks. The correlation is between each pair of network vectors, whose elements are all the saliency maps pairwise distances for the different pairs of images, for the same network

and classification of intellectuals vs non-intellectuals that, with the increase in the number of convolution layers, the structure of the distributions also changes. In the case of face shape classification, the average height of the peak close to zero decreases and that of the peak close to a distance of one increases. In the case of classification of intellectuals vs non-intellectuals, the average height of the secondary peaks at an approximate distance of one increases and more networks exhibit a more pronounced third peak at a distance around 1.4 for the 5 convolution layer network. These peaks at higher image pairwise distance indicate that the deeper networks reveal more differences between image occlusion maps than the more shallow.

In Figures 3.20 and 3.21, a strong overlap of the KDE distribution curves for the classification of face shape, facial hair and intellectuals vs non-intellectuals can be observed. However, this is not enough to say that the different networks behave in the same manner in terms of the occlusion sensitivity. Is the structure of the space of occlusion maps actually similar between networks? In other words, do the relationships between the distances of the different pairs of image occlusion maps change from one network to another? As was done with the saliency maps in the previous section, to answer this question we need to look beyond the shape of the occlusion maps pairwise distance distribution, and take into account the distances for each pair of maps of the sensitivity to occlusion. To do it, we represent each network by a vector whose elements are the pairwise distances of all the pairs of occlusion maps, preserving the ordering of the pairs for all the networks. We then use the correlation between these vectors to assess if the relationships between pairs of occlusion maps are maintained from one network to another. To visualize if there are groups of networks for which this happens, i.e., that share similar patterns in the space of occlusion maps, we do a MDS projection of the networks where we use the correlation between their vectors of distances as a similarity measure. These projections are presented in Figures 3.22, 3.23, 3.24 and 3.25. The figures do not reveal clear clusters of networks, instead showing a mostly uniform pattern. The average correlation value between networks was also determined and is shown in table 3.8. The fact that the average pairwise correlations are non-negligible together with the uniformity of the projection where the correlation was used as a measure of similarity, suggest the networks share, to some degree, the same behaviour in what regards the sensitivity to occlusion of the images.

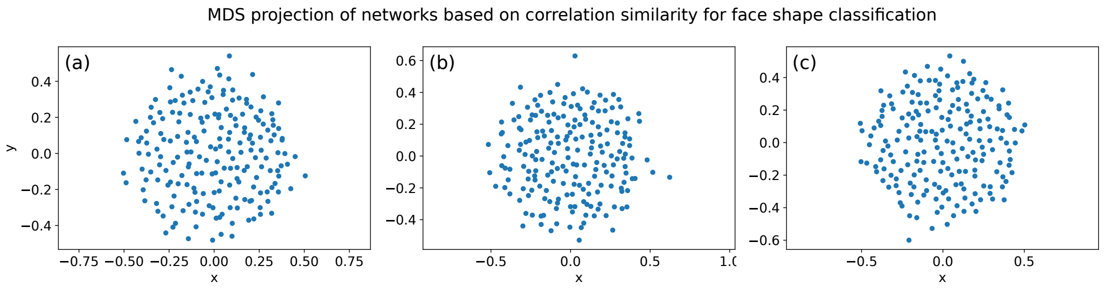


Figure 3.22: MDS 2D projection of the networks for face shape classification. Each point corresponds to a network and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between pairs of occlusion maps. The three panels correspond to the results of the (a) three, (b) four and (c) five convolution layers models.

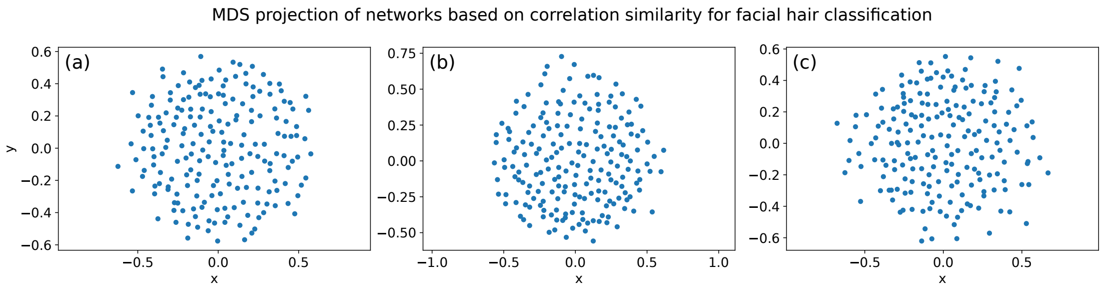


Figure 3.23: MDS 2D projection of the networks for facial hair classification. Each point corresponds to a network and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between pairs of occlusion maps. The three panels correspond to the results of the (a) three, (b) four and (c) five convolution layers models.

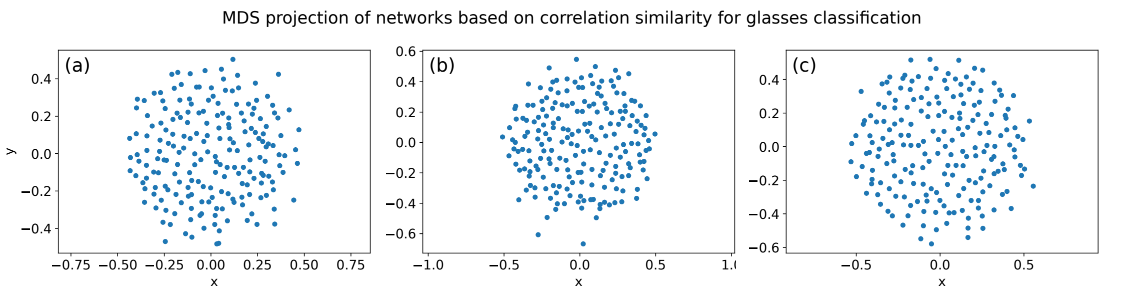


Figure 3.24: MDS 2D projection of the networks for glasses classification. Each point corresponds to a network and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between pairs of occlusion maps. The three panels correspond to the results of the (a) three, (b) four and (c) five convolution layers models.

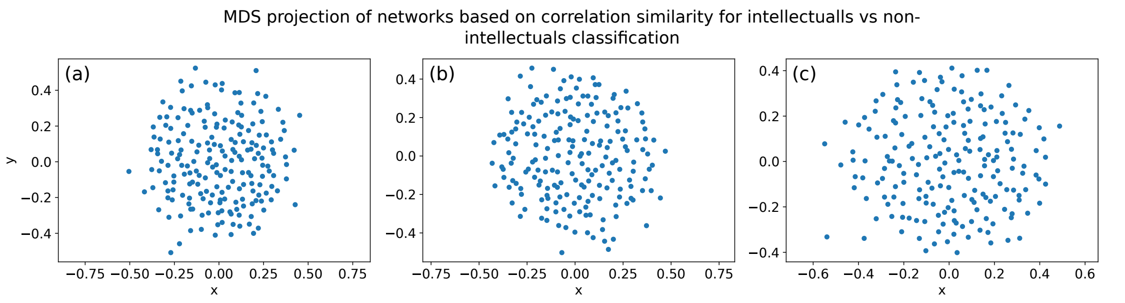


Figure 3.25: MDS 2D projection of the networks for intellectuals vs non-intellectuals classification. Each point corresponds to a network and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between pairs of occlusion maps. The three panels correspond to the results of the (a) three, (b) four and (c) five convolution layers models.

3.5.2.2 Variability per image

In this section we investigate in more detail the distribution of distances between occlusion maps produced by different networks for the same image. With this goal, we started by plotting the KDE curves for these distributions of distances, which are shown in Figures 3.26 and 3.27, where each line is the KDE curve for a single image.

Looking at the KDE distribution curves for the different tasks, one thing immediately stands out. For all the classification tasks, and especially for the shallower networks, with three convolution layers, there is a very high spike close to zero. This high density spike close to zero indicates that there are a number of images for which a very significant part of the networks produce an identical or nearly identical occlusion map. The peak becomes smaller for deeper networks and is specially high for the classification of intellectuals vs non-intellectuals.

It is also interesting to note that the task for the classification of facial hair has the lowest number of images that have a very high peak close to zero. For the majority of images, the KDE appears to be a unimodal or close to unimodal curve centered at a distance of two.

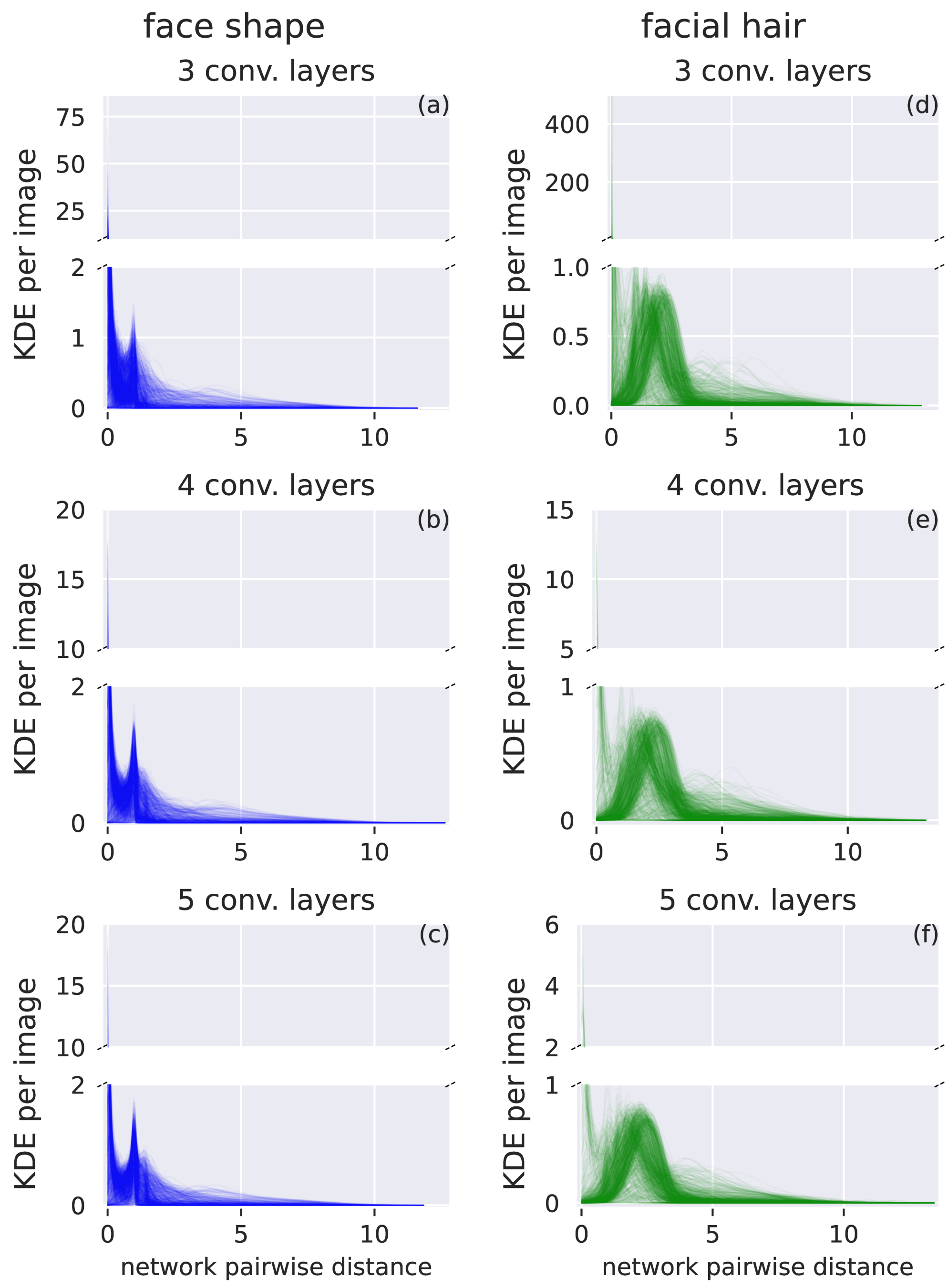


Figure 3.26: KDEs for each of the 1000 random images for the tasks of classification of facial shapes and facial hair. Each line represents the KDE of the distribution of pairwise distances between networks for a single image. Plots in the same row are associated with the same model architecture.

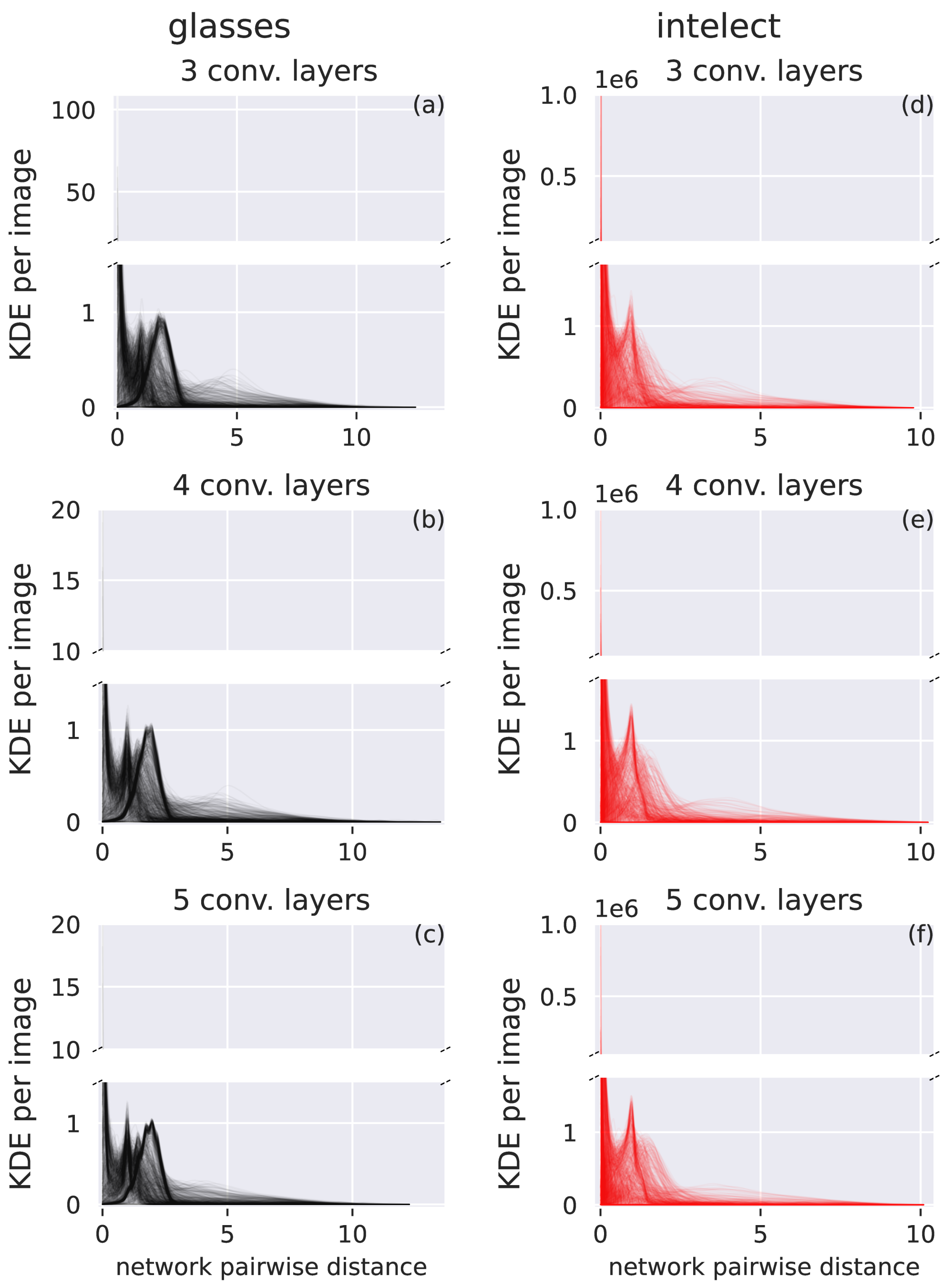


Figure 3.27: KDEs for each of the 1000 random images for the tasks of classification of glasses and intelect. Each line represents the KDE of the distribution of pairwise distances between networks for a single image. Plots in the same row are associated with the same model architecture.

For all the tasks, there appear to be groups of images for which the KDEs overlap significantly, i.e., have very similar distribution curves. To verify whether, for these groups of images, the relationships between networks are the same, i.e., if for one image, a pair of networks has a higher distance than another pair, the same will happen for other images of the same group, we apply the same technique as with the variability of the saliency maps per image. For each image, we build a vector with all the distances between the occlusion maps for that image obtained from all the unique pairs of networks. Using the correlation between this vectors as a similarity measure, we plot the MDS projection of these vectors, presented in Figures 3.28, 3.29, 3.30 and 3.31. In these plots, each point corresponds to an image and is colored according to its class. Close points correspond to images for which the relationships between the distances of pairs of occlusion maps obtained from different networks are maintained. Each point is colored according to its class. In other words, they correspond to images for which the different pairs of networks present a similar relative behaviour in terms of the differences between their occlusion maps.

For the task of face shape classification, clustering of points corresponding to images of the same class are observed. In addition, the compactness of the clusters for the four convolution layer network appears to increase. This clustering suggests that the networks relative pairwise distances for the occlusion maps of images of the same class agree more than for images of different classes. In other words, the behaviour of the networks in terms of the differences between occlusion maps is more similar for images of the same class than between images of different classes.

In the MDS projections for the cases of classification of the facial hair and glasses, the situation is different. The networks behaviour in terms of distances between occlusion maps they produce for the same image seems to be similar for the class with most elements (no facial hair and no glasses respectively). However, for other classes, though there is some clustering for some classes, for example class 2 in the facial hair classification and class 1 in the glasses classification, overall the separation of classes is much less pronounced and there is more mixing between points associated with images of different classes than in the classification of face shape. This indicates that the patterns of the maps in the occlusion maps space are not similar enough between images of the same class and different enough between different classes for them to be separable. In other words, there is not sufficient regularity in the relative differences of occlusion maps for the different pairs of networks.

For the task of classification of intellectuals vs non-intellectuals, the two classes appear separated and the compactness of the class with less elements, the class of intellectuals, appears to become more compact with the increase in the number of convolution layers. This suggests that the networks in general find similar attributes in the sensitivity to occlusion maps for the images of the same class. In addition, with increasing layer the networks appear to consistently find attributes in the occlusion maps from the two classes that distinguish them, since considering the similarity between vectors with all the distances between pairs of networks for each image, lead to a separation in the two classes.

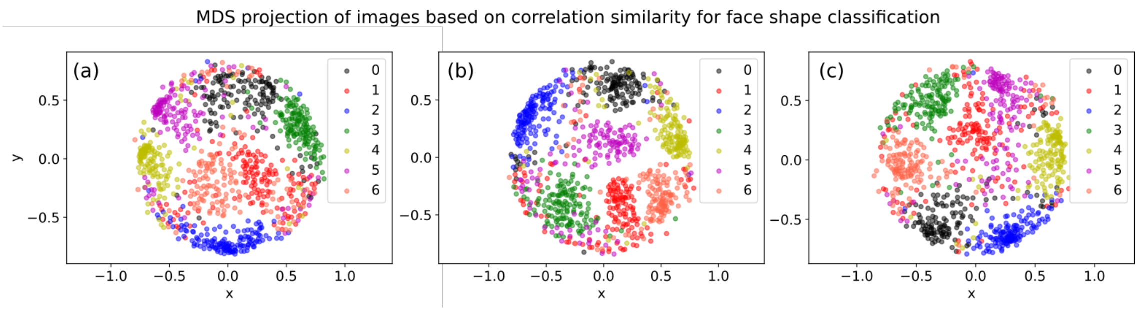


Figure 3.28: MDS 2D projection of the images for face shape classification. Each point corresponds to an image and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between the occlusion maps from all the different pairs of networks. The three panels correspond to the results of the (a) three, (b) four and (c) five convolution layers models.

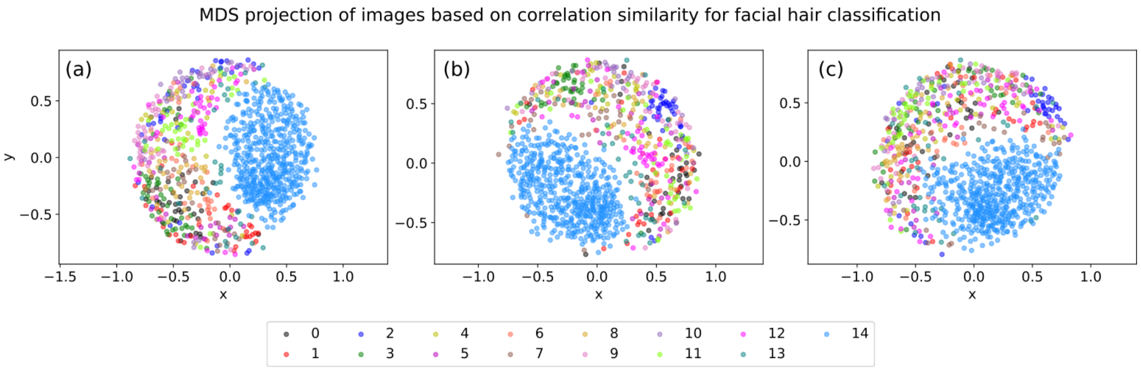


Figure 3.29: MDS 2D projection of the images for facial hair classification. Each point corresponds to an image and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between the occlusion maps from all the different pairs of networks. The three panels correspond to the results of the (a) three, (b) four and (c) five convolution layers models.

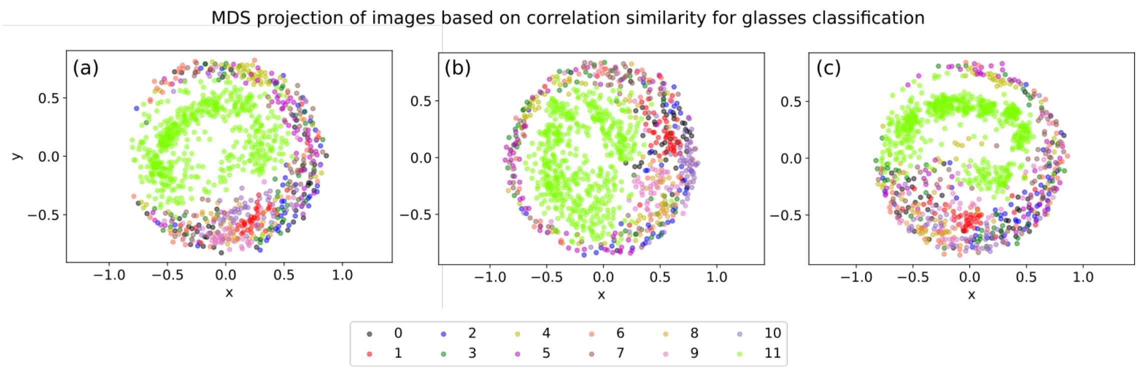


Figure 3.30: MDS 2D projection of the images for glasses classification. Each point corresponds to an image and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between the occlusion maps from all the different pairs of networks. The three panels correspond to the results of the (a) three, (b) four and (c) five convolution layers models.

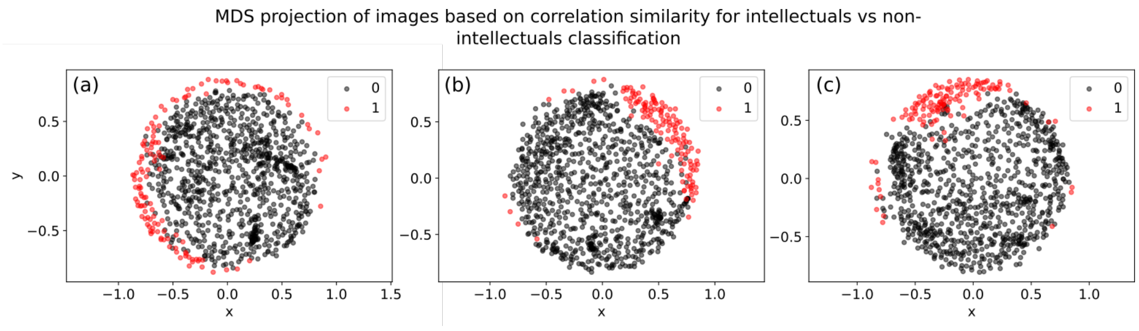


Figure 3.31: MDS 2D projection of the images for intellectuals vs non-intellectuals classification. Each point corresponds to an image and the dissimilarity is calculated as one minus the correlation between the vectors with all the distances between the occlusion maps from all the different pairs of networks. The three panels correspond to the results of the (a) three, (b) four and (c) five convolution layers models.

3.6 Preliminary results on the variability analysis for the CIFAR05 dataset and the VGG11 model

In this section, preliminary results on the variability analysis for the VGG11 model and CIFAR05 dataset are presented. A more detailed analysis with more trained instances and additional data exploration if left for future work.

3.6.1 Saliency for a random sample of 500 images for the CIFAR05 dataset

In an analogous manner to what was done for the Cartoon dataset and the several models considered in the previous section, we start the variability analysis for the VGG11 model trained on the CIFAR05 dataset by determining its aggregate variability per network and per image. For the first case, this means determining for each network the average of the pairwise distances between the saliency maps for different images, and then taking the average of these values over the number of networks. Similarly, for the aggregate variability per image, we first determine the average of the pairwise distances between saliency maps from different networks for each image, and then take the average of these values over the number of images. The results obtained are shown in Table 3.9.

Average variability per image	4.39
Average variability per network	5.49

Table 3.9: Average variability per image and per network for 80 instances of the VGG11 model and a sample of 500 images from the CIFAR05 dataset.

The average variability per network is greater than the average variability per image. The VGG11 model is, however, much more complex than the models used in the previous section, which results in a greater training time. For this reason, a more limited number of network instances was used. A more detailed analysis of the variability of the saliency maps also reveals some unexpected results.

The distributions of distances between saliency maps of different images for the same network; and between saliency maps of the same image obtained from different networks, was approximated using kernel density estimation. The plots of these distributions are shown in Figure 3.32, for each network and for each image, respectively.

In the left panel of Figure 3.32, it can be seen that there is a great overlap between the KDE curves for the distributions of image pairwise distances for each network. To try to understand whether only the pairwise saliency maps distributions are close from one network to another, or if the actual disposition of the maps in the saliency map space is approximately the same, i.e., the relationship of between distances of the image maps pairs are similar, a MDS projection of the networks was done using correlation as a similarity measure. This correlation was measured between vectors, where the vector for each network contained all the distances between the saliency maps unique pairs, in the same order. This projection is shown in the left panel of Figure 3.33, where it can be seen that the points corresponding to the networks are approximately uniformly distributed. This means their similarity does not change much between pairs of networks, so no groups are formed. The average correlation between networks is 0.315, which is not negligible. This suggests that the network behaviour, in terms of the attributes captured by the saliency map diagnostic, shares some similarities.

In the right panel of Figure 3.32, it can be seen that the KDE curves for the distribution of pairwise distances of saliency maps produced for the same image by different networks appear to be unimodal, but varying in peak position and variance. To investigate if there were groups of images which shared similar patterns in the space of saliency maps

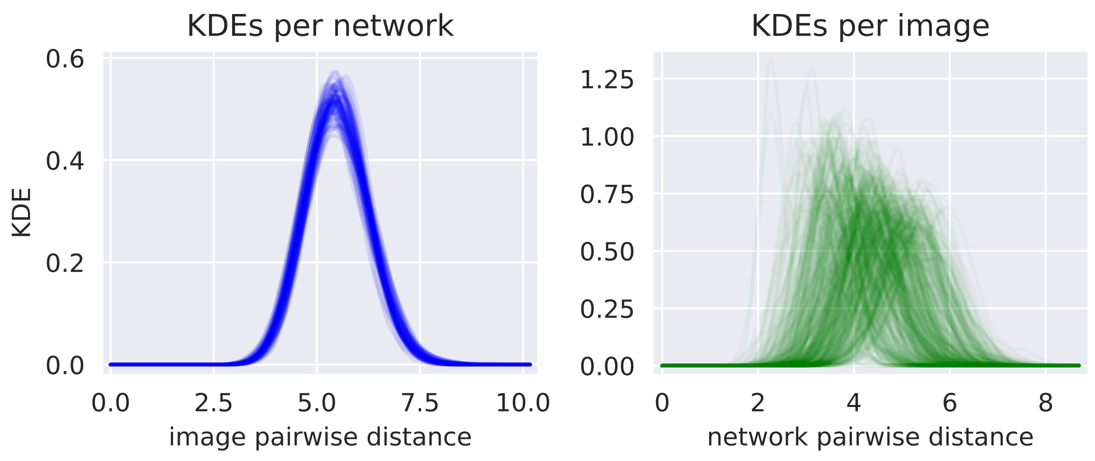


Figure 3.32: Left: Kernel density estimation plots for the distributions of the pairwise distances between saliency maps produced by each network (per network) for the 500 images in the sample. Each line represents the distribution of one network. Right: Kernel density estimation plots for the distribution of pairwise distances between saliency maps produced by the 80 different network instances for each image (per image). Each line corresponds to a distribution for one of the sample images.

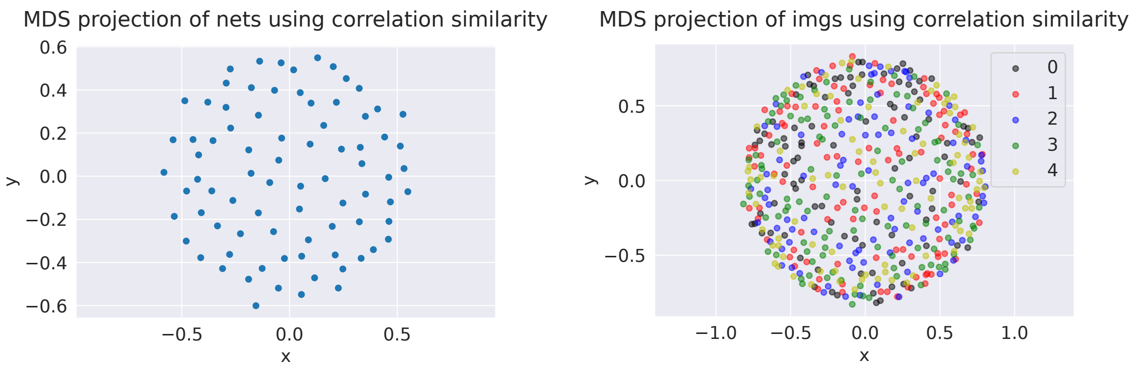


Figure 3.33: Left: MDS projection of the networks using as similarity measure the correlation between the vectors that hold all the pairwise saliency map distances produced by each network. Right: MDS projection of the images using as similarity the correlation between vectors whose elements are the pairwise distances of saliency maps from all the unique pairs of networks. Each point represents an image and they are colored according to their class.

produced by different networks, a MDS projection of the images was done using as a similarity metric the correlation of vectors, each containing all the pairwise distances of saliency maps produced by each of the unique pairs of networks for that example. This projection is presented in the right panel of Figure 3.33, where each point is associated to one image and they are colored according to its class label. Clear structures in this projection are not observable, though there is some greater density of points in the outside parts of the pattern. This indicates that there are no groups of significant number of images that are regarded as similar by the different networks in terms of their saliency attributes. Importantly, this is a departure from what was seen in the CNN models trained on the cartoon dataset, where some clusters were always visible, at least for the class with most images, which was clearly separated from the remaining. Here, there is a mixing of the points from all the classes. This suggests that the attributes the different networks capture as relevant in the saliency diagnostic are not good distinguishing marks to separate images between classes.

3.6.2 Occlusion sensitivity for a random sample of 500 images for the CIFAR05 dataset

The occlusion to sensitivity diagnostic results depend on the occlusion window size and the stride. If the window chosen is smaller, finer differences between occlusion maps can be captured. On the other hand, if the window is too small, occluding the region it covers will not make a difference for the prediction probability. In the following, we present results for an occlusion window of 2×2 cells. Since the CIFAR10 images are 32×32 pixels, the ratio between the area of the occlusion window to image size is the same as for the Cartoon dataset.

The left panel of Figure 3.34 shows the KDE curves for the distributions of pairwise distances between occlusion maps for the same network. Each line in the plot represents a different network. It can be seen that the distributions peak at, or very close to, a distance of zero between occlusion maps. This suggests a significant fraction of the pairs of occlusion maps are nearly identical, which can be an artifact of the small window size. It could happen that for many images, occluding a small region with a grey square does not affect the prediction of the network.

The right panel of Figure 3.34 shows a MDS projection of the networks. This plot was done using as a similarity measure the correlation between vectors that represent each network by the distances between the occlusion maps of the unique pairs of images obtained with it. The projection shows a mostly homogeneous dispersion of the network points, without groups. This, together with the fact that the average correlation between the networks is approximately 0.5, could lead one to believe that they are similar in terms of the attributes the sensitivity to occlusion diagnostic captures for the different images. However, given that all networks show a significant fraction of nearly identical occlusion maps, this could be another manifestation of the occlusion region being too small to affect

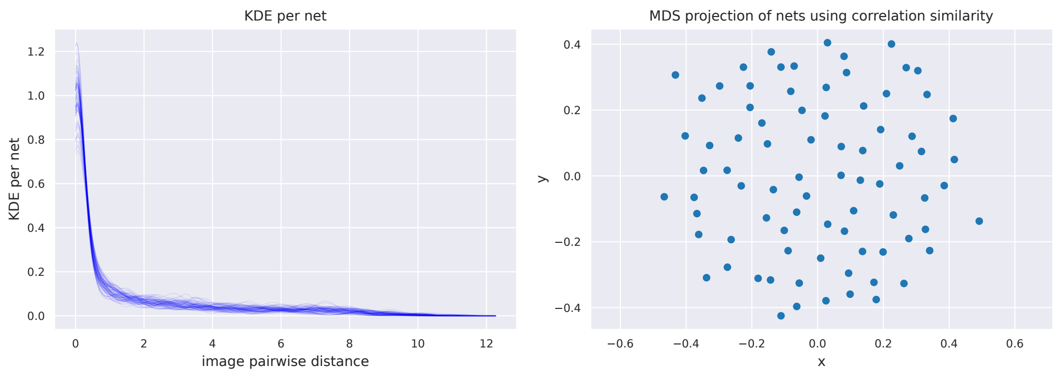


Figure 3.34: Left: Kernel density estimation plots for the distributions of the pairwise distances between sensitivity to occlusion maps produced by each network (per network) for the 500 images in the sample. Each line represents the distribution of one network. Right: MDS projection of the networks according to a correlation similarity where each network corresponds to a vector whose elements are all the pairwise distances between the occlusion maps of different examples.

network predictions.

In the analysis of the distribution of distances between occlusion maps obtained with different networks for the same image numerical problems associated with the small occlusion region size were encountered. In particular, it was found that for one of the images in the test sample, the probability attributed to the predicted class by all of the 80 networks was equal to one. Several different window sizes were employed to see when the occlusion map would show some difference. This was found to occur only when the window size was 16×16, resulting in a very low resolution. For this reason, the variability analysis per image between different networks was not possible.

However, in future work, not just different window sizes but also different window strides could be used to see if one can obtain occlusion maps that are not too coarse and at the same time able to capture more information regarding the sensitivity of the networks to the occlusion of different regions.

Chapter 4

Conclusions and future work

Given the increasing usage of neural networks in the development of applications with significant repercussions in our lives, such as automated driving and medical diagnosis, the fact that they are still largely black-boxes remains an issue. This has led to the growth of research in the domain of neural networks interpretability. In the first part of this work, an overview how neural networks work, and in particular convolutional neural networks, was provided. This was followed by a revision of recent interpretability techniques.

These interpretability methods are, in general, applied to a single instance (a network) of a model (task and neural network architecture). Moreover, many of these methods only provide an explanation for a single example from the dataset. On the other hand, the simple random initialization of the weights of neural networks at the beginning of their training will lead to differences between them. These two facts mean that, not only are the explanations associated to a single example, and therefore limited in what they can inform us about the network behaviour, but they can also differ for the same exact example, from one instance of the model to another. The variability of the explanations in these two perspectives, for the same network between different examples, and for the same example for different instances of the same model, raises the issue of reliability and consistency of these interpretability diagnostics.

In this work, we developed tools and methodologies that allow us to explore this variability for different interpretability diagnostics and assess their consistency. We did this in the context of convolutional neural networks models and applied them to two well known interpretability diagnostics, but the approach is easily transferable to other diagnostics and to other CNN models and datasets. It comprises three steps. The first step is the generation of a multitude of network instances of the model under study. In the second step, the interpretability diagnostics are applied to the instances and a subset of the test dataset, and distances between explanations are collected, in this case heatmaps, which serve as a measure of the variation between them. In the final step, the distributions of distances between explanations of different examples, for the same network; and of distances between the explanations of the same example for different networks, are analysed. This methodology and associated tools were described in chapter 3 as well as the results obtained for different models and datasets. For the Cartoon dataset, which was used in most of the work, twelve different models were employed, from the combination of three different architectures and four different tasks. It was found that, for some of the models (architecture + task), the variability between saliency maps for the same image but from different networks was higher than the variability between saliency maps of different images from the same network, which is the opposite of what one would expect. This occurred for the test sample with 1000 randomly selected images in the cases of glasses classification with the 3, 4 and 5 convolution layer architectures; intellectuals classification with the 3 and 4 convolution layer architectures; facial hair classification with the 4 and 5 convolution layer architectures; and face shape classification with the 5 convolution layer architecture. This lack of consistency for explanations of the same example suggests that some of the attributes the diagnostic is capturing may not be very meaningful in terms of the network predictions. The same analysis was performed with the sensitivity to occlusion diagnostic and it was found that it does not present the same problem for the models at study.

Preliminary analysis of the variability of both saliency and occlusion maps diagnostics was also performed for a VGG11 network trained on a subset of the CIFAR10 dataset with only 5 of its categories. For the saliency diagnostic, it was found that the attributes captured in the saliency maps appear not to be conducive to a distinction between images of the same class consistent between the different networks.

This work opens several avenues for further study, which were not possible to explore in the time devoted to this thesis. It would be interesting, for instance, to apply this tool on other interpretability diagnostics and models. Further analysis of the effect of window size and stride on the variability of the maps of sensitivity to occlusion is needed. On the cases where the variability analysis points to a lack of reliability of the diagnostic, it would be interesting to test whether regularization techniques, such as decreasing the number of filters in the convolution layers, could mitigate the problem.

Bibliography

[1] J. De Fauw, J. R. Ledsam, B. Romera-Paredes, S. Nikolov, N. Tomasev, S. Blackwell, H. Askham, X. Glorot, B. O'Donoghue, D. Visentin, et al. "Clinically applicable deep learning for diagnosis and referral in retinal disease." In: *Nature medicine* 24.9 (2018), p. 1342.

[2] J. Sirignano, A. Sadhwani, and K. Giesecke. "Deep Learning for Mortgage Risk." In: *SSRN Electronic Journal* (2018). ISSN: 1556-5068. DOI: 10.2139/ssrn.2799443. URL: http://dx.doi.org/10.2139/ssrn.2799443.

[3] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving." In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2722–2730. DOI: 10.1109/ICCV.2015.312.

[4] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, and J. Zhao. *End to end learning for self-driving cars.* 2016.

[5] Equivalent. *Correctional Offender Management Profiling for Alternative Sanctions (COMPAS).* Software. URL: https://www.equivant.com/northpointe-risk-need-assessments/.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems.* 2012, pp. 1097–1105.

[7] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." In: *IEEE Signal processing magazine* 29.6 (2012), pp. 82–97.

[8] S. Jean, K. Cho, R. Memisevic, and Y. Bengio. "On Using Very Large Target Vocabulary for Neural Machine Translation." In: *CoRR* abs/1412.2007 (2014). arXiv: 1412.2007. URL: http://arxiv.org/abs/1412.2007.

[9] G. Montavon, W. Samek, and K.-R. M uller. "Methods for interpreting and understanding deep neural networks." In: *Digital Signal Processing* 73 (2018), pp. 1 –15. DOI: 10.1016/j.dsp.2017.10.011.

[10] R. Courtland. “Bias detectives: the researchers striving to make algorithms fair.” In: *Nature* 558 (2018), pp. 357–360. DOI: 10.1038/d41586-018-05469-3.

[11] G. Ras, M. v. Gerven, and P. Haselager. “Explanation Methods in Deep Learning: Users, Values, Concerns and Challenges.” In: (2018). eprint: 1803.07517.

[12] J. Angwin, J. Larson, S. Mattu, and L. Kirchner. “Machine Bias.” In: *ProPublica* (2016).

[13] J. Dastin. “Amazon scraps secret AI recruiting tool that showed bias against women.” In: *Reuters* (2018). URL: https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G.

[14] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan. “A survey on bias and fairness in machine learning.” In: *arXiv preprint arXiv:1908.09635* (2019).

[15] *General Data Protection Regulation (GDPR)*. 2018. URL: https://gdpr.eu/tag/gdpr/.

[16] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. “Intriguing properties of neural networks.” In: *arXiv preprint arXiv:1312.6199* (2013).

[17] S. S. Haykin. *Neural networks and learning machines*. Third. Upper Saddle River, NJ: Pearson Education, 2009.

[18] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning.” In: *Nature* 521 (2015), 436–444.

[19] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. “Backpropagation Applied to Handwritten Zip Code Recognition.” In: *Neural Computation* 1 (1989), pp. 541–551.

[20] M. Elgendy. *Deep Learning for Vision Systems*. Manning Publications, 2020. ISBN: 9781617296192.

[21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. “Generative adversarial nets.” In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[23] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. “Photo-realistic single image super-resolution using a generative adversarial network.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4681–4690.

[24] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. “Image-to-image translation with conditional adversarial networks.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

[25] J. Langr and V. Bok. *GANs in Action: Deep learning with Generative Adversarial Networks*. Manning Publications, 2019. ISBN: 9781617295560. URL: https://books.google.pt/books?id=HojvugEACAAJ.

[26] H. Uzunova, J. Ehrhardt, T. Kepp, and H. Handels. "Interpretable explanations of black box classifiers applied on medical images by meaningful perturbations using variational autoencoders." In: *Medical Imaging 2019: Image Processing*. Ed. by E. D. Angelini and B. A. Landman. Vol. 10949. International Society for Optics and Photonics. SPIE, 2019, pp. 264 –271. DOI: 10.1117/12.2511964.

[27] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks." In: *Advances in neural information processing systems*. 2016, pp. 3387–3395.

[28] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. "Visualizing higher-layer features of a deep network." In: *University of Montreal* 1341.3 (2009), p. 1.

[29] K. Simonyan, A. Vedaldi, and A. Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps." In: *arXiv preprint arXiv:1312.6034* (2013).

[30] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. "Understanding neural networks through deep visualization." In: *arXiv preprint arXiv:1506.06579* (2015).

[31] M. T. Ribeiro, S. Singh, and C. Guestrin. "Why Should I Trust You?: Explaining the Predictions of Any Classifier." In: (2016), pp. 1135–1144. DOI: 10.1145/2939672.2939778.

[32] X. Ren and J. Malik. "Learning a classification model for segmentation." In: *null*. IEEE. 2003, p. 10.

[33] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, et al. "Least angle regression." In: *The Annals of statistics* 32.2 (2004), pp. 407–499.

[34] Y. LeCun, C. Cortes, and C. J. Burges. *The MNIST database of handwritten digits*. 1998. URL: http://yann.lecun.com/exdb/mnist/.

[35] A. Thampi. *Interperetable AI*. Manning, 2022. ISBN: 9781617297649.

[36] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, and R. Sayres. "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)." In: *arXiv preprint arXiv:1711.11279* (2017).

[37] P. Stock and M. Cisse. "ConvNets and ImageNet Beyond Accuracy: Understanding Mistakes and Uncovering Biases." In: *The European Conference on Computer Vision (ECCV)*. 2018.

[38] A. Mordvintsev, C. Olah, and M. Tyka. *Inceptionism: Going Deeper into Neural Networks*. 2015. URL: https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

[39] G. Alain and Y. Bengio. "Understanding intermediate layers using linear classifier probes." In: *arXiv preprint arXiv:1610.01644* (2016).

[40] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[41] A. Shrikumar, P. Greenside, and A. Kundaje. "Learning important features through propagating activation differences." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 3145–3153.

[42] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. "Network dissection: Quantifying interpretability of deep visual representations." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 6541–6549.

[43] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. "Scene parsing through ade20k dataset." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 633–641.

[44] S. Bell, K. Bala, and N. Snavely. "Intrinsic Images in the Wild." In: *ACM Trans. Graph.* 33.4 (July 2014). ISSN: 0730-0301. DOI: 10.1145/2601097.2601206. URL: https://doi.org/10.1145/2601097.2601206.

[45] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille. "The role of context for object detection and semantic segmentation in the wild." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 891–898.

[46] X. Chen, R. Mottaghi, X. Liu, S. Fidler, R. Urtasun, and A. Yuille. "Detect what you can: Detecting and representing objects using holistic models and body parts." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1971–1978.

[47] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. "Describing Textures in the Wild." In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 3606–3613.

[48] P. Agrawal, R. Girshick, and J. Malik. "Analyzing the Performance of Multilayer Neural Networks for Object Recognition." In: *Computer Vision – ECCV 2014*. Ed. by D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars. Cham: Springer International Publishing, 2014, pp. 329–344.

[49] J. Wang, Z. Zhang, C. Xie, V. Premachandran, and A. Yuille. "Unsupervised learning of object semantic parts from internal states of CNNs by population encoding." In: *arXiv e-prints*, arXiv:1511.06855 (Nov. 2015), arXiv:1511.06855. arXiv: 1511.06855.

[50] R. Fong and A. Vedaldi. "Net2vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8730–8738.

[51] M. D. Zeiler and R. Fergus. "Visualizing and understanding convolutional networks." In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[52] M. D. Zeiler, G. W. Taylor, and R. Fergus. "Adaptive deconvolutional networks for mid and high level feature learning." In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 2018–2025.

[53] 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*. Ieee. 2009, pp. 248–255.

[54] F. M. Graetz. *How to visualize convolutional features in 40 lines of code*. Ed. by T. D. Science. 2019. URL: https://towardsdatascience.com/how-to-visualize-convolutional-features-in-40-lines-of-code-70b7d87b0030.

[55] P. E. Rauber, S. G. Fadel, A. X. Falcao, and A. C. Telea. "Visualizing the hidden activity of artificial neural networks." In: *IEEE transactions on visualization and computer graphics* 23.1 (2016), pp. 101–110.

[56] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng. "Reading digits in natural images with unsupervised feature learning." In: *Proc. Neural Information Processing Systems*. Vol. 2011. 2011, p. 5.

[57] A. Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009. URL: https://www.cs.toronto.edu/~kriz/cifar.html.

[58] T. Cox and M. Cox. *Multidimensional Scaling, Second Edition*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. CRC Press, 2000. ISBN: 9781420036121. URL: https://books.google.pt/books?id=SKZzmEZqvqkC.

[59] P. Geurts, D. Ernst, and L. Wehenkel. "Extremely randomized trees." In: *Machine learning* 63.1 (2006), pp. 3–42.

[60] Y. Li, J. Yosinski, J. Clune, H. Lipson, and J. E. Hopcroft. "Convergent learning: Do different neural networks learn the same representations?" In: *FE@ NIPS*. 2015, pp. 196–212.

[61] M. Raghu, J. Gilmer, J. Yosinski, and J. Sohl-Dickstein. "Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability." In: *Advances in Neural Information Processing Systems*. 2017, pp. 6076–6085.

[62] A. Morcos, M. Raghu, and S. Bengio. "Insights on representational similarity in neural networks with canonical correlation." In: *Advances in Neural Information Processing Systems*. 2018, pp. 5727–5736.

[63] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *International Conference on Learning Representations*. 2015.

[64] U. Rizwan. *Saliency Maps in Tensorflow 2.0*. 2020. URL: `https://usmanr149.github.io/urmlblog/cnn/2020/05/01/Salincy-Maps.html`.

[65] *Cartoon dataset*. URL: `https://google.github.io/cartoonset/index.html`.

[66] B. Doran. *UniDip Python Port*. URL: `https://github.com/BenjaminDoran/unidip`.

[67] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, 226–231.