LORIS GIULIVI – 920992 A.Y. 2019/2020

POLITECNICO
MILANO 1863

PERCEPTION VISUALIZATIONS FOR SIAMESE NETWORKS

SEEING THROUGH THE EYES OF A CONVOLUTIONAL NEURAL NETWORK

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

RELATORE: GIACOMO BORACCHI

- To unrestrained curiosity

Abstract

Today's artificial intelligence models power the world we live in; deep neural networks transform the way we solve problems by harnessing the vast amount of data available to people of the information age, and by somewhat mimicking the brain's functioning. Deep learning models are able to solve tasks in an ever-expanding landscape of scenarios, but our eagerness to widely apply these powerful models makes us only focus on the correct functioning of the systems and leaves behind the capability of understanding them.

In certain circumstances, understanding how the system derived his conclusions is not vital, and a functioning model might be enough. However, more and more often, this kind of models are being used in fields where decisions are crucial, and errors even more so. In this case, understanding why a model took the specific path that led to its decision might be essential in developing a model that is safe and trusted. A worrying consequence of the inability of explaining the decisions of these systems takes form in adversarial attacks: a set of techniques that can trick deep neural networks by focusing on their sensitivity to targeted noise. These attacks can completely change the output of a network by modifying the input ever so slightly that even the most attentive of humans could not notice. In this work, we develop a set of techniques to better understand the functioning and "reasoning" of a specific set of models, namely convolutional neural networks (CNN) and Siamese networks. We propose a modification on the general architecture of these models to increase their interpretability, and a novel explainability technique based on a saliency map generated through the study of adversarial attacks.

Abstract (Italiano)

I modelli di intelligenza artificiale odierni sono alla base del mondo tecnologico in cui viviamo; le reti neurali *deep* hanno trasformato le modalità in cui gli ingegneri risolvono i problemi postigli permettendo loro di sfruttare tutto il potenziale proveniente dalla grande quantità di dati disponibile, e in qualche modo anche imitando il funzionamento del cervello. Le reti neurali sono in grado di risolvere problemi in uno scenario di compiti sempre più grande; questo però ci rende impazienti, avari di risultati, e costringe a lasciare indietro alcune aree dello studio di questi modelli, in particolare quelle che mirano a capire le decisioni di questi ultimi.

In particolari circostanze, comprendere perché il sistema ha ottenuto un determinato risultato non è di vitale importanza; in tal caso un modello funzionante potrebbe essere abbastanza. In molti altri casi, però, sistemi di questo tipo vengono applicati in campi dove le decisioni prese sono cruciali, e a maggior ragione lo sono gli errori. In questi casi, per sviluppare modelli sicuri e fidati, è importante comprendere il perché delle decisioni prese. Una grave conseguenza dovuta alla difficoltà di comprendere queste reti prende forma nei cosiddetti *attacchi avversari*: una serie di techniche che permettono di confondere una rete neurale colpendo le debolezze che questi modelli presentano nei confronti di specifici disturbi. Questi attacchi permettono di cambiare drasticamente i risultati di questi modelli agendo minimamente sui loro *input*; cosí poco, infatti, che neanche il più attento degli utenti potrebbe accorgersi della differenza. In questa tesi, il nostro obiettivo sarà quello di sviluppare una serie di tecniche per meglio comprendere il funzionamento e il "pensiero" di un specifico insieme di modelli *deep*, identificato dalle reti neurali convoluzionali (CNN) e dalle reti siamesi. I nostri contributi consistono principalmente nello sviluppo di una nuova architettura per questi modelli in grado di migliorare la loro comprendibilità, e in nuove techniche di *explainability* basate sugli attacchi avversari che permettono di meglio comprendere le decisioni prese da questa tipologia ti reti neurali.

Table of Contents

Figures

Algorithms

Tables

1 Introduction

1.1 Research focus

The objective of this work is to develop new explainability techniques for deep neural networks. If these models are to be applied to critical tasks, there is a need for debugging and validation of the system they compose. The starting point is the current research in the field of explainable artificial intelligence (XAI). XAI is a sub-field of artificial intelligence that promotes a set of tools, techniques, and algorithms that can generate high-quality, interpretable, intuitive, human-understandable explanations of AI decisions [1].

By explanations, we refer to their definition as seen in [1] and clearly distinguish between *interpretability* and *explainability*. The former is the ability of a model to provide an output that is clear to a human; in other words, the output must be presented in a way that intrinsically contains an explanation of the decisions made. A system is instead explainable if the result is accompanied by a separate, self-sustained explanation that is given on top of the output of the network. Our objective, if we want to achieve explainability, will thus be that of building techniques to provide additional information together with the model's result to aid a human user in understanding it.

We will focus on the class of models that perform binary ranking. This class of models has not been previously tackled from an explainability point of view, at least explicitly, but we think that there is a need for such an analysis; as we will show in this work, these models can show intriguing properties. We will validate our findings with experiments on example case studies, which, albeit non-critical, are very informative regarding our result. The simplicity of the selected case studies renders them transparent for analysis, hence they might be more instructional than possible production-class tasks; we will however hint at how our methodologies can be applied to real world scenarios.

1.2 Advancements and methodologies

In this work, we take advantage of adversarial attacks and build explanation techniques using information coming from the attacks. In standard circumstances, the implications of adversarial attacks are clear: imagine, for example, to add a sticker to a street sign and trick a self-driving car into going well over the speed limit. This was successfully done in [2] and [3], by means of physical attacks in the real world. However, there exists a duality between adversarial attacks and explanations [20]; exploiting this can allow us of making good use of these attacks.

For models that processes images, such as those analysed in this thesis, an adversarial attack takes form in a slight perturbation of the input image that is able to trick the network into "seeing" something different. A classic example of this can be seen in *Figure 1*, where an image that was previously classified as "panda" is slightly perturbed (note the $.007$ multiplier applied to the adversarial noise), resulting in an image which to the naked eye is undistinguishable from the first, but that is able to deceive the model into thinking it is an image representing class "gibbon".

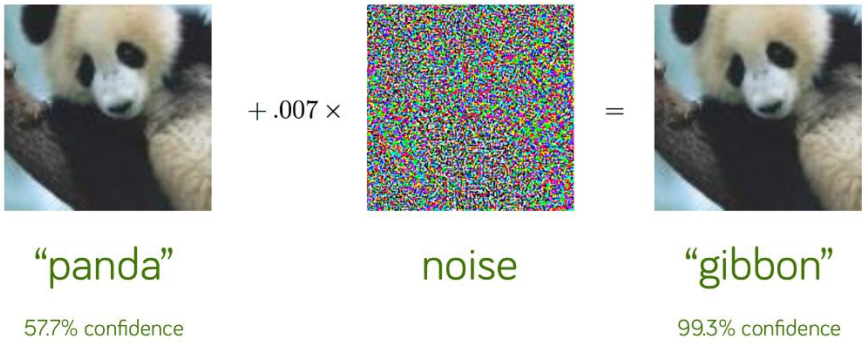


Figure 1 - Example of adversarial attack [1]

The adversarial noise seen in *Figure 1* is not random; it is *targeted* to push the result of the model towards another result, in this case the class “gibbon”. Both in the picture and in our thesis, noise is generated by following the model’s output gradient to find perturbations in the input space that result in the desired target outcome. In this setup, the structure of the generated noise can give information on what parameters of the input the network is most sensitive to. This means that we can obtain knowledge regarding what feature change the model is most vulnerable to, and that these vulnerabilities coincide with the features that most influenced the model’s result. In this thesis, we will show how we can harness these concepts in order to generate *saliency maps*: explanation images that show where in the image the model posed the most attention in deriving its output.

We will develop a series of novel techniques to build saliency maps for the models that are object of our research, namely convolutional Siamese networks for binary comparison. These techniques will help users of such models in understanding the decisions taken by highlighting the regions of the image that most impacted the resulting outcome. Saliency maps will be built with the objective of making them complete but not overwhelming for the user, which means that they must be able to display the totality of the extracted information regarding the model’s decision in a compact way. In this work, we will test our findings with experiments on the MNIST and UTKFace dataset, respectively composed of images of handwritten digits and of human faces. The objective for models trained on the MNIST dataset is to identify which digit is greater (in the arithmetical sense); these first trials will be used to narrow the research direction and refine our techniques. With the knowledge gained from these experiments we then apply our methodologies to the much richer problem of comparing people’s faces based on their age. Our models were very performant both in giving precise prediction of the age relationship between input samples and in providing relevant, understandable explanations that could be used to comprehend which features in the image were most relevant for the model’s decision. This information can also be used to gain a deeper understanding of what the model learned regarding age from the samples it was shown during training. In *Figure 2*, we

[1] Image source: I. J. Goodfellow, J. Shlens, C. Szegedy, Explaining and Harnessing Adversarial Examples, arXiv:1412.6572

show an example of our results. In the figure, we show how our model, trained to compare people's faces by age, was able to give a prediction score indicating that the first sample (left) was younger than the second one (right). In the figure, we also show the targeted adversarial perturbation that was generated through our methodologies, which, when applied to the original image (left), deceives the model into thinking that the sample represents someone older. The brighter pixels in the perturbation represent the regions in the image the model is most sensitive to (for the purposes of giving the comparison result). At the bottom of the figure, we show the outcome of one of our saliency map techniques coming from the left sample of this specific input pair. This saliency map merges the input and the adversarial perturbation transforming it into a more understandable form; the result is an image that highlights the regions of the input that were most significant for the model's comparison prediction.

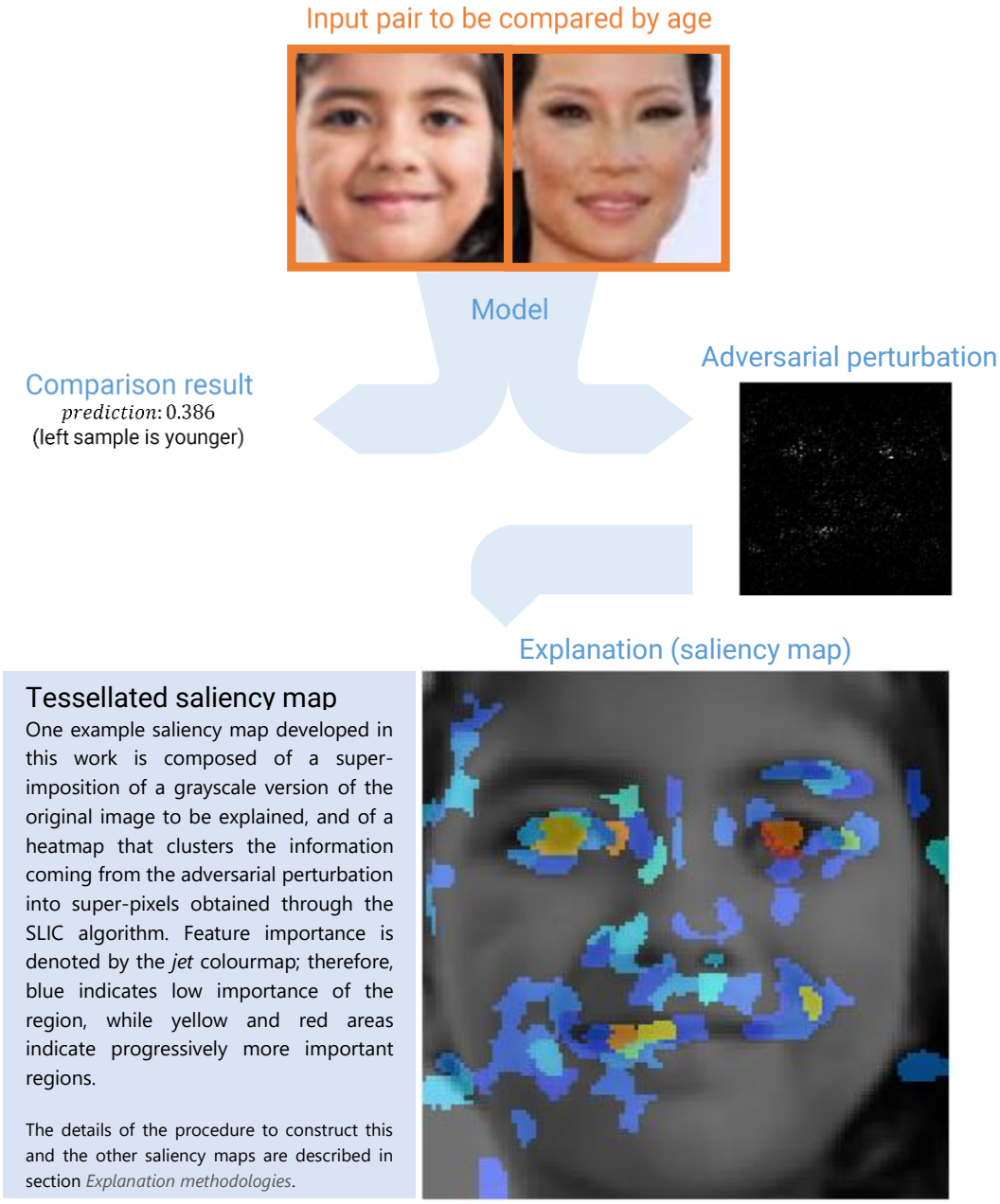


Figure 2 - Example saliency map developed in this work

The other main focus of this work lies however in the perception of the models in analysis. We will develop architectural modifications to convolutional Siamese networks that take form in an additional generative branch able to project the network's latent space into image space. We use this new concept to visualize the network's perception of the input. These visualizations help in the validation of our other techniques, but can also be used to generate a new class of explanation images.

The result of these new architectural improvements is the concept of *perception visualization* of the model, which will give us insight on how the network perceives the input and how the adversarial attacks modify this perception. This is done by creating an *ad-hoc* model that outputs both the result and a relative explanation. Study of the perception visualizations can shed light on how the perturbations that are generated through our techniques and highlighted by our saliency maps affected the network inner representation of the input. This information can be used to obtain a more semantical understanding of the feature importance as perceived by the model, but also to validate that the perturbations applied through gradient descent affected the model coherently with the specified target outcome.

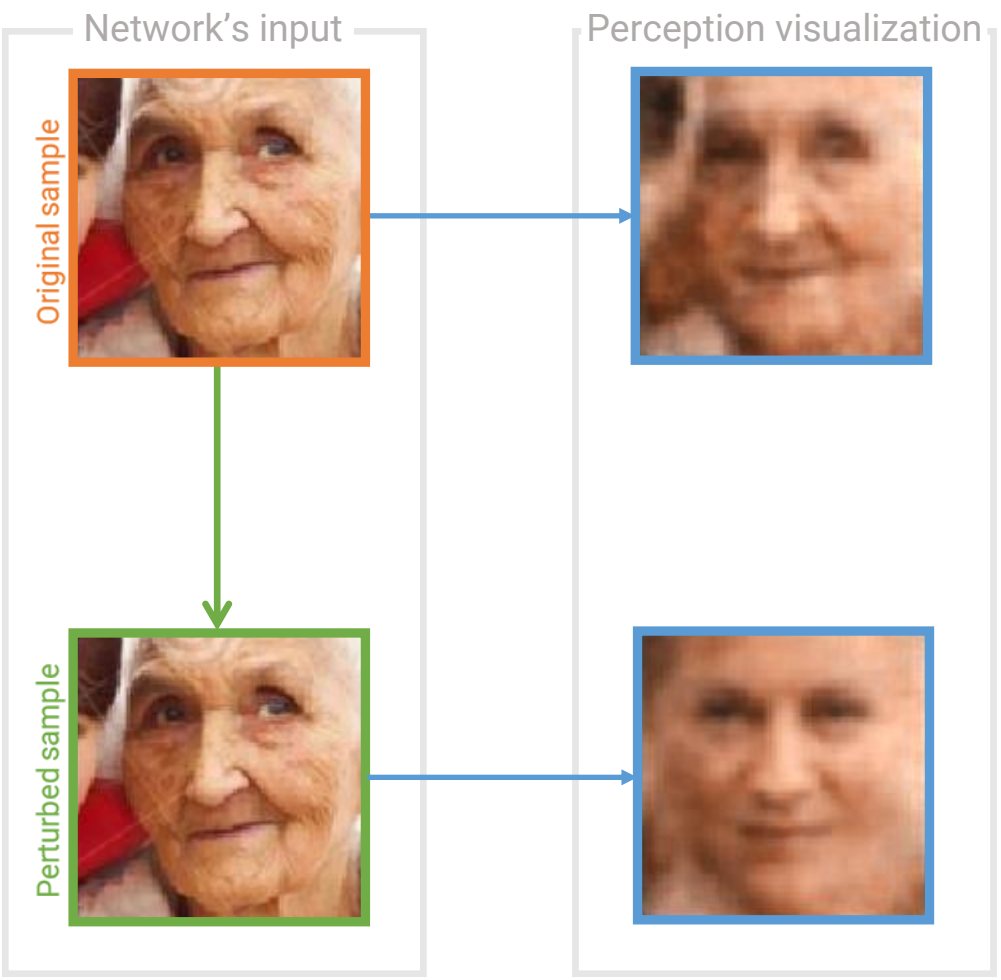


Figure 3 - Example of perception visualization

In *Figure 3* we show an example of how perception visualization can be used to inspect and validate whether the consequences of the adversarial attack on the original sample

actually reflect the desired effects on the network's perception of the input. In the figure, the perception visualization reveals how the original sample is correctly reconstructed by the network; indeed, the two images in the top row display strong similarity. With the adversarial attack, in the example, we aimed at deceiving the network into thinking that the input sample represents someone much younger than the original. The result is a perturbed input sample that, as for *Figure 1*, from a human stand point is indistinguishable from the first, but that when fed through the model takes a completely different form. Because the resulting perception visualization resembles a younger version of the original one, we can say that the perturbation in the space of the perception visualization is coherent with our attack, and that the regions that we have modified in the input space were relevant for the model. In sections *Proposed solution* and *Experiments and discussion* we will detail the theoretical and implementation details of these techniques.

1.2 Thesis structure

In the first section of this thesis, we will describe the scientific foundations this work builds upon. The aim is to present the previous pieces of work that enabled our research with a particular focus on describing how the specific literature helped us in achieving our goals. For this end, for every subsection in *Theoretical background and state of the art*, we will give a brief description of how the analysed study was used and which were the most relevant parts that relate to our work.

In *Problem formulation*, we will formally describe the problem we want to solve and provide examples in order to better clarify our goals. The aim is to provide a clear and concise description of the problem at hand, in order to ease the understanding of the following sections and to enable formal evaluation of our findings.

The *Proposed solution* section focuses on describing all the details of our work. In this section, everything spanning from the algorithms to the implementation details will be discussed, with a particular emphasis on the newly developed explainability techniques, which are the main object of this thesis.

In the *Experiments and discussion* section, we will describe and evaluate our experimental findings following our research. In this section, we will examine how the experiments were carried out and investigate which are the conclusions that can be made out from the results. The focus of this section is to provide these findings in a clear and understandable way; we will therefore accompany the experiments with various detailed figures.

The last section regards *Conclusions and future work*; in this section, we will briefly explore the possible future research direction stemming from our work. We will also provide a general description of our findings and summarize our main contributions to the explainability research field.

2 Theoretical background and state of the art

In this section, we will analyse the main results upon which this work was built and the technological advancements that made its implementation possible.

2.1 Convolutional neural networks

Convolutional neural networks, CNN for short, are a class of models that, owing to their structure, are very well suited to solve visual tasks, such as image classification. This is due to their inherent bias towards image composition, that gives these networks an innate prior knowledge on natural images.

CNNs have been an ever-expanding topic of research in the last decade; the increased power of GPUs and the efficient implementation of algorithms for training these models has let researchers achieve breakthrough results. For example, in [4], enhancements in the practical application of convolutional neural networks allowed to build an image classifier that significantly improved on the previous ones that mainly relied on hand-crafted features such as SIFT [5].

2.1.1 CNN Architecture

A convolutional neural network is composed of a series of convolutional layers that act as filters, and pooling layers that reduce the dimensionality of the problem whilst keeping some image structure. In this section, we will dive in depth into the theoretical foundation of CNNs and see how they are structured.

In the context of deep neural networks, the convolution operation is a discrete filtering operation conducted on an input image $A \in \mathcal{M}^{a,b,z}$ with a kernel matrix $K \in \mathcal{M}^{x,y,z}$. The result is a new matrix $B \in \mathcal{M}^{a,b,1}$ where the two first dimensions a, b are the same but the third dimension has size 1.

$$B_{i,j} = \sum_{h,k,l=(1\dots x;1\dots y;1\dots z)} K_{h,k,l} * A_{i+h-\left\lfloor\frac{x}{2}\right\rfloor\ ,\ j+k-\left\lfloor\frac{y}{2}\right\rfloor\ ,\ l} \qquad E\ 1$$

For values of i, j that are in the outermost $\frac{x}{2}$ and $\frac{y}{2}$ positions in A, we may have undefined behaviour of the operation. The solution to this edge problem can be solved in two ways, either by using *padding* or by returning a differently sized matrix. In the first case, we assume that each out of range value is equal to zero; in this way, the result will be a matrix $B \in \mathcal{M}^{a,b,1}$ of the same dimension as the input. In the second case, we do not provide a result for out of range value; the result will hence be a matrix $B \in \mathcal{M}^{a-x,b-y,1}$.

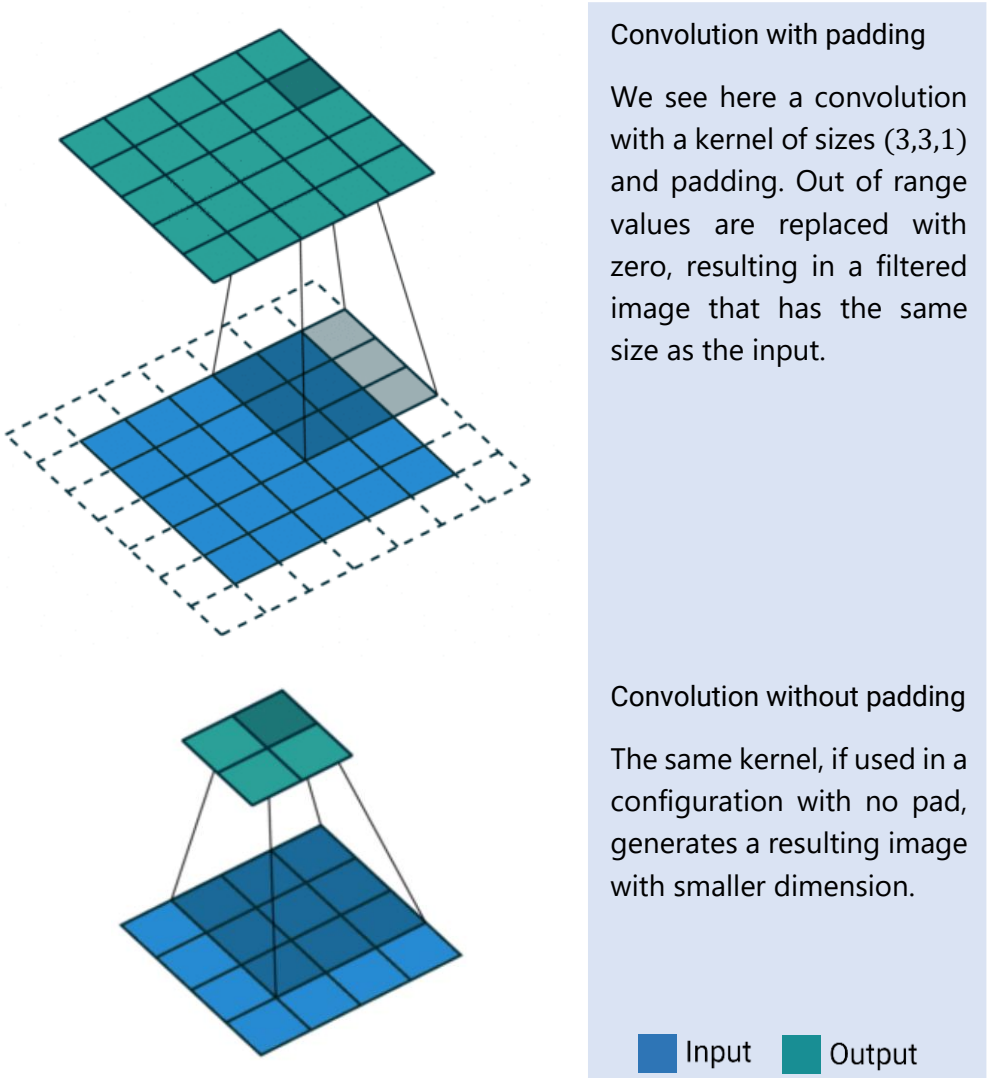


Figure 4 - Convolution with and without padding [2]

A generalization of the convolution operation is called strided convolution. With strided convolution, the filtering is only applied to a portion of the possible submatrices. If the stride amount is s, we will compute the filter operation only for submatrices centred in i, j with $mod(i, s) = mod(j, s) = 0$. The result of a convolution with stride s for a matrix $A \in \mathcal{M}^{a,b,z}$ is a matrix $B \in \mathcal{M}^{\frac{a}{s},\frac{b}{s},1}$ when padding is applied. We show that this is actually a generalization of the method explained above as, when $s = 1$, we find $mod(i, s) = mod(j, s) = 0$ for every possible i, j.

The controlling parameters of the operation are placed in the kernel of the convolution that consists in a matrix of weights. The dimensions x, y, z are the kernel size and they define the receptive field of the convolution, while the values of the kernel parameters define the convolution behaviour when presented with the input.

[2] Image credit: Convolution arithmetic https://github.com/vdumoulin/conv_arithmetic/blob/master/LICENSE

A convolutional layer is a function $\mathcal{M}^{a,b,c} \rightarrow \mathcal{M}^{a',b',c'}$ that performs the convolution operation c' times on the input, each time with a different kernel. Typically, all kernels have the same dimensions, but they always must have the same third dimension c to properly define the convolution on the whole input matrix.

Because the convolution operation is linear, in order to obtain a model that is able to solve complex tasks, we must induce nonlinearity through other means. Activation functions are one of the two sources of nonlinearity in a standard CNN architecture, and consist in applying a nonlinear function to each output of every convolution. For the standard, non-strided (equivalent to stride 1), padded convolution, we obtain:

$$B_{i,j} = g\left(\sum_{h,k,l=(1\ldots x;1\ldots y;1\ldots z)} K_{h,k,l} * A_{i+h-\left\lfloor\frac{x}{2}\right\rfloor\ ,\ j+k-\left\lfloor\frac{y}{2}\right\rfloor\ ,\ l}\right) \qquad E\,2$$

Where g is a nonlinear function, typically either a sigmoid or ReLU.

Convolutional layers, as we have seen, allow to filter the image and transform it to obtain relevant information; the kernels of the filters are in fact learnable parameters that can be optimized through gradient descent. The objective of a convolutional neural network is, however, to reduce the dimensionality of the input while keeping information regarding spatial locality. While information about locality is preserved by convolution, dimensionality reduction is still only achievable through strided convolution or using unpadded convolutions, which can be suboptimal solutions and lead to information loss in some situations.

To solve this issue, and to induce even more nonlinearity in the model, in a standard CNN architecture, convolutional layers are alternated with pooling layers. Pooling layers reduce the spatial size of the input $A \in \mathcal{M}^{a,b,1}$ by constructing an output $B \in \mathcal{M}^{\frac{a}{n},\frac{b}{n},1}$ composed of representative values for regions of the input. These layers compute the pooling operation for every channel in the input, obtaining an output which has the same third dimension.

A typical pooling layer is $maxpool$, that for every region of size $(n, n, 1)$ in the input A takes the maximum value and discards the rest, placing the selected value in the respective location in the output B.

$$B_{i,j} = \max(A[(i-1)*n+1{:}\,i*n\ ;\ (j-1)*n+1{:}\,j*n]) \qquad E\,3$$

From a wider point of view, the architecture of a CNN is constructed to contract the x, y dimensions of the input and "vectorize" them into the z dimension. Practically, this translates into an increasing number of convolution filters and a decreasing spatial size throughout the layers. Typically, the last layer will conclude this process and output a vector that can then be fed to fully connected layers or be used otherwise.

Vectorization

In a standard CNN architecture, we iteratively increment the number of filters and reduce the other dimensions to finally obtain a vector.

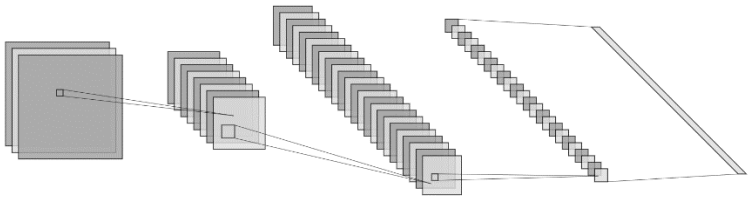


Figure 5 - Example of standard CNN architecture

In this thesis, a CNN is the base block over which the model is constructed. Because we want to give insight over how machines perceive visual tasks, these models were the obvious choice, due to their high performance in this kind of job.

2.2 Autoencoders

Another very important class of models regarding image analysis that have found use in this work is that of convolutional autoencoders. In this section, we will describe their theoretical foundation and how they relate to our efforts.

Autoencoders, in general, are models whose task is capturing the most salient features present in a dataset and, as the name suggests, encode them in a vector space with lower dimensionality. In other words, we might say that autoencoders reduce the dimensionality of the input optimizing for low information loss.

To do so, autoencoders are usually trained using a specular network, composed of an encoder and a decoder. The encoder takes as input the original data and embeds it in a feature vector of a specified size, lower than that of the input; this feature vector is then fed to the decoder that is trained to reconstruct the original input. The objective is shared between the two networks and consists in providing a reconstruction that is most similar to the input.

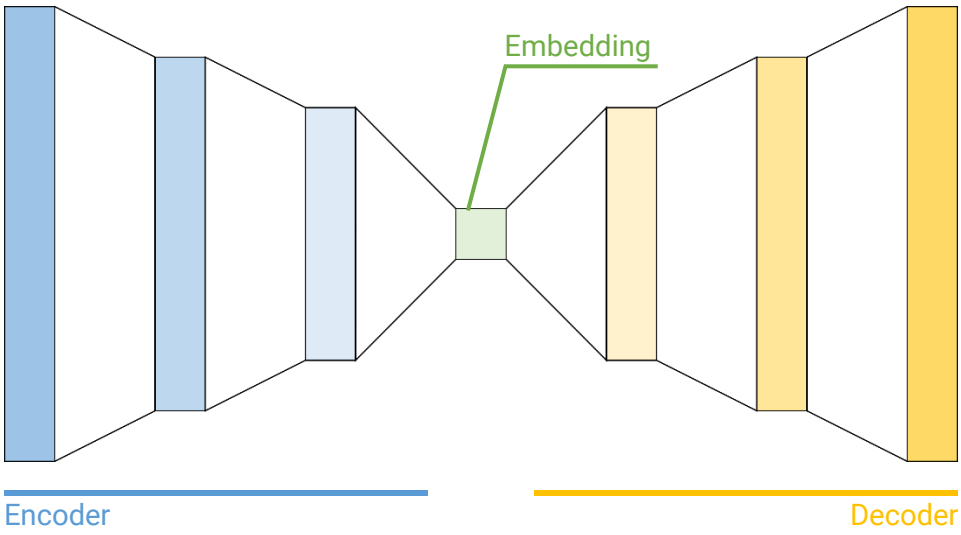


Figure 6 - Autoencoder architecture

Simultaneously training the encoder and the decoder results in an encoder able to discern between relevant and useless features and encode those that are more important into the resulting embedding, and a decoder able to understand the specific embedding and reconstruct the input.

Convolutional autoencoders do so with images; we can define inverse operations to the ones seen for CNNs, thus allowing for upsampling layers and deconvolutional layers that can transform a feature vector into an image. We can therefore build a standard CNN able to encode an image into a feature vector, and a convolutional decoder able to bring it back to an image.

For the decoder part of a convolutional autoencoder, we must be able to reverse the convolution operation. While upsampling is possible by using interpolation or other known techniques, these methodologies lack the complexity of the operation they are made to invert. The theoretical foundation that makes possible the inversion of the convolution operations resides in the concept of transposed convolution, which is used to up-sample the input feature map to a desired output feature map using some learnable parameters.

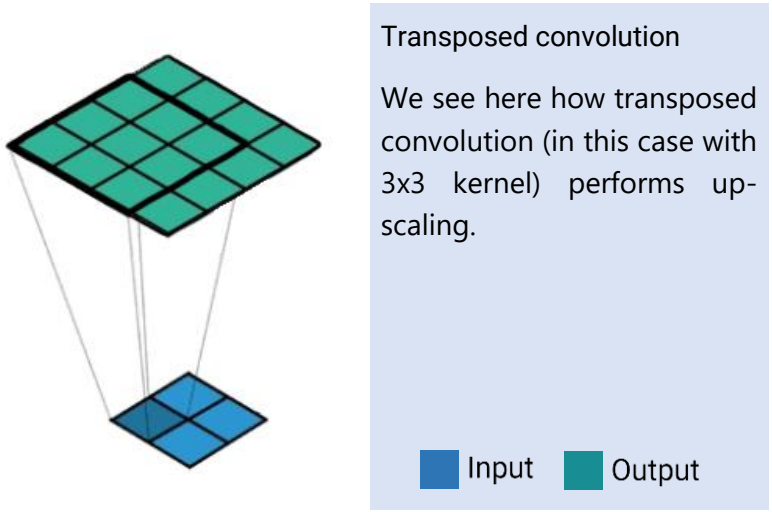


Figure 7 - Transposed convolution [1]

The transposed convolution operation from an input $A \in \mathcal{M}^{a,b,1}$ constructs its result $B \in \mathcal{M}^{a',y',1}$ by summing up matrices made from the element-wise product between each element of A and elements of a kernel $K \in \mathcal{M}^{x,y}$. Each product $A_{i,j} *_{element\ wise} K$ thus gives a matrix $R_{i,j} \in \mathcal{M}^{x,y}$ that can be padded to dimensions a', b' and centered at location i, j.

For each i, j in $(1..a', 1..b')$, we will therefore construct the matrix $R_{i,j}$:

$$j - \left\lfloor \frac{y}{2} \right\rfloor$$

$$i - \left\lfloor \frac{x}{2} \right\rfloor \begin{bmatrix} 0 & 0 & \dots & & & 0 \\ 0 & \dots & & & & \dots \\ \dots & & & & & \\ & & A_{i,j} * K_{1,1} & \dots & A_{i,j} * K_{1,y} & \\ & & \dots & \dots & \dots & \\ & & A_{i,j} * K_{x,1} & \dots & A_{i,j} * K_{x,y} & \\ & & & & & \\ 0 & \dots & & & & 0 \end{bmatrix} \quad E\,4$$

The transposed convolution is then finally computed as:

$$R = \sum_{i,j \in (1..a', 1..b')} R_{i,j} \quad E\,5$$

Similar concepts are used in this work to provide visual explanations coming from models that process images.

2.3 Learning to rank

Learning to rank means to learn an order relationship and, given a set of samples, list them following the correct sequence according to the required ordering. The output of a sample can therefore not exist for a single input but can only exist in relation to other samples in a set. More formally, given a set of samples $D = (d_1, d_2, \dots, d_n)$ and an order relationship "$\succcurlyeq$", an n-ary ranking model M_n is a function that is able to sort the samples so that they follow the ordering specified by "$\succcurlyeq$".

$$M_n : D \mapsto D' : D'_i \succcurlyeq D'_{i+1} \forall i = 1 \dots n-1 \quad E\,6$$

In the literature, learning to rank is a task that has a very defined presence in data mining, mainly in the context of information retrieval and recommender systems. In these systems, the objective is to extract from a dataset a subset of samples that are most relevant to a query. It will be clear that our method does not rely on queries; instead, we want to learn to give a ranking to any pair of samples in the dataset.

In [6] we can see the capabilities of CNNs and particularly of a Siamese CNN in learning n-ary rankings. In [6], the model was able to learn a similarity ranking with respect to a query image and recognize which image was of a similar class as that of the query image. This allowed for few or zero-shot learning of an image classifier, as class features do not need to be learned in this context but can be implicitly described by the query.

The results of this work show how learning a similarity metric in a data-driven way can achieve better results than using a hand-crafted metric. For our purposes, this demonstrates that CNNs can learn rankings and that these rankings are "natural". In other words, the network can learn metrics purely from the data to obtain an understanding of the order relationship over which the ranking takes place.

Learning to rank has been also applied in a more general framework than that of data mining. For example, in [7], a probabilistic cost function is developed to allow a network to learn a global ranking from a series of pairs of samples (n-dimensional feature vectors). In [7], the RankNet network was able to learn a ranking by using a natural extension of the standard cost functions used in neural networks.

In this thesis, we will make use of models able to solve rankings over images, but we will limit our scope only to binary ranking; this means that the cardinality of the input set is equal to 2. This allows to achieve the desired property of obtaining always relative results while keeping the problem simple and flexible for our explorative work.

2.4 Siamese networks

Another class of deep networks that was fundamental for the development of this work was that of Siamese networks. These are deep networks composed of two cloned sections that share parameters and of a tail section that joins the two copied branches. In the literature, these networks are mainly used to solve tasks that involve learning some metric and then determining whether two samples are similar or different with respect to the learned measure.

A Siamese network is trained to encode an input into an embedding such that from the latent space of the embeddings it is possible to distinguish between similar and dissimilar samples with respect to the metric that is required by the task; in other words, Siamese networks learn the embedding that exposes the highest variance to the required metric. An example of such similarity metric is that of similarity in the task of face recognition; a Siamese network could be trained to give a similarity score which is higher if the two input images are pictures of the same person, and lower otherwise. In the example, the network might discard, in its inner representation of the input, features that are irrelevant for recognizing a specific person, and only keep those that most define identity. In particular, the model might generate embeddings that are invariant of face pose, expression, and of light condition of the input, but that are also very sensitive to the relative position of nose, mouth, eyes and to other identifying features.

In *Figure 8*, we see a possible architecture for a Siamese network. The cloned branches of the network (in blue) are responsible of computing the embedding $f(\bullet)$ of the input pair; the reason for the branches to be duplicated is that the embeddings must be consistent between the two samples in order to perform the same encoding and be comparable. The tail section (in yellow) is instead responsible of interpreting the embeddings; in some networks, the tail is only made up of a component that computes the vector difference between the resulting embeddings; in other models, the tail is composed of one or more fully connected layers that allow for more complex learning over the embedding's differences. In the latter case, we need a connector function (in green) that takes the two embeddings generated by the Siamese branches and that generates a new vector to then be fed to the dense layers downstream. This function can be of various forms but is typically a simple concatenation layer that takes the two embeddings and joins them.

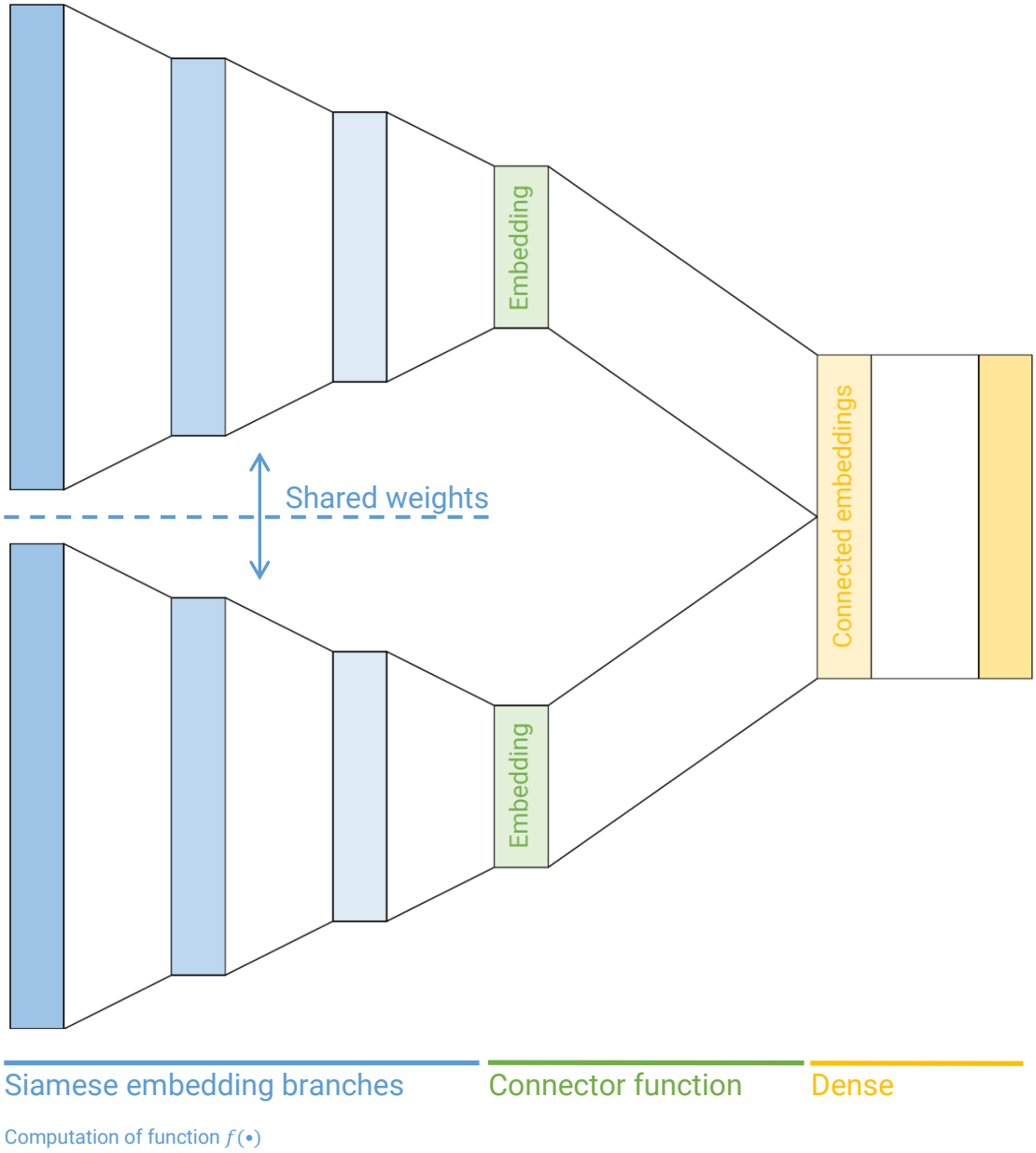


Figure 8 - Siamese network architecture

Siamese networks are quite flexible in their base form; this means that the input and therefore the Siamese branches (in blue in *Figure 8*) can be of any structure. To apply these models to visual tasks, this portion of the network can be composed by a CNN. As it is shown in [8], using a Siamese CNN can be used to solve the task of matching images that show the same landmark.

To train Siamese networks in their typical shape, pairs of input samples are shown to the model; the samples are then embedded, and the embeddings are compared. The optimization process then works to minimize either the contrastive loss [9] or the triplet loss [10].

With contrastive loss, the model is forced to pull together similar sample pairs and push apart different ones. This means that the target for ground-truth-similar pairs will

be 1 and 0 for the other pairs. Unlike conventional learning systems where the loss function is a sum over samples, the loss function here runs over pairs of samples.

Given a pair of input samples X_1, X_2 and a label Y assigned to this pair, we will have $Y = 1$ if the pair elements are deemed similar, $Y = 0$ otherwise. We now run the sample through our model to obtain embeddings $f(X_1), f(X_2)$ and compute the Euclidean distance between these results as $D = |f(X_1) - f(X_2)|_{L2}$. Contrastive loss will finally be:

$$\mathcal{L}_{contrastive} = \sum_{i=1..P} L\big(W, (Y, X_1, X_2)^i\big) \qquad E\ 7$$

Where W is our model (and its weights) and:

$$L\big(W, (Y, X_1, X_2)\big) = (1 - \mathrm{Y})\mathrm{L_S}(D) + Y L_D(D) \qquad E\ 8$$

With L_S and L_D loss functions designed so that minimizing L with respect to W would result in low values of D for similar pairs and high values of D for dissimilar pairs.

$$L_S(D) = \frac{1}{2}(D)^2$$
$$L_D(D) = \frac{1}{2}(\max(0, m - D))^2 \qquad E\ 9$$

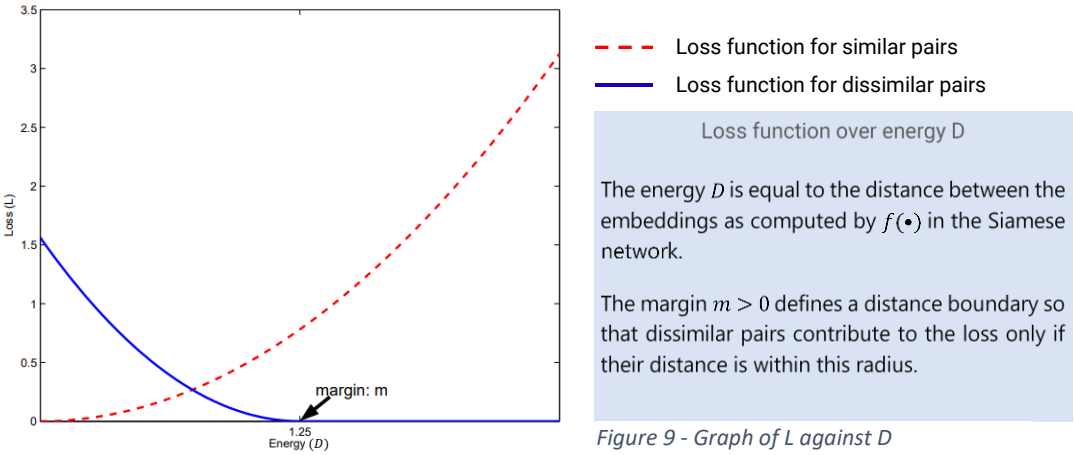


Figure 9 - Graph of L against D

By using triplet loss, instead, the model is shown two pairs of images per training iteration, where one of the images is the same for both pairs and is called anchor point. The other image in the pairs is a similar image in one pair and a distant image in the second. Triplet loss tries to minimize the distance between the similar samples and at the same time maximize the distance between the different samples. This allows for a more relative concept of distance and greater elasticity of the model.

Given anchor point A, similar input P, and dissimilar input N, the triplet loss function is defined as follows:

$$\mathcal{L}(A, P, N) = (||f(A) - f(P)||_2 - ||f(A) - f(N)||_2 + \alpha\ , 0) \qquad E\ 10$$

Where $f(X)$ indicates the model's embedding for sample X and α is a regularizing margin between similar and dissimilar pairs.

It is clear that these networks can learn to solve tasks relating to equivalence relationships between images. In our work, we extend the capabilities of Siamese networks to not only solve equivalence relationships and give a binary output, but to give a relative ranking metric, hence solving an order relationship between the samples. To do so, we will let aside contrastive and triplet loss and devise a training method of our own.

2.5 Explainable AI (XAI)

The explosion of artificial intelligence in recent years highlights a series of issues that we must solve for the discipline to continue to flourish. Every technology will, at some point, clash with the interaction it has with its human user; for artificial intelligence, this point lies in the complexity of comprehending how the model "thinks" and in understanding where the results come from. This is most important when we apply these techniques to vital tasks such as in the medical field, but it has a strong impact even when utilized in areas of a less critical nature. We can see an example of this in motorsport, particularly in Formula 1, where AWS Insights [11] powers a series of on-screen graphics that show the output of complex data-driven prediction and analytics algorithms. The opacity of such information is very disliked by fans, that have trouble interpreting these results, albeit relevant and accurate.

It is clear that the pervasive use of AI systems must be met by an equally pervasive use of methods able to explain the decisions taken, but the reader may ask why are AI models and in particular deep learning models so hard to understand. The roots of the problem reside in the different way humans and machines learn: deep models could inherently learn or fail to learn representations from the data which a human might consider important. Moreover, neural networks can find structure in any dataset with exceptional performance, even when the dataset is composed of random noise [12]; learned features can be completely meaningless to a human, making it impossible to analyse and debug the models. This problem is made worse by the strong driver identified by results: the end user is often more focused on getting accurate solutions than understanding the inner workings of the system; engineers are thus pushed to allocate less resources to the scope of explanations. This is further worsened by the trade-off that we find between the performance of a model and the ability of explaining it. Recently, however, there has been a surge in interest towards explainable AI, mainly from governing bodies, that aim at posing the foundations for regulating the transparency, trust, and bias of AI, as well as the impact of adversarial examples on the models; the latter being one of the main concepts explored in this work.

An *explanation*, in the context of XAI, is a way to verify the output decision made by a model. This definition is purposely vague and, for example, might mean a textual description of the reasoning behind the verdict, a heatmap indicative of the pixels that most contributed to the result, or a graph that matches the decision with some knowledge base. Once again, we make the distinction between interpretability and explainability; indeed, the explanation examples given here relate to additional pieces

of information that can have various form, but that are always detached from the raw output of the model.

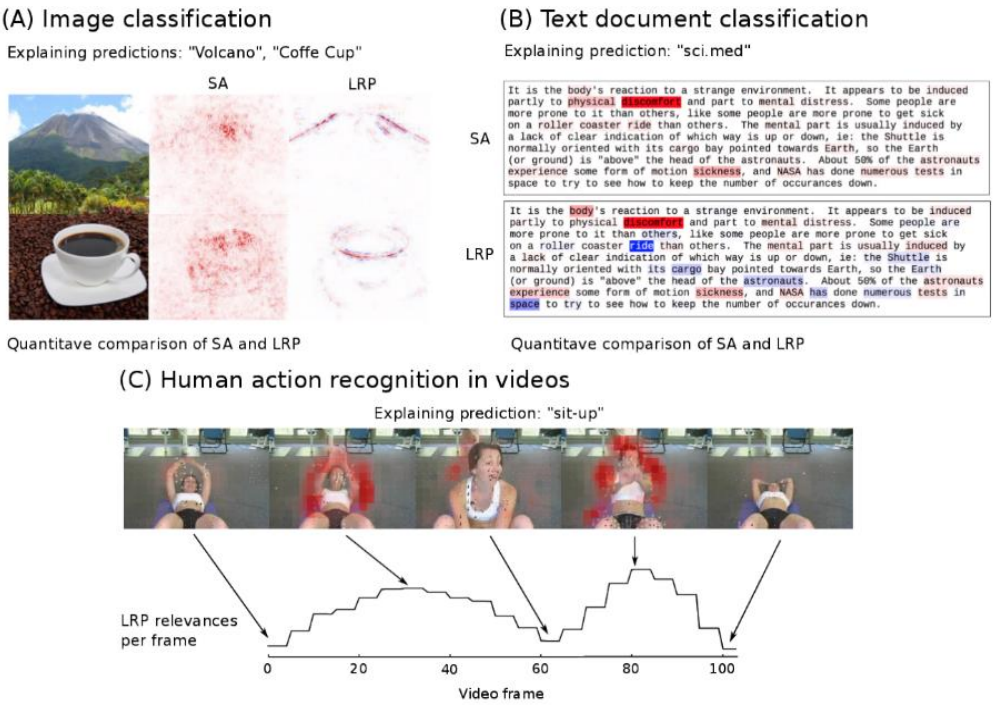
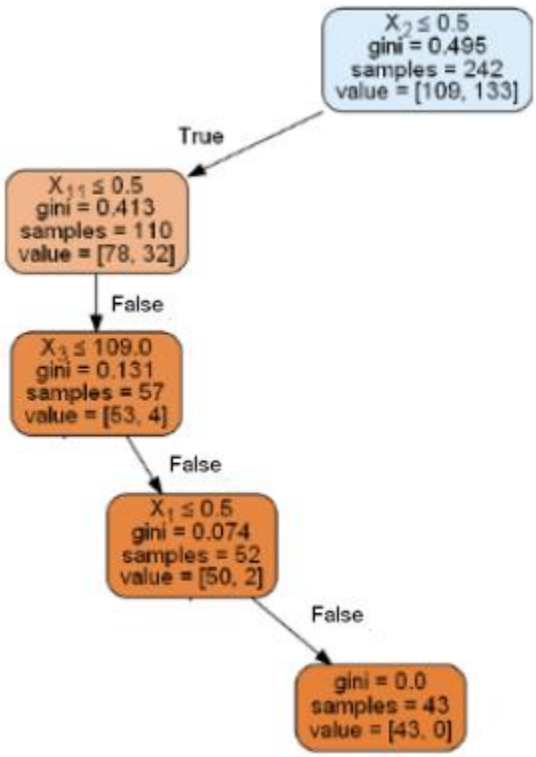


Figure 10 - Example of explanations in the literature [3]

In an interpretable model, instead, the result cannot be disentangled from its explanation. Classic examples of interpretable models are decision trees. Decision trees give an output which is the direct consequence of a finite number of decision iterations, each of which is based on explicit parameters defined on each node of the tree. The result of the decision is deterministically given by the path on the tree relative to the input sample; this path, therefore, defines both the output and the explanation of the different decisions that led to it.

[3] Image credit: W. Samek, T. Wiegand, K.R. Muller; Explainable artificial intelligence: understanding, visualizing and interpreting deep learning models. arXiv:1708.08296v1.



Decision tree path

Each sample that is run through a decision tree follows a path defined by the tree splitting parameters, which are set at each node.

This path, therefore, defines the output of the model but also describes each decision and the parameters that regulated it.

For this reason, the output of a decision tree cannot be isolated from its explanation; this renders decision trees an interpretable class of models.

Figure 11 - Decision trees as interpretable models [4]

2.5.1 Taxonomy

We begin our exploration of explainable AI by defining a taxonomy of explanation techniques that have been used in recent research [1]. This taxonomy groups explainability methods over three axes: scope, methodology and usage.

Scope

The scope of an explanation relates to whether the objective is to explain why a specific output came from a certain sample, in which case we say the scope to be *local*; or whether we want to describe how the model takes decisions in general, regardless of a specific input sample, in this case we say the scope to be *global*.

A locally explainable model's outcomes might be described by feature attribution, for example by giving a score for each word in a text that contributed to the output. A global explanation could instead provide a decision tree that describes the model reasoning.

Methodology

Methodology refers to the algorithmic concepts that are used to construct the explanations. In general, two main approaches have been used in the literature, the first regards *backpropagation-based* methods, while the other techniques are *perturbation-based*.

[4] Image reference: Towards data science, Interpretable Machine Learning Models, Hennie de Harder, https://towardsdatascience.com/interpretable-machine-learning-models-aef0c7be3fd9

In *backpropagation-based* methods, one or more backpropagation passes allow to construct attribution maps by using the partial derivatives of the activations. It is in fact possible to back-trace the specific output of one neuron in a deep neural network by computing:

$$W = \frac{\partial S}{\partial z} \qquad E\ 11$$

Where W represents the importance of the activation of some neuron z in predicting output S. This is used, for example, in computing attribution maps in image classification models, where we can compute:

$$W_{x,y}^{c} = \frac{\partial S^{c}}{\partial I_{x,y}} \qquad E\ 12$$

Where the importance $W_{x,y}^{c}$ of pixel in location x, y of image I can be computed with respect to how it affects the SoftMax score S^{c} for class c.

In *perturbation-based* methods, the original input A_o is modified, either randomly or in a targeted manner, in order to show how changes in the input reflect in the output. This class of explainability techniques can make use of a wide range of alterations on the input, depending also on its form. The result for processes of this class is typically a perturbed input A_p that can then be fed to the same model M. The model's output $M(A_p)$ can then be compared against the original output $M(A_o)$. The explanation is entailed in the differences in the inputs and in the relative outputs.

Usage

Usage refers to how the method is developed. An explainability technique could be built-in and specific to a model, in this case we say that it is *intrinsic* or *model-specific*; it could be appliable to a specific class of models (for example CNN classifiers), making it *model class-specific*; or it could be independent of the model's architecture and applied externally to already trained networks, in this case we call the technique *post-hoc* or *model-agnostic*.

Our work will focus on a local, intrinsic technique, that will however overlap over both backpropagation-based and perturbation-based methodologies.

The concept of explanation is grounded in social science research, and because the target is ultimately a human person, we might want to translate some of the properties of a correct explanation into strictly computer science requirements that can be verified over the developed system. In [13], these are materialized as eleven functional properties over which we can assess the validity of an explanation technique. Most importantly, an explanation must be *sound* and *complete*: the output should be adherent to the underlying model and the explanations should be consistently generalizable. Explanation of similar samples should indeed be similar if the model's reasoning was similar.

Another important property of an explainability technique is *actionability* and it regards the usefulness of the method. The explanation should thus provide information that is

relevant for a human and upon which the user may act. This could mean a suggestion of action if the system is prescriptive, for example in a loan-granting system; or an insight on the functioning of the model that could be used to debug it or improve its performance. This property is strictly related to that of *novelty*, which is also a core property in every machine learning algorithm; in fact, providing the user with an expected explanation is not more useful than just giving the original output. This does not mean that an explanation should be always unexpected, but that the explanation should be able to detect and display abnormal behaviour, emphasizing any instance in which the model acts in unusual ways.

The *complexity* of an explanation strictly depends on its target user. For example, if the objective of explanation is debugging of the model, for example to find flaws in the dataset, in the learning process, on in the architecture itself, we can expect the user to be proficient in analysing deep neural networks. Therefore, if the target user is an expert, we can provide a more complex explanation that can be more informative but that is less clear to the uneducated user, which might instead need a more simple explanation, albeit with less information potential. We might want an explanation method that can be *personalized*, for which the output is tailored to who the audience is. In every case, however, we would like the output to have *parsimony*; this means that the explanation should be selective and never be overwhelming, or else there would be the risk of making the explanation so complex that it could become opaque in itself, thus requiring an explanation of the explanation.

In the context of real-world systems, providing explanations could pose several security risks. Indeed, an attacker might want to obtain information about the system's functioning in order to exploit its vulnerabilities. Because neural networks can be very fragile, explanation methods must be conservative with which data they leak. Failure to do so could allow attackers to subvert fraud detection, malware detection, or mislead autonomous navigation systems [14, 15]. These concepts have however been disregarded in this work, as we will focus on prototype models to better understand the general working of CNNs.

2.5.2 Saliency maps

The output of an explainability technique can have many forms; saliency maps being one of the most popular within the methods where pictures are the input. A saliency map is an image, likely of the same size of the input one, that visually highlights the contributions of sections of the input to the model's result. A plethora of practices have been developed in recent years, most notably with class-activation maps [16] and subsequent improvements, such as [17]. These techniques allow to see which parts of the image were most responsible for a particular classification, revealing where the model focuses and possibly showing mistakes the network made in the prediction. The explanation is given by superimposing the generated saliency map over the input image: the saliency map will show darker (red) shades over regions the network deemed useful in the decision, and lighter (blue) shades where the features of the images were not taken into consideration for the decision. As we can see in the

example, a network might recognize a shark only by its teeth and disregard the other features. In this case we might realise that the dataset is biased towards front facing shark pictures and that there is no training regarding sharks in other orientations. In this case, the explanation can be useful to scientists to fix issues relative to the training.

Figure 12 - Saliency map for class shark

It is clear that saliency maps are a great way to visually explain what the model is doing; however, these explanations are only reasonable in the eye of the examiner and they might not show what the network is actually focusing on. For example, a saliency map might be built such that it shows structures of the input independently of the model and its flaws. For this reason, we might want to check for the validity of such techniques. In this work, we check our saliency maps following the findings of [18], where a series of so-called *sanity checks* were proposed to assess the validity of techniques that produce this kind of output. In particular, the two main concepts introduced are those of *model parameter randomization test* and that of *data randomization test*. The former is based on the idea that an explainability technique's results must depend on the model parameters to truly explain something about the model. An explainability technique that does not depend on the model's parameters will produce results that might only describe structure of the input image and may never be used to debug or evaluate the network's reasoning. The second test is instead focused on checking the dependency between input and labels, the underlying principle being that an insensitivity to the permuted labels reveals that the method does not depend on the relationship between instances and labels that exists in the original data. The latter is performed by retraining the model on a different version of the dataset where the labels are scrambled, finally checking whether there is difference in the explanations between the two trained versions of the model.

The findings of [18] show how even some established results in the literature, such as guided back-propagation and guided GradCAM have a very limited explainability potential, comparable to that of edge detection in the input image. This is shown in *Figure 13*, where results for the *cascading randomization* procedure are displayed. During this test, each layer of an inception v3 network was gradually (layer-wise) replaced with an untrained (randomly initialized) copy of itself. This process aids in the execution of the *model parameter randomization test* and shows how the explanation changes for different techniques when the model's parameters change.

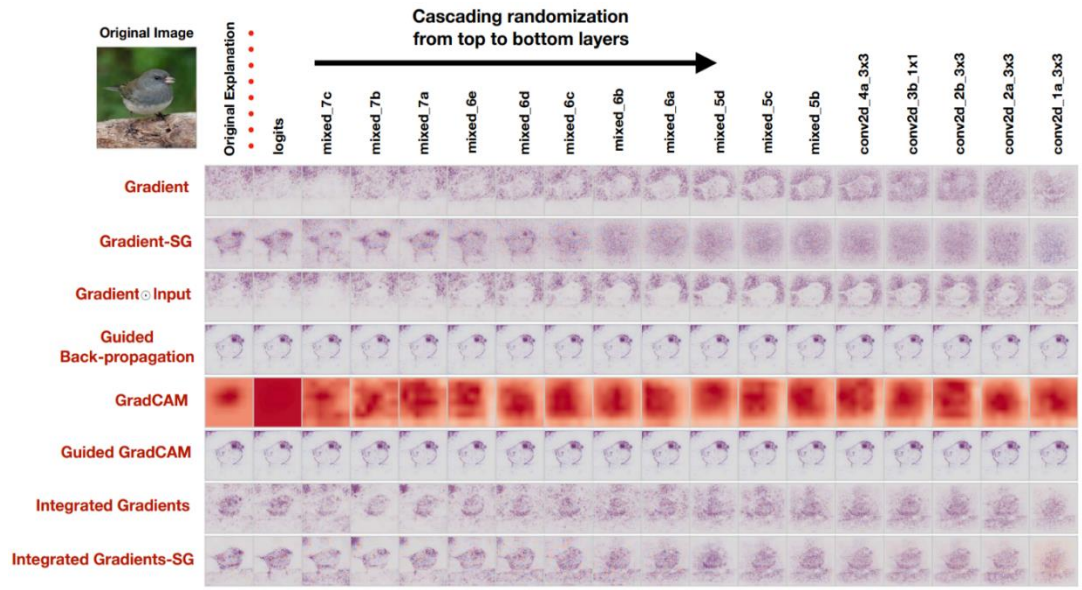


Figure 13 - Model parameter randomization test for different explanations

We see how the explanation for the two techniques listed before does not significantly change when the model layers are randomized, displaying the ineffectiveness of the explanation. Visual inspection is not however enough; the lack of change must in fact always be backed up by analytical processes.

In this work, the same checks have been carried out over the saliency map techniques that were developed, to avoid falling into the same pitfalls shown here. As it was the case in [18], evaluation was performed through Spearman rank correlation between pairs of images.

2.5.3 Perturbation based explanations

In the taxonomy section, we distinguished between *gradient-based* and *perturbation-based* methodologies; in this section, we will explain in more details what are the *perturbation-based* explainability techniques.

A perturbation is a modification of the input that might result in a change in output when fed through the network. A virtually unlimited amount of perturbations can be imposed on the input; explainability techniques, however, mostly limit their reach to alterations that create a significant change in the output with only slight changes in the input. Like for saliency maps, these methods can display which features are most important for the model's decision. In fact, we assume that if a feature is irrelevant to the model's output, changing it will not affect the result; instead, a relevant feature will be critical, hence changing it will strongly affect the output.

Perturbations can be of various nature; the methods used in the literature include partially occluding the input, for example in images, to show if a portion of the input is particularly relevant; permuting input sequences, to check if the ordering of specific input tokens is vital, especially for models that receive multiple symbols such as translators or caption generators; or using generative algorithms to generate synthetic samples that have specific features re-tuned to see if that particular visual feature was important for the decision.

A classic example of perturbation-based explainability comes from data mining and is that of partial dependence plots (PDP). This technique aims at showing how a specific numerical feature affects the model's prediction. The procedure works by iteratively running the model on modified versions of the same dataset where, for each sample, the feature in analysis is set to a different value for every iteration. The model's output is then compared to display how predictions change based on the feature value that was set for the various iterations. Modifying features with less relevance for the model will not result in change in the output; instead, changes to important features will. This method will also show numerically how this features actually impact the results.

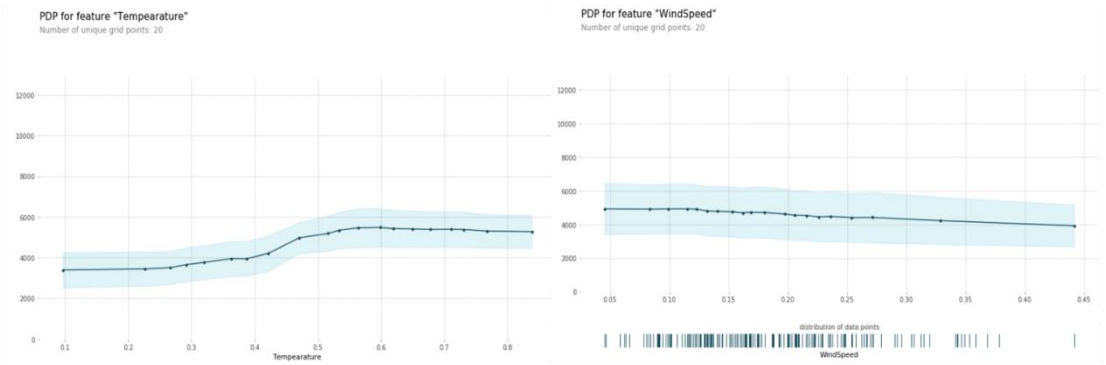


Figure 14 - Partial dependence plot for bike rental

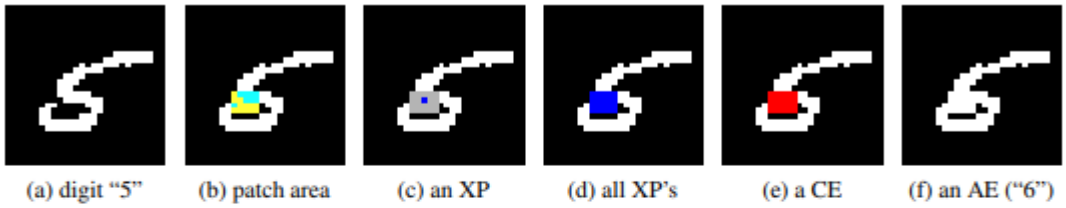
In *Figure 14*, we see an example of PDP for a model that predicts how many bikes will be rented in a day based on weather and other parameters. We see how the model, regardless of other feature values, will give a higher prediction if temperature is higher (values are normalized in the example), and a slightly decreasing prediction for increasing wind speed. We can hence say that temperature has a higher impact on the model's prediction than wind speed and visualize which are the values of these features for which the model will predict a higher value. In the example, we see confidence intervals because the test is run on the whole dataset; predictions for a given assigned feature value will hence be many: one for each sample.

In our work, we will use targeted perturbations to produce saliency maps that will give information about which portion of the image was most relevant for the decision.

2.5.4 Adversarial attacks

The immense number of parameters that make up a deep learning model makes it possible to learn very difficult tasks; these emerging computational capabilities are however not all under our control. Firstly defined in [19], among these intriguing properties, we find that the smoothness assumption that underlies many kernel methods does not hold for neural networks. This means that imperceptible changes in the input could lead to drastic changes in the output, allowing for noise to be weaponized against these models. This is the underlying principle behind adversarial examples and adversarial attacks: we can follow the model gradient to optimize for its error, thus finding a small perturbation of the input that will cause a significant shift in the output.

Neural networks continue to work well even when adversarial attacks are possible because these models are trained and used with images that come from the real world. The adversarial examples represent low-probability pockets in the feature space for which the network never trained, and which it will very unlikely see when used. This property, therefore, only poses a risk when modifying the output of the model can give some benefit to a potential attacker.



(a) digit "5" (b) patch area (c) an XP (d) all XP's (e) a CE (f) an AE ("6")

Figure 1: An example of digit five.

(a) digit "6" (b) patch area (c) an XP (d) all XP's (e) a CE (f) an AE ("5")

Figure 15 - Breaking explanation with counterexamples

Even though adversarial attacks are a clear indication of the shortcoming of current neural networks, we can use this property to our advantage, as adversarial examples can give us insight on which are the blind spots and which are the focus points of a model, thus allowing for the generation of an explanation. This is shown in [20], where the authors prove how each explanation breaks every counterexample, and each counterexample breaks every explanation. In [20], the authors define an explanation and a counterexample as a subset-minimal set of implicants, in other words a set of feature values assignments, such that the model will surely classify respectively the currently predicted class or a different class. Breaking such a set means to have another set which contains a contradiction with the current explanation or counterexample; the main result being that a consistent set of implicants that simultaneously breaks all explanations is a counterexample, and vice versa.

These findings are substantiated by experimental results on a handwritten digit classifier. The results show how modifying the pixels that were most relevant for the explanation creates a counterexample that breaks it, generating a new sample that, if fed through the same network, will result in a different output class. This highlights a clear duality between explanations and adversarial examples; in this thesis, we will make use of this correlation to generate saliency maps based on the vulnerabilities of the developed models.

Another angle from which this problem could be tackled was also explored in [21]; whereby using models that are very resistant to adversarial examples, the researchers were able to generate semantically meaningful attacks that could show the model's internal representation of classes.

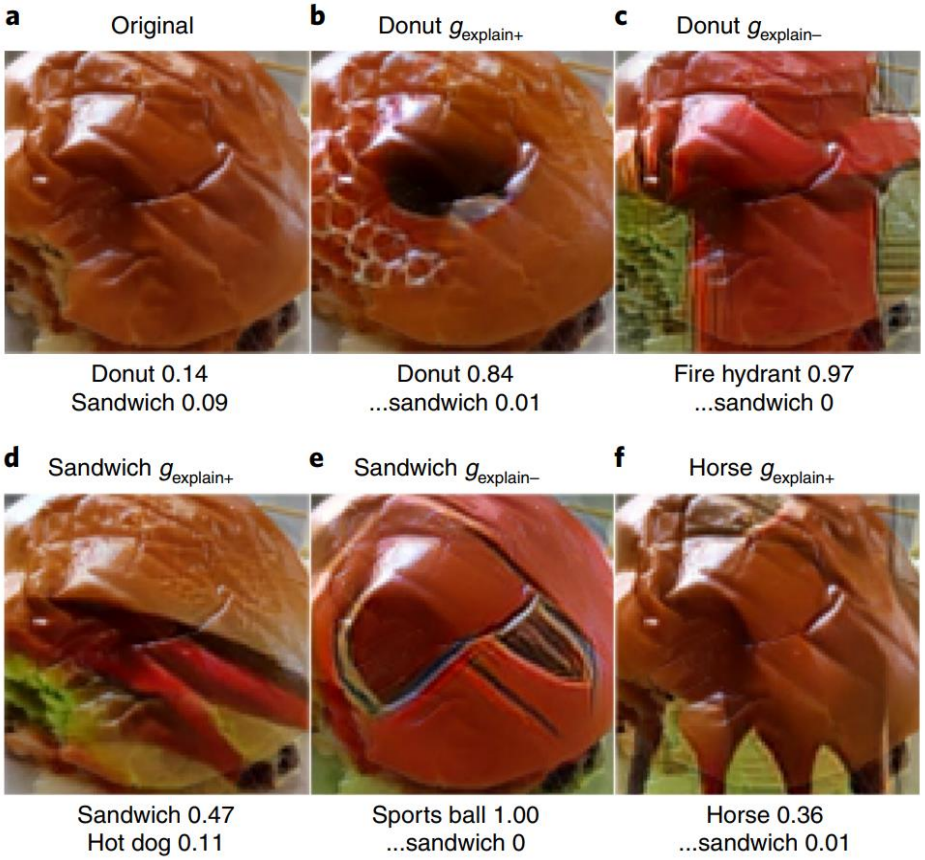


Figure 16 - Generating semantically meaningful adversarial examples

In the image, we see how fooling a very resilient model requires substantial modifications to the input, so much so that the input starts to show features of the class we are trying to aim for in the output. For example, this significant perturbation adds "legs" to a sandwich image when the example is driven to give the result class "horse". This is forced by the model's internal representation of the target class and can give us insight on how the network perceives objects, thus giving us a possibly significant explanation.

3 Problem formulation

The goal of this work is to produce novel explainability techniques concerning deep learning models that focus on binary rankings over images. Our objective is to build a network m that firstly is able to solve the task given to it, and secondly that it is able to consistently explain what it perceives. We theorize that such class of models could be able to give a better insight into their functioning whilst maintaining the high performance expected from a standard network. In this section, we will formally describe the environment we will build upon.

3.1 Ranking problems

A ranking is a relationship between a set of objects $a_1, a_2, \dots, a_n$ such that, for any pair of items a_i, a_j in the set, either the first is 'higher than', 'equal to', or 'lower than', the second. We define ranking also as the process of sorting a set of objects according to their relationship.

3.2 Binary ranking over images

Given a pair of images $A, B \in \mathcal{M}^{h,k,3}$ of height and width respectively h and k and three colour channels (RGB), and an order relationship "$\succcurlyeq$", binary ranking is the process of sorting the two images according to their relative relationship. In *E 6*, we have defined the result of this process as a new tuple of same dimension as the input tuple where the elements are ordered according to "$\succcurlyeq$". For binary ranking, however, the problem is reduced to predicting whether $A \succcurlyeq B$ or $B \succcurlyeq A$. This problem can therefore be translated to a binary function where the result 0 would indicate that $B \succcurlyeq A$ and 1 would indicate $A \succcurlyeq B$. Ties can be broken arbitrarily, but in our work we will assume that $A \succ B$ (strictly) and return 1.

$$r^{binary}: \mathcal{M}^{h,k,3} \times \mathcal{M}^{h,k,3} \to \{0,1\} \qquad E\ 13$$

An example of such class of models is a model r^b which, given a pair of images of two cars, is able to return a result indicating whether the first image is of a car with a higher top speed than the second, in which case it would return 1; or return 0 if the second image depicts a car which is faster than the one in the first sample.

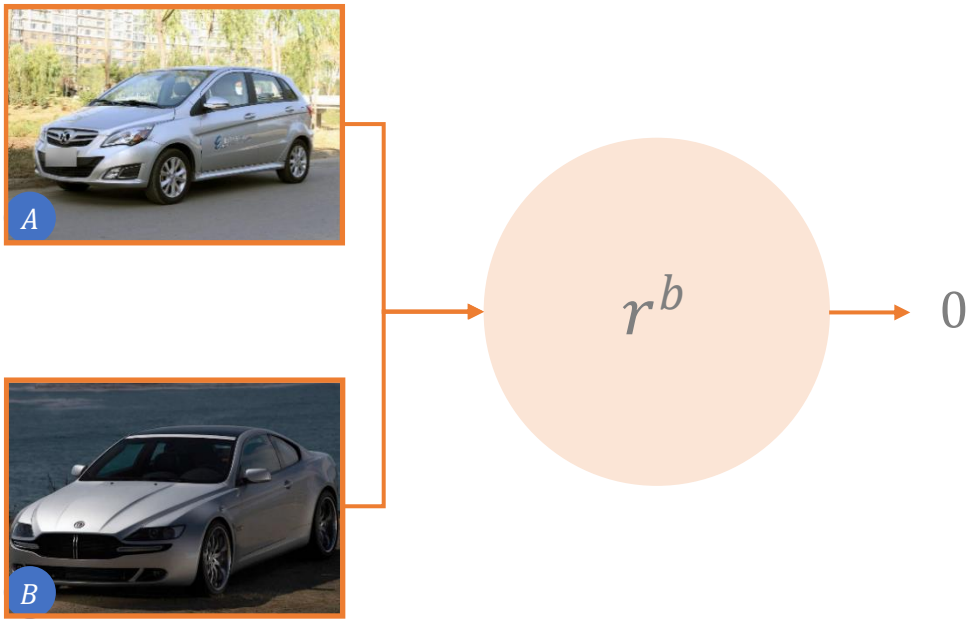


Figure 17 - Example of model for binary comparison

For order relationships where we can define a distance metric, this concept can be extended to provide a fine-grained ranking, which results in a real number in $[0,1]$ indicating the relative position of A and B. The boundaries of the order relationship metric are defined by its maxima and minima. Values lower than 0.5 indicate that $B \succ A$, values above instead indicate that $A \succ B$. The distance from the mid-point give information about how much A is superior or inferior to B with respect to the order relationship metric. In this case, the ranking is a function:

$$r^{metric}: \mathcal{M}^{h,k,3} \times \mathcal{M}^{h,k,3} \rightarrow [0,1] \qquad E\ 14$$

In the running example, we can define a distance metric as the difference in speed, normalized to the lowest and highest top speed of cars that we find in our dataset. If, for example, values are set in the range $[50\frac{km}{h}\ \ 400\frac{km}{h}]$, and the cars in the sample have a top speed of respectively $150\frac{km}{h}$ and $250\frac{km}{h}$, we expect our model to give a result that states the difference in speed of the sample cars proportionally to the range they are set in. We compute this value as the relative position of the two values in the range, and we obtain:

$$v_a = 150 - 50 = 100$$

$$v_b = 250 - 50 = 200 \qquad E\ 15$$

$$target = \frac{v_a - v_b}{(400 - 50) * 2} + \frac{1}{2} = 0.36$$

3.3 Explanations

Given a model r that performs a binary ranking and an input pair A, B given to the model, an explanation is a piece of information E that is external to the raw outcome $O = r(A, B)$ of the ranking function, and that can help a human user in understanding it. An explanation function for a ranking model is a function that returns such piece of information, and is defined as:

$$s(r, A, B) \mapsto E \qquad E\ 16$$

To be informative, the explanation must depend on both the model r and the current input pair A, B; additionally, the explanation must also be coherent with the model's output O. Following the findings of [20], its validity can be checked against its dependence on the input parameters and its consistency with respect to the outcome O. Insensitivity to one of the input parameters or irrelevant results imply ineptitude of s in giving useful additional information.

3.4 Saliency map

A saliency map is an explanation function where the additional information E has the form of an image with the same size as the input images A, B, such that $E \in \mathcal{M}^{h,k,3}$. The resulting image has the property of directly highlighting how the input affected the model's outcome. A saliency map is therefore a function:

$$s(r, A, B) \mapsto E \qquad A, B, E \in \mathcal{M}^{h,k,3} \qquad E\ 17$$

In the example, a saliency map might be a picture of a car where the most relevant features that drove the model's decision are highlighted with colour.

Figure 18 - Example of saliency map

4 Proposed solution

4.1 Ranking model

We implement image binary ranking as a function r_ϕ with parameter vector $\phi \in \Re^m$ that takes as input two images $A, B \in \mathcal{M}^{h,k,3}$ and computes a result in $[0,1]$ that represent the relative ordering according to $\succcurlyeq$ between A and B where the order relationship "$\succcurlyeq$" is the ranking.

$$r_\phi(A,B) \longmapsto [0,1] \qquad \textit{E 18}$$

The output represents the distance between the two samples according to the metric that can be defined over "$\succcurlyeq$", but it can also be interpreted as a confidence of the model in its decision.

An intermediate result of the ranking model are the embeddings: composed of *n*-dimensional feature vectors, one for each image, they characterize how the model perceives the input. Because we want to analyse the phenomena related to the inner understandings of the model, we require the latter to also expose this piece of information. We finally have:

$$r_\phi\colon\ \mathcal{M}^{h,k,3} \times \mathcal{M}^{h,k,3} \to [0,1] \times \Re^n \times \Re^n \qquad \textit{E 19}$$

As we will see later, we would like to modify the embeddings to see how these changes affect the prediction and the saliency maps that derive from them. For this purpose, we expand the function r_ϕ into functions w_ϕ^e and w_ϕ^r that compute the embeddings and the ranking prediction, respectively. We have:

$$\begin{gathered} w_\phi^e\colon\ \mathcal{M}^{h,k,3} \times \mathcal{M}^{h,k,3} \to \Re^n \times \Re^n \\ w_\phi^r\colon\ \Re^n \times \Re^n \to [0,1] \\ r_\phi(A,B) = \left\{ w_\phi^r\left(w_\phi^e(A,B)\right), w_\phi^e(A,B) \right\} \end{gathered} \qquad \textit{E 20}$$

We will use notation $w_\phi^e(A)$ (with a single input variable) to identify a single embedding result for function w_ϕ^e. The notation is intended as:

$$w_\phi^e(A,B) = \left\{ w_\phi^e(A), w_\phi^e(B) \right\} \qquad \textit{E 21}$$

4.2 Perception visualization

We want to develop an explainability technique which can fall into the saliency map category, but which is not a simple heatmap of the image regions that were most significant. We want to also be able to describe how changes in these regions may impact the model's result. To do so, we will make use of a new class of map that we call *perception visualization*. A perception visualization is the result of a function p_ψ with parameter vector ψ that projects the embeddings generated by the model r_ϕ back into image space. The input will thus be an embedding in $\Re^n$, and the output an image of the same size as the images that are input to the original model r_ϕ.

$$p_\psi\colon\ \Re^n \to \mathcal{M}^{h,k,3} \qquad \textit{E 22}$$

Because this visualization aims at providing an image that characterizes the inner representation of the original images A, B from the point of view of r_ϕ, it is clear that the parameters ψ of p strictly depend on the parameters ϕ of r. Also, to allow p to generate the best possible explanations, we might want r to provide embeddings in a meaningful form so that they can be easily projected to images; thus making r dependent on p. We finally find that the ranking model and the perception visualization are mutually dependent.

We have shown that r and p are inherently entangled; we hence define a function m_θ where $\theta = \{\psi, \phi\}$ which combines them and represents an interpretable model able to solve the ranking model, and simultaneously provide a way for a human to understand the decisions taken. In this case, the perception visualizations and the other saliency maps are a way to give this information to the user. The input will be the same input given to r, while the output is the result of the ranking prediction and the output of the perception visualization for each of the input images.

$$m_\theta: \ \mathcal{M}^{h,k,3} \times \mathcal{M}^{h,k,3} \rightarrow [0,1] \times \mathcal{M}^{h,k,3} \times \mathcal{M}^{h,k,3} \qquad E\ 23$$

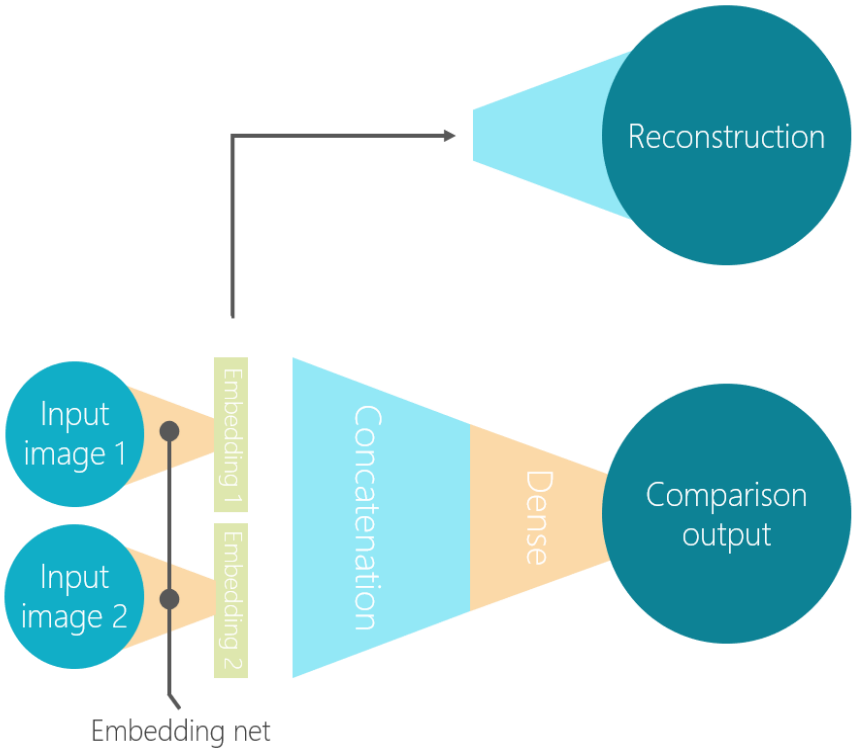


Figure 19 - The embedding is used for both ranking prediction and reconstruction

We call this class of models *semi-interpretable* because they provide an explanation together with the output, like an interpretable model would, but this explanation is not entailed in the output itself; it is in fact a detached piece of information that the network gives in addition to the prediction result.

In practice, this model can be optimized for both the parameters ψ and ϕ, thus generating significant explanations and accurate rankings; to this end, we want the model to be trained simultaneously to solve the task and to give explanations. We will see later how this simultaneous training can be achieved and which are the consequences of our design choices.

We build a convolutional Siamese network in which the branches are copies of a CNN. The result is a feature vector that can then be fed to a single or multilayer perceptron and solve the ranking problem. The same convolutional neural network can however be thought also as an encoder; feeding the same feature vector to a convolutional decoder builds an autoencoder inside the ranking model, therefore completing the *semi-interpretable* network.

Testing the trained model on this notion alone would however not give us more insight than just using a standard autoencoder. To really grasp the understanding the network has of the task it has to solve, we must resort to perturbation analysis.

What we propose is a technique to explore the latent vector space of the embeddings and then project it to a space that is similar to the input space to see how the network encodes the features that are relevant for the ranking task. This will tell us, in a human understandable way, if and what the network cognized. In *Algorithm 1* we describe the procedure that allowed us to do so. We start by feeding two input images A, B through the network r_ϕ (line 3), obtaining a prediction c and two embeddings e_1, e_2 of the input images A, B respectively. We can then compute (line 4) the reconstruction h_1 that should be similar to the input image A. After this, we iteratively modify (lines 5-10) one of the embeddings (e_1) by following the gradient d of the loss $\mathcal{L}$ with respect to one of the embeddings: this drives the output c towards the increasing or decreasing direction. The other embedding (e_2) is kept fixed. This procedure allows to have a very small change in the embedding e_1 resulting possibly in a very large shift in the prediction value c; we can then analyse the perception visualizations (line 11) of these perturbed embeddings to see how they changed when we forced them to provide a different result.

set user defined $\boldsymbol{target}$, $\boldsymbol{iterations}$, $\boldsymbol{\lambda}$
set input images $\boldsymbol{A}, \boldsymbol{B}$

$\boldsymbol{c}, \boldsymbol{e_1}, \boldsymbol{e_2} = r_\phi(A, B)$
$\boldsymbol{h_1} \leftarrow p_\psi(e_1)$ // reconstruction from original

for **i** in 1:iterations
    $\boldsymbol{c} = w_\phi^r(e_1, e_2)$
    $\boldsymbol{\mathcal{L}} \leftarrow \mathrm{MSE}(c, target)$
    $\boldsymbol{d} \leftarrow \left[\nabla \mathcal{L}_{|c}\right]_{e_1}$ // Gradient in c, restricted to dimensions of e1
        $e_1 \leftarrow e_1 + \lambda d$ // Hyperparameter λ determines the step size
end

$\boldsymbol{h_1'} \leftarrow p_\psi(e_1)$ // reconstruction of modified embedding

Algorithm 1 - Perturbation analysis from embeddings

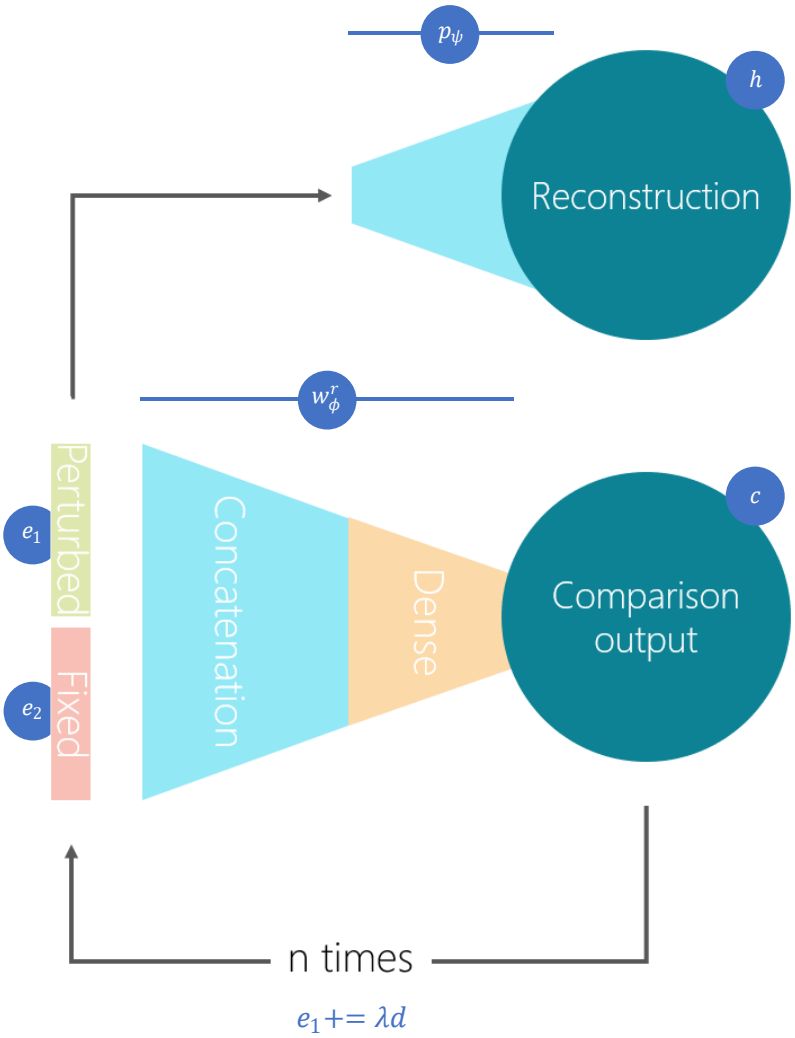


Figure 20 - Visualization of the base perturbation procedure

The modifications that are propagated to the reconstructed images are instructional in understanding whether the network has learned the concepts required to solve the ranking problem in the same way in which a human would do. This process also tells us what features the network most cares about; if some visual features are kept intact during the procedure, we can say that they are irrelevant for the model; otherwise, we would see a sharp change in the final reconstruction. Finally, we might assess how these changes are structured: we might see very natural changes, suggesting that the network has learned notions that are analogous to those that drive our instinctual comprehension of the problem, or very artificial changes, pointing not only at different ways of perceiving the input, but also at possible errors in the network's learning process. For example, we might observe a network that modifies visually irrelevant portions of the inputs, such as black or otherwise very feature poor sections; this might shed light on errors in the dataset as the network might be focusing on noise. In contrast, a network that is well trained will provide changes to more feature rich portions of the input, and most importantly to those features that provide the most information regarding the task.

The perturbation seen in *Algorithm 1*, however, is limited to the latent space and may be hard to trust, as there is no reference with respect to the input. Therefore, we would like to frame these methodologies to the original image and to more well-known techniques. To do so, we establish a different methodology that can be developed starting from the one just defined, that allows to produce the same modified reconstructions, but can simultaneously produce a saliency map in the input space. This saliency map (obtained by the perturbation generated through *Algorithm 2*) and the perception visualization can then be compared in order to prove that the developed technique is sound. We however argue that the addition of the perception visualization renders the explanation much more informative, as it shows not only which portions of the image are important, like a simple activation map would, but it also shows how changes in these regions affect the network's understanding of the input.

The idea is to build an adversarial attack by directly perturbing the input in the same way done before with the embedding. We use the attack image as an explanation by comparing it to the original image and highlighting the regions where the two samples most differ. This lets us locate the portions that most contributed to the prediction, following the duality between explanations and counterexamples. The procedure is detailed in *Algorithm 2*. The first steps of the procedure (lines 1-5) are identical to those seen in the previous algorithm, and are required to obtain an initial comparison prediction c and the embeddings e_1, e_2 of the input image pair A, B. In line 6, we copy the first input image into image A_m that will be object of the procedure. We now iteratively perturb the latter (lines 7-12) by following the gradient d of loss $\mathcal{L}$ with respect to A_m. Throughout the iterations, the perturbation of A_m will drive the output c towards the desired direction, as seen in *Algorithm 1*; also, as the network will receive a different input, the relative embedding e_1 will change. Finally, we can obtain (line 13) the perception visualization h_1' relative to the embedding e_1 and the resulting perturbed input A_m (at the final iteration). The information we are interested in lies in

the difference $A - A_m$ which characterizes the perturbation that was induced in the input image throughout the procedure. In section *Explanation methodologies* we will show the different ways in which this information was processed in order to obtain saliency maps.

We might also induce some natural image bias into the attack procedure to drive it to give additional information. For example, by forcing the modifications to be contained in contiguous patches, we can find the most relevant high-level features of the image, instead of obtaining a sparse explanation that only gives emphasis to the alteration hotspots. These concepts will be analysed in section *Advanced perturbations*.

```
set user defined target, iterations, λ
set input images A, B

c, e1, e2 = r_ϕ(A, B)
h1 ← p_ψ(e1)              // reconstruction from original
Am ← A

for i in 1:iterations
    c, e1, e2 = r_ϕ(Am, B)
    L ← MSE(Am, target)   // Note how the gradient is w.r.t the input
    d ← [∇L|c]_Am         // Gradient in c, restricted to Am
    Am ← Am + λd          // Hyperparameter λ determines the step size
end


h1' ← p_ψ(e1)             // reconstruction of modified input
```

Algorithm 2 - Adversarial attack explanation

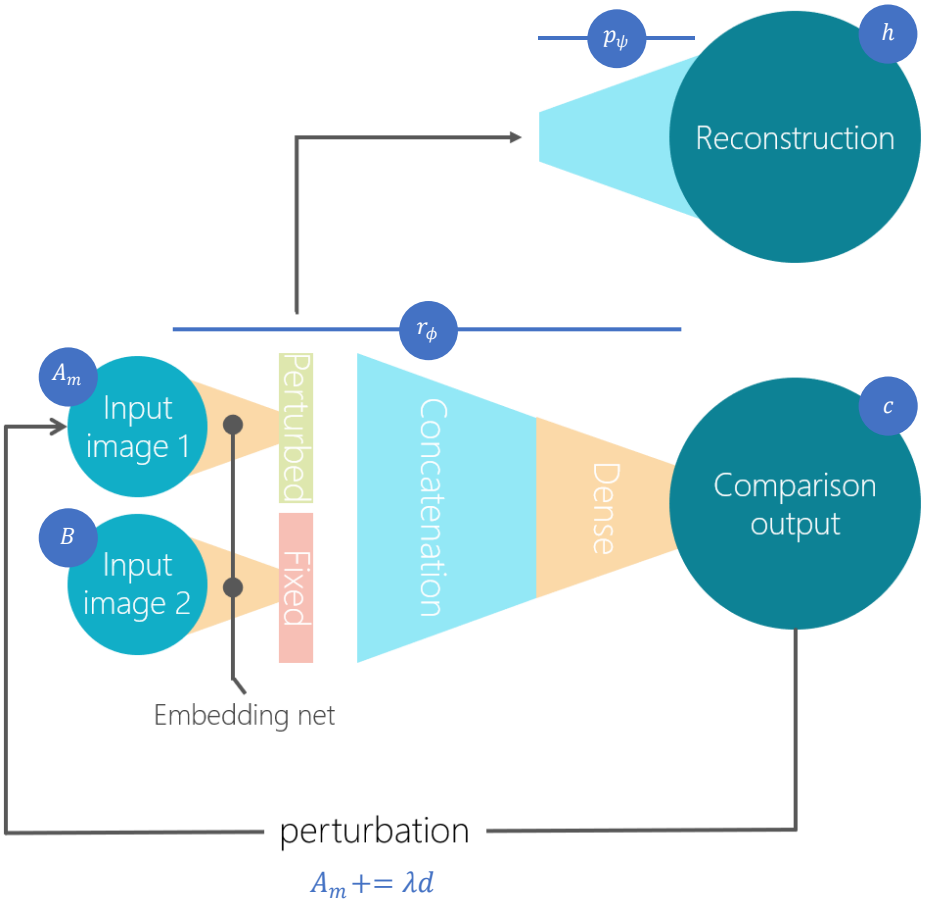


Figure 21 - Direct perturbation by adversarial attack

What we theorize is that if the network has a natural awareness to the images it processes, then the salient features we find by performing the adversarial attack are also the features that are then modified in its perception reconstruction. We want to find whether there is correlation between the perturbation $h_1' - h_1$ obtained in perception visualization, which is generated by the model, and the perturbation $A - A_m$ (with A_m at the final iteration), which is directly performed on the input by the algorithm. If so, we would like to analyse such correlation and use this information to improve the explanation value of the adversarial attack technique.

4.3 Advanced perturbations

For both algorithms previously described, we always optimized for a loss function that only took into consideration the target outcome of the prediction. Given the inherent nature of the input images, we can impose some *bias* (a parameter that drives the algorithm into favouring perturbations that satisfy some property) into the perturbation process in order to obtain a better perturbation from the adversarial attack. To apply these concepts to our algorithm, we will modify the loss function so that the perturbation will be driven to favour the desired properties that compose the bias. This might seem to defy the purpose of explaining the genuine decision-making process of the network, but following the duality between explanations and adversarial

attacks, as long as we obtain a valid counterexample, the explanation resulting from it is legitimate.

Imposing bias in *Algorithm 1* might not make sense, as the target of the perturbation is the embedding, which has no purpose in the development of an adversarial attack. In *Algorithm 2*, instead, changing the loss can indeed result in a different A_m, thus giving a potentially different explanation. The proposed changes to the loss function relate to a total variation factor, a pixel change factor and a total difference factor. We define hyperparameters to tune the effects of these three and call them, respectively: $TVForce$, $ZeroForce$ and $DiffForce$.

The first component is grounded on the organic nature of the images we want to analyse with our models. Total variation norm has been extensively used in literature to drive networks to build more natural-looking images; this stems from the fact that natural images are thought to be composed of contiguous patches of similarly coloured pixels. Total variation computes the sum of all the differences between each pixel and their neighbours, both on the horizontal and vertical axis; minimizing this number forces the image's pixels to be similar to their neighbours.

$$TVNorm(E) = \sum_{i=1..a-1, j=1..b-1} \left|E_{i,j} - E_{i+1,j}\right| + \left|E_{i,j} - E_{i,j+1}\right| \qquad E\ 24$$

We want our explanation to be as natural as possible; therefore, we also want the changes to be made in contiguous patches. Minimizing total variation can drive the attack to achieve this result.

The second component aims at forcing the explanation to be less dispersed; we want the least number of pixels to be changed, forcing those that are to be heavily so. This factor counts the number of pixels that have been changed and penalizes the network for a greater count.

The last component focuses on how much the original image changes; we compute the mean square error between the original image and the modified input at the current iteration and increase the loss if the attack is sharply deviating from the original image. This factor helps keep the perturbed image similar to the original one and allows the explanation to remain relevant to the local input space region. This forces the attack to be more efficient, since an efficient attack is one where a very slight modification produces a very strong change in the output.

The resulting loss function to be substituted in the algorithm is:

$$\begin{aligned} \mathcal{L} \leftarrow\ & \mathrm{MSE}(A_m, target) \\ & + TVForce * TVNorm(A_m - A) \\ & + ZeroForce * NonzeroCount(A_m - A) \\ & + DiffForce * MSE(A_m, A) \end{aligned} \qquad E\ 25$$

An alternative way of focusing the explanation to specific patches is that of only allowing the process to modify certain regions of the input image. This is quite easy, as we do not have to substantially change the algorithm. Because limiting the gradient

to specific elements of the input does not change its value with respect to those elements, we can simply compute it using the same loss as above, but at each iteration change A_m only for pixels in the required region. Practically, we will change line 11 of *Algorithm 2* to limit the change to those pixels.

```
11    [A_m]_region ← [A_m + λd]_region   //Modify only the required region
```

Another technique that is very popular in the context of deep learning is that of optimizing by using momentum. Momentum is a property that refers to the previous optimization iteration and it consists in a fraction of the previous variation that was applied to a specific weight in a neural network. This change is propagated to the next iteration to avoid stalling into local minima. We can use the same idea to increase our chances of obtaining a valid counterexample by saving what was the pixel-wise change in the previous iteration and multiplying it by a hyperparameter called $damping$. With this technique, at each perturbation iteration, we update the input image A_m like so:

$$A_m = A_m + \lambda d + d_p \qquad E\ 26$$

Where d and λ are the same as seen in the previous algorithms, and d_p is equal to the gradient at the previous iteration multiplied by $damping$. During the first iteration, the momentum is zero.

```
set user defined target, iterations, λ, damping
set input images A, B

c, e_1, e_2 = r_φ(A, B)
h_1 ← p_ψ(e_1)              // reconstruction from original
A_m ← A
d_p ← 0


for i in 1:iterations
    c, e_1, e_2 = r_φ(A_m, B)
    compute L               // Loss as seen in eq E 24
    d ← [∇L_|c]_{A_m}       // Gradient in c, restricted to Aₘ
    [A_m]_region ← [A_m + λd + d_p]_region   //Update image
    d_p ← d * damping       // Momentum
end


h_2 ← p_ψ(e_1)          // reconstruction of modified input

H ← A_m                 // adversarial attack image computed during loop
```

Algorithm 3 - Advanced perturbation with momentum

In the experiments, we will analyse how the different contributions add up in the construction of the different explanations.

4.4 Mid-point attacks

In the previous sections, we have explained how adversarial attacks can be constructed in order to produce a visualization of the regions of the input that most affected the model's decision. To further base this notion into a solid theoretical foundation, we introduce the concept of mid-point attacks, stemming from the findings of [20].

In [20], a strong relationship is proven to exist between adversarial attacks and explanations. This proof starts from the formalization of machine learning models into first order logic. In this framework, a model is specified by a set of first order logic sentences, where the input is modelled by a set of features $F = \{f_1, f_2.., f_L\}$. Given the domain D_i for each of the features, all inputs to the models will reside in the feature space $\mathcal{F} = D_1 \times D_2 \times ... \times D_L$. In [20], researches show how an explanation can be thought as an assignment in $\mathcal{F}$ (possibly non complete) that assures a specific result of the model. The relationship with counterexamples comes from the fact that we can change features of an input to match those of an explanation of an opposite class, thus creating a counterexample, as the model will now be forced to give a result in the opposite class. In *Figure 15*, we see how we can build explanations for the MNIST dataset starting from counterexamples for a specific digit. In *Figure 22*, we repropose the results shown in *Figure 15* with a focus on our purposes.

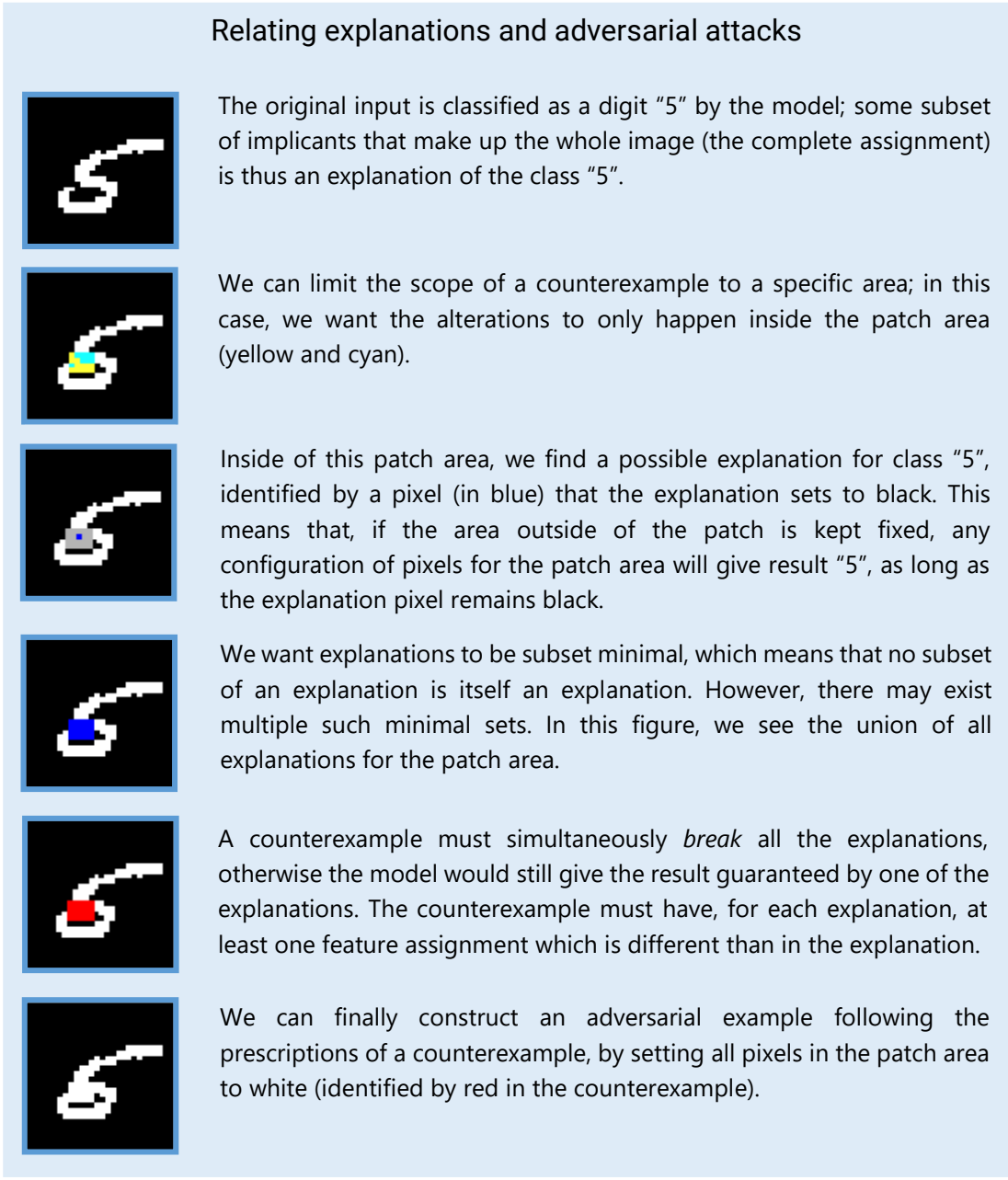


Figure 22 - Relating explanations and attacks [18]

In *Figure 22*, the explanation is obtained, following the paper's procedures, by a subset of feature assignments such that the prediction of the model is certain and equal to a class π (digit "5" in the example). To compose a valid explanation, this set must also be subset-minimal with respect to certainty of model prediction; this means that there must be no subset of the explanation assignment such that we can be certain that the model will still predict class π. This set can be composed of many features, in which case the explanation is not very informative per se, or it can be contained, highlighting features that are critical for the model's decision.

The main takeaway for our work is that an adversarial example for a sample A of class π displays features that are different than those of explanations for the sample. Furthermore, this work shows how we can define an explanation to be a set of

implicants, in our case a set of pixel colour values, such that the model predicts a specific class. Even though our models have not been thought as classifiers until now, it is possible to fit our work into this theoretical framework by thinking about our model as a binary classifier that discerns whether sample A is greater (or equal), or lower than sample B with respect to the order relationship "$\succcurlyeq$". It is easy to see how binarizing the result of our model is enough to transform it into such a classifier; this is akin to the kind of model presented in *E 13*.

In our setup, we will create adversarial examples through gradient descent; after this, we can compare the original sample with the perturbed one and analyse their differences. Because the adversarial example must display features that are different than those of the explanation for the original sample, we can infer the explanation by analysing the pixels that were changed during the perturbation process.

The method to obtain an attack that follows these concepts is similar to the process seen in *Algorithm 2*, where instead of pushing the result towards the edges of the range, we force the modified input to result in a prediction exactly equal to 0.5. We call this procedure mid-point adversarial attack. The result is an imperceptibly perturbed input that gives a result on the boundary of the two classes our model is trying to distinguish. In this scenario, for each pixel the output gradient is sensitive to, a slight modification of such element can change the classifier prediction.

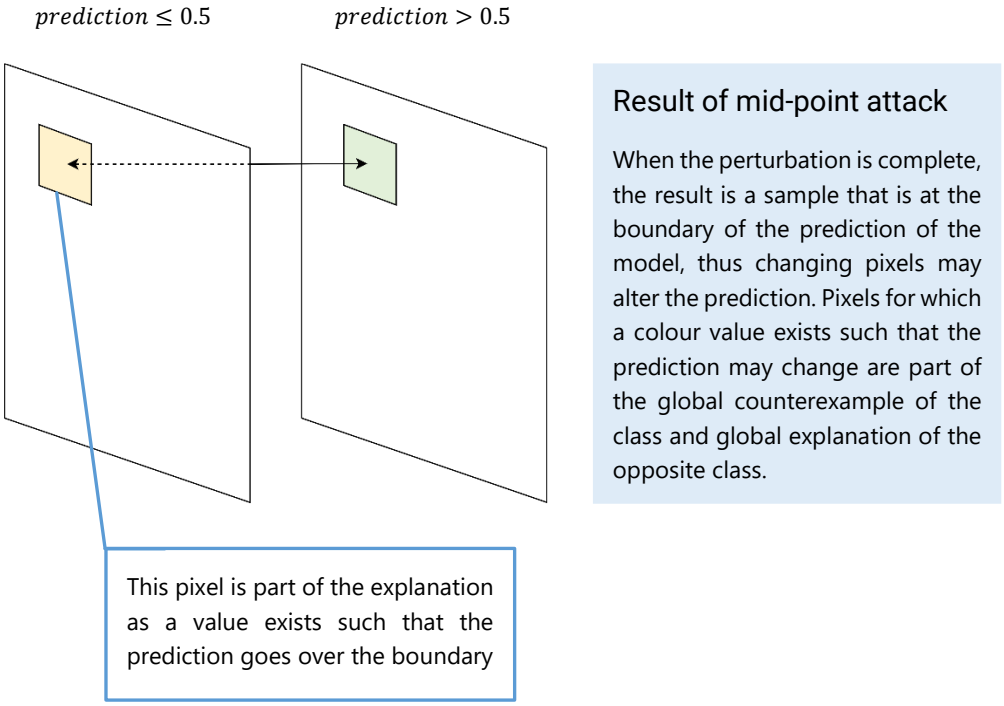


Figure 23 - Result of mid-point attack

Supposing that the model predicts a class from the available classes $\{\pi, \rho\}$, where class π relates to posterior $[0\ 0.5]$ and class ρ to posterior $(0.5, 1]$, we have two possible cases as result from this procedure. When prediction for the original sample is class ρ; an adversarial attack resulting in a prediction of 0.5 is a global counterexample to class

ρ. In the second case, pushing a sample from an initial prediction in class π to 0.5 will not make it switch class; it is however a very fine approximation of a global counterexample for class π as any slight modification to the result can make it switch. Because we are interested in proposing the result as a visual explanation to the user, we assume that very precise approximations, where imprecisions are not visible with the naked eye, are still valid. We might solve this issue by pushing samples to a prediction of $0.5 + \epsilon, \epsilon > 0$, but we expect other forms of inaccuracy, mostly due to the inability to exactly reach the target value in a limited number of perturbation steps. We will hence always obtain approximations, albeit accurate, of counterexamples.

We have thus found a way to obtain a very precise approximation of a global explanation of the opposite class which, due to the efficient nature of adversarial attacks, is also local to the original sample. We argue that analysing the differences between the original sample and its local-global counterexample is indeed a way to extract information about the model's decision and therefore it is an explainability technique.

It was proved that adversarial attacks can be used to generate explanations, in the section regarding *Explanation methodologies*, we will show how we can implement this idea practically, and how this concept will be our main link between our explanation techniques and more standard methodologies.

4.5 One-step attacks

Claims could be made against our procedures, concerning how a walk in feature space can bring us to describe information that is not relevant to the input image but that is relevant only to the already perturbed sample. We argue that our methodologies only aim at providing more accurate perturbations in terms of target values and efficiency, and that their relevance with respect to the original sample is left intact, even after the many perturbation iterations.

The solution to these arguments is developed in the form of one-step attacks, which follow the exact same procedures as seen in the algorithms, where the number of iterations is instead set to one. This means that all the gradient information that was used to perform the perturbation is relevant to the specific original image, and not to its perturbed versions.

The effectiveness of the many-iterations attacks stems from their intrinsic auto-tuning properties; as the latent space is not a flat surface, minor direction adjustments along the perturbation path allow the procedure to find a valid perturbed point which is closer to the original one. We also argue that, if the latent space is smooth enough, and this is expected to be true for our models, the direction changes experienced during the many-iterations attack are minimal. This smoothness is a consequence of the small amount of overfitting experienced in our models and is also due to other preventive techniques such as batch normalization that were applied during training. Therefore, a one-step attack of the same intensity as the sum of the intensities of all iterations of a many-iterations attack will give very similar results. Moreover, in case we

want an attack that fools the model into giving a specific output value, such is the case in mid-point attacks, a many-iterations attack will allow to find this perturbation in a fixed amount of steps. Instead, for a one-step attack, we would need to search for the correct step value in order to find the precise perturbation amount.

In the experiments, we will demonstrate the correlation between the one-step attack and the many-step attack and show that the direction changes that happen throughout the iterations of the many-iterations attack are indeed quite limited. This will be used to prove the validity of the many-step attack methodology in providing relevant information for the original input sample.

The techniques described up to this point make use of adversarial attacks to generate saliency maps that tell us which are the most relevant feature for the model. This analysis, however, has always been performed on the input, leaving untouched the latent space and subsequent perception visualizations. In this context, as described in section *Perception visualization*, the resulting reconstructions serve the sole purpose of mirror in validating the results of the perturbation techniques. In fact, we can use the reconstructions coming from perception visualization to verify whether the change that was induced by the perturbation also reflects in the perception visualization and, if that is the case, we can justify the results of the adversarial perturbation. In the next section, we will see a different technique to utilize the perception visualization also as a stand-alone explainability technique.

4.6 Perturbation recycling

What we aim to do with *perturbation recycling* is to develop a new saliency map stemming directly from the reconstructions generated by perception visualization. Therefore, we want to consider the inner perception the network has of the input it receives, and not the external input in its base form.

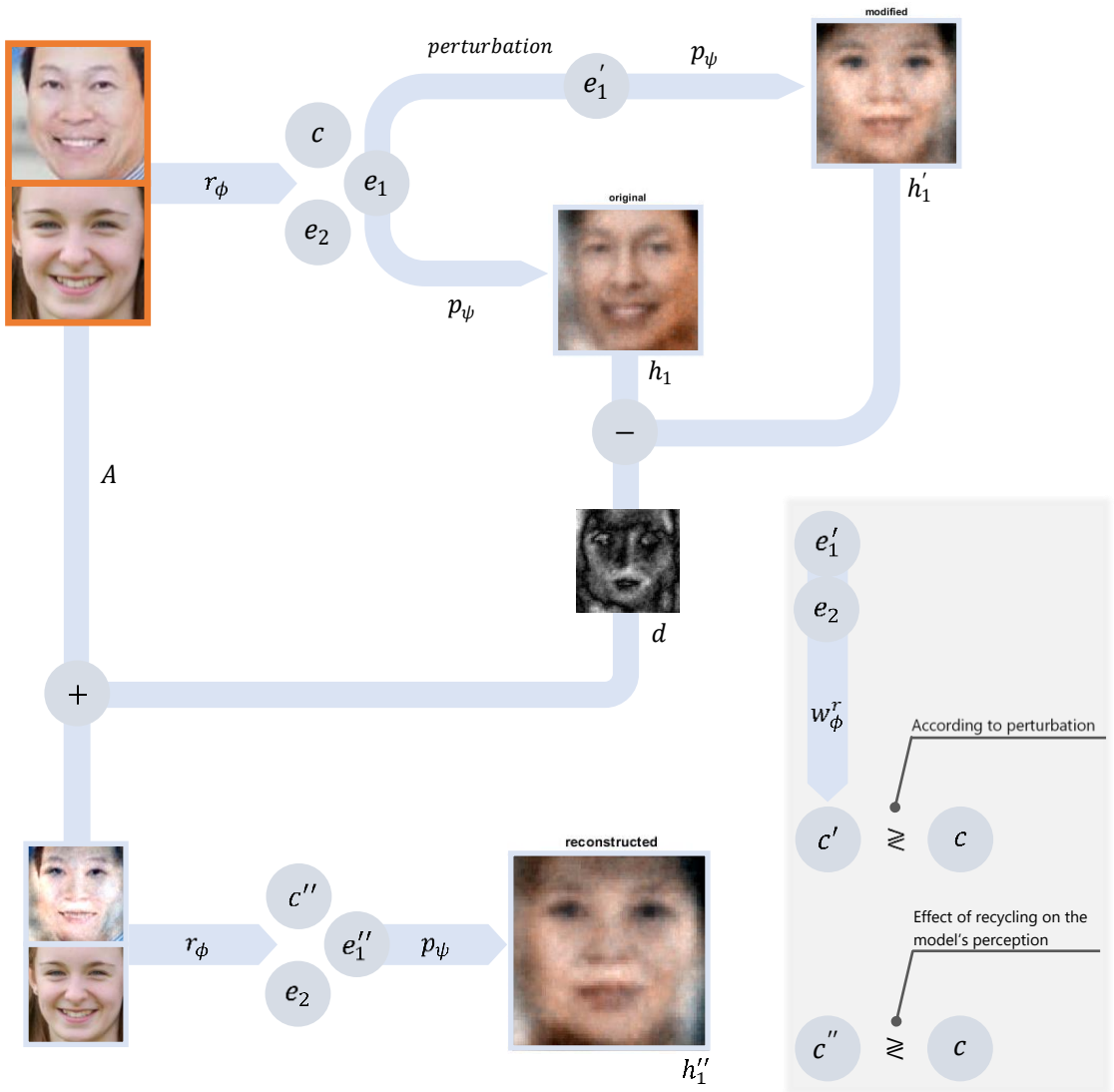


Figure 24 - Procedure for perturbation recycling

In *Figure 24*, we show the schematics of the perturbation recycling procedure, which is also detailed in *Algorithm 4*. We remind the reader that:

$$r_\phi(A,B) = \left\{w_\phi^r\left(w_\phi^e(A,B)\right), w_\phi^e(A,B)\right\} \qquad E\ 20$$

As seen in the previous algorithms, we start from input images A, B which, plugged in our model, will give $r_\phi(A,B) = (c, e_1, e_2)$ (line 2) which consists in a prediction c and perception visualizations $h_1 = p_\psi(e_1), h_2 = p_\psi(e_2)$ (line 3) for the two inputs respectively (h_2 is however not used in the procedures). The result after the perturbation is some perturbation of A, namely $A + \epsilon$ (line 4), such that $r_\phi(A+\epsilon, B) = (c', e_1', e_2')$ where c' is closer than the original prediction c to a specified target t (line 5). We also obtain the perception visualization for the perturbed sample $h_1' = p_\psi(e_1')$ (line 6) and now compare it to its original counterpart, obtaining the image $d = h_1' - h_1$ (line 7). This image d represents the salient features that were changed throughout the perturbation, akin to the input perturbation ϵ, but in the space of the perception visualizations. The idea is that the difference in output is generated by the network

itself, therefore it is much closer to its perceived difference than the difference that results in the input after having performed gradient descent.

To validate this process, and to prove that the changes we see are actually relevant for the network, we examine what happens when the image difference d, which represents the perceived perturbation, is plugged in the input as a perturbation. To do so, we construct the input $A + d$, and feed to the network the pair $(A + d, B)$, obtaining $r_\phi(A + d, B) = (c'', e_1'', e_2'')$ (line 8) and therefore the perception visualization $h_1'' = p_\psi(e_1'')$ (line 9).

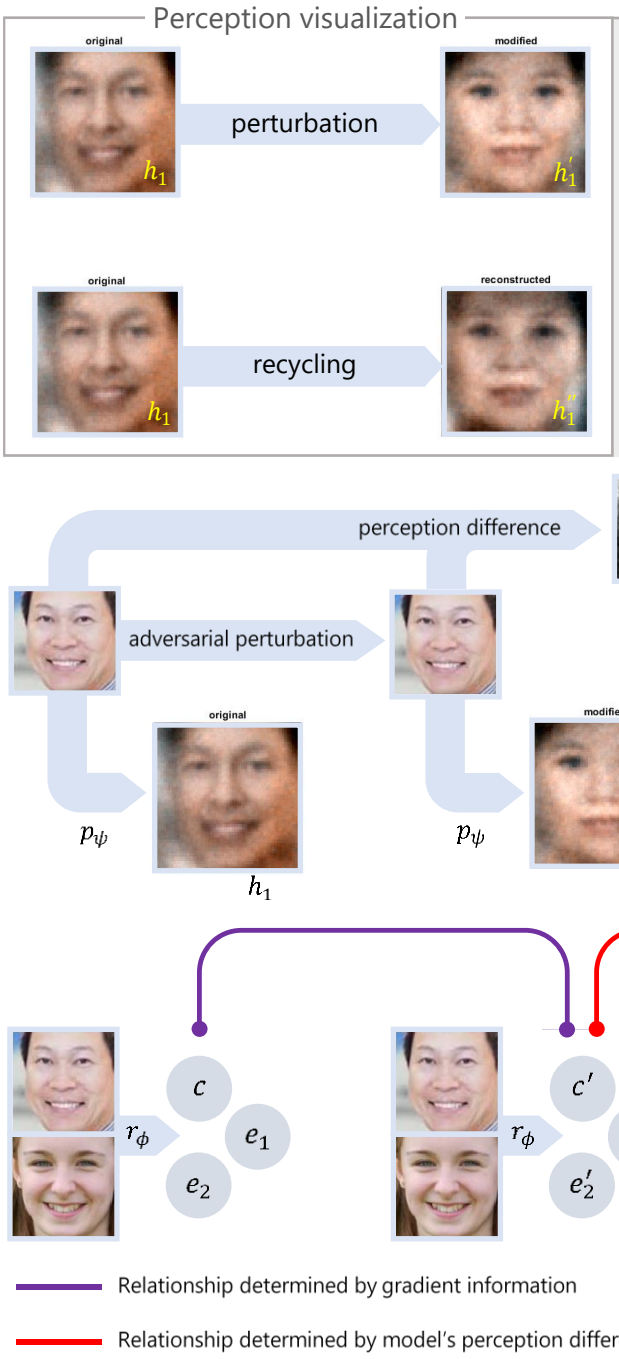
```
1  set input images A, B

2  c, e_1, e_2 = r_φ(A, B)
3  h_1 ← p_ψ(e_1)                    // reconstruction from original
4  A + ε ← perturb(A)                // run one of the perturbation
                                     // algorithms to obtain the perturbed
                                     // sample
5  c', e_1', e_2 = r_φ(A + ε, B)
6  h_1' ← p_ψ(e_1')                  // reconstruction of modified input
7  d ← h_1' − h_1                    // reconstruction difference

8  c'', e_1'', e_2'' = r_φ(A + d, B) // run model again on modified input
9  h_1'' ← p_ψ(e_1'')                // obtain the new perception visualiz.
```

Algorithm 4 - Perturbation recycling

We theorize (*Figure 24*, *Figure 25*) that if the effect of this perturbation d on the reconstruction p_ψ and on the prediction c has an equal or similar effect as seen when the network is given the pair $(A + \epsilon, B)$, then it means that d correctly captures what the network perceives as perturbation and how it interprets it. Indeed, if we perform a change in the input that reflects a change found in the perception visualization, and this modification then results in a perception visualization that is similar in features to the perception visualization obtained with perturbation methods, we can assume that this technique properly capture how the network perceives the input space.



In the top row, the differences in the two perception visualizations are due to the perturbation which is obtained from information on the gradient of the output.

The recycled perception visualization (bottom right) is instead the result of a perturbation obtained from the differences in the perception visualizations that we see in the top row.

A relationship between the perception visualization of the perturbed image (top right) and of the perception visualization of the recycled perturbation (bottom right) can tell us whether the differences in perception visualization between the original image and the perturbed one have semantic meaning for the network.

A correlation between the outputs of the network when given the perturbed sample and the recycled sample can instead tell us if perturbation recycling also drove the network in the same direction as the perturbation, thus proving that the perturbation recycling changed the visual, internal representation the network has of the input, but also that there is a direct relationship between the differences in reconstruction and the differences in perceived age.

Figure 25 - Reconstruction of recycled perturbations

4.7 Architecture

As shown before, the architecture that was used to test our work is a variant of a Siamese network where an additional decoder is attached after the embedding branches. In this section, we will explore the network structure in more depth and show which are the implications of the design choices.

The architecture is built around our methodologies, we are therefore in an *intrinsic* explainability context. Because the model is trained to simultaneously optimize the reconstruction loss of the decoder and the prediction performance, it is impossible to apply this technique to a generic pre-trained model without fine-tuning or in some cases full re-training of the network. Indeed, a model that was pre-trained to solve only the comparison problem might discard, in its embeddings, information needed to obtain good reconstructions. However, we argue that this issue does not present any drawback to our research target because our aim is that of developing new models from scratch that have better explainability potential.

In detail, the network takes two images A_{ij} and B_{ij} as input, feeds them through one of two copied branches of a Siamese network, and finally concatenate the results of the branches (embeddings e_1, e_2) to then complete the ranking prediction using one or more fully connected layers. This standard Siamese data flow is then spiked by attaching the decoder network to the output of the Siamese branches, obtaining two reconstructed images, by running the decoder once for each embedding (*Figure 19*). We can build a model of this type from scratch and train it to optimize for a joint loss that considers both the reconstruction and the prediction. In this way we can obtain good reconstruction quality and at the same time a network able to solve the comparison task with high performance.

To do so, we must start with a dataset of sample images, and an *oracle* that, for each pair of images A, B, gives the target ranking in the interval $[0,1]$. The ranking network will give a *prediction* result, while the decoder will give a reconstruction image for each embedding it receives as input. During training, we want to optimize:

$$\mathcal{L}_m = k * MSE(target, prediction) + MSE\left(A, p_\psi\left(w_\phi^e(A)\right)\right) + MSE\left(B, p_\psi\left(w_\phi^e(B)\right)\right) \qquad E\ 27$$

Where the MSE for images is computed by taking the sum of squares of the pixel-wise difference between the two images.

To train the network, we randomly pick two items from the dataset and create a sample; the pair of images is then fed through the network and each of the generated embeddings is sent to the reconstruction net. We can then find the terms of our loss from the outputs of the comparison and the reconstruction, and finally compute the loss function as stated above. It is important to note that this training method places a bias in the way the CNN produces the embedding, as it is driven to preserve the information needed to reconstruct the image, and not only the information required to perform the comparison. This clearly exposes the trade-off between explainability and performance of deep learning models, however, we argue that this bias is

irrelevant, as long as the ranking performance obtained by these networks remains very similar to that of other non-interpretable models.

4.8 Explanation methodologies

In this section, we will describe the implementation details for the explainability techniques that were developed in this thesis.

4.8.1 Saliency map for adversarial attack

This type of explanation is directly obtained by computing the difference between the input sample to be explained and the same input after the perturbation process is complete. The result is a grayscale image where brighter pixels represent regions that were most affected by the adversarial attack, thus representing a region that is more relevant for the network. To obtain the saliency map, we compute the pixel-wise absolute difference between the original input A and the modified input A' such that:

$$\hat{E}(x,y) = ||A(x,y) - A'(x,y)||_2 \qquad E\ 28$$

Pixels are taken as three-dimensional vectors with dimensions describing their three colour channels R, G, and B; the distance will hence be computed in this three-dimensional space. To obtain a displayable image, the result is then rescaled to fit in the range $[0,1]$; we finally obtain:

$$E(x,y) = \frac{\hat{E}_{x,y} - \min(\hat{E})}{\max(\hat{E}) - \min(\hat{E})} \qquad E\ 29$$

Saliency map for adversarial attack

Example of a saliency map for a face, where we can barely make out the facial features of the subject.

Figure 26 - Example saliency map

In some cases, it is useful to superimpose the generated map to the original input, in order to match and detect which features where taken into consideration in the image. We perform this superimposition by placing the grayscale version of the original input into one channel $C1$ of the explanation; the saliency map is finally placed in a second colour channel $C2$ of the output, obtaining a bi-colour saliency map. The remaining colour channel is left empty (zero valued).

$$E^s(x,y,C1) = E(x,y)$$
$$E^s(x,y,C2) = \frac{A(x,y,1) + A(x,y,2) + A(x,y,3)}{3} \qquad E\ 30$$
$$E^s(x,y,C3) = 0$$

To aid colour-blind readers, some of these explanations are provided in two flavours: one where the saliency map is in green, superimposed over a red original input, and a second type where the saliency map is in red, superimposed over a blue original input.

Different flavours of superimposition

The two explanations, one green on red and the other red on blue, are used to aid colour-blind people in discerning the saliency map from the original input image over which it is superimposed.

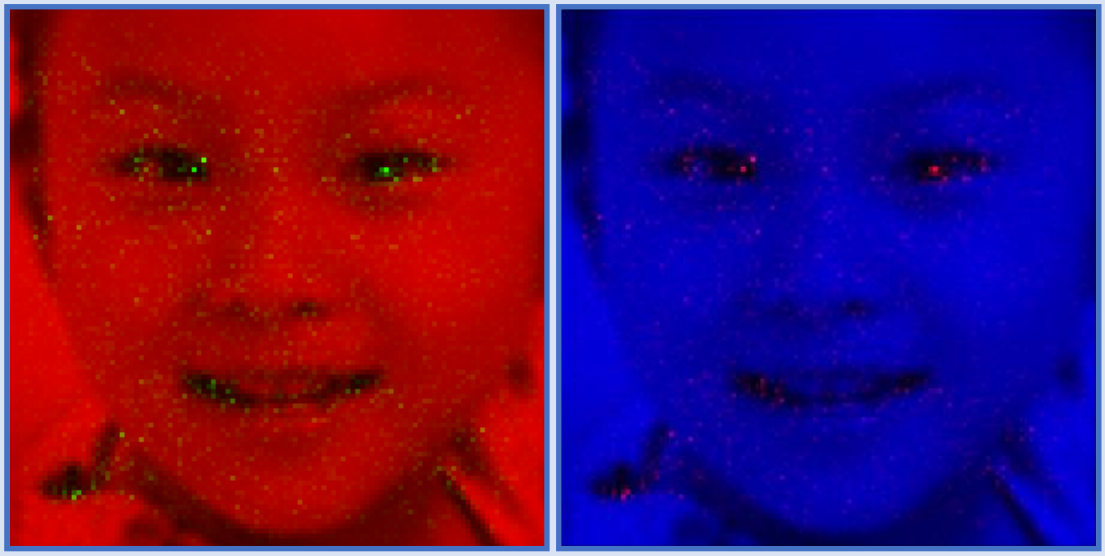


Figure 27 – Superimposition examples

Another variant of this kind of saliency maps can be generated by segmenting the original image into super-pixels using the SLIC [22] algorithm and then colouring the resulting regions based on the maximum value of the original perturbation in that specific region. This helps in understanding better the explanation as the image results less noisy, and the granularity of the explanation is increased from the pixel to the super-pixel mass, which is more reasonably sized for a human to see and discern against other regions. The generated tessellation is coloured following the jet colourmap and is then superimposed to a grayscale version of the original input image, similarly to how it was done in the previous techniques.

Tessellation of heatmap

Image is tessellated into super-pixels, the maximum perturbation amount for all pixels in each region is considered to colour it according to the jet colour map.

The coloured regions are then superimposed over a grey-scale version of the input (above), thus generating the final map (below).

Figure 28 - Tessellation example

Application of this techniques to mid-point adversarial attacks allows to generate proper explanations for why the network deemed one sample to be greater/lesser than the other. We will see, however, how we can also use other type of attacks to achieve the same result of explaining the model's decision.

4.8.2 Perception visualization

The perception visualizations $p_\psi\left(w_\phi^e(A)\right)$ and $p_\psi\left(w_\phi^e(A_m)\right)$ of the original image and of the perturbed image can be used as an explanation. In this case, the explanation is offered as a pair of images, one displaying the perception visualization before the perturbation and one after the process is complete. The explanation is intrinsic in the differences between the images and it is not given as a simple saliency map. Information about what features were most relevant in the network's decision can be entailed from the differences in perception visualization before and after the perturbation. In this form, the perception visualization will be used to validate other saliency maps by comparing the effects of the attack on the perception visualization with the regions highlighted by the attack itself.

Perception visualization as explanation

The perception visualization of the input before and after the perturbation process can be visually inspected in order to find which features were relevant for the network's decision. In the example we see how the face's shape and peculiar visual features where modified during the perturbation.

Comparing these modifications to the salient features extracted by an adversarial attack can help validate it.

Figure 29 - Example of explanation through perception visualization

4.8.3 Saliency map for perception visualization

In order to bridge the gap between saliency maps and explanation through visual inspection of the perception visualizations, we also propose a way to further analyse the latter through a saliency map. The objective of this saliency map is to highlight the regions in the perception visualization that were most impacted by computing the pixel-wise difference between the perception visualization before and after the perturbation, namely P and P'. The result is obtained by computing:

$$\hat{E}(x,y) = ||P(x,y) - P'(x,y)||_2$$
$$E(x,y) = \frac{\hat{E}(x,y) - \min(\hat{E})}{\max(\hat{E}) - \min(\hat{E})} \quad \text{E 31}$$

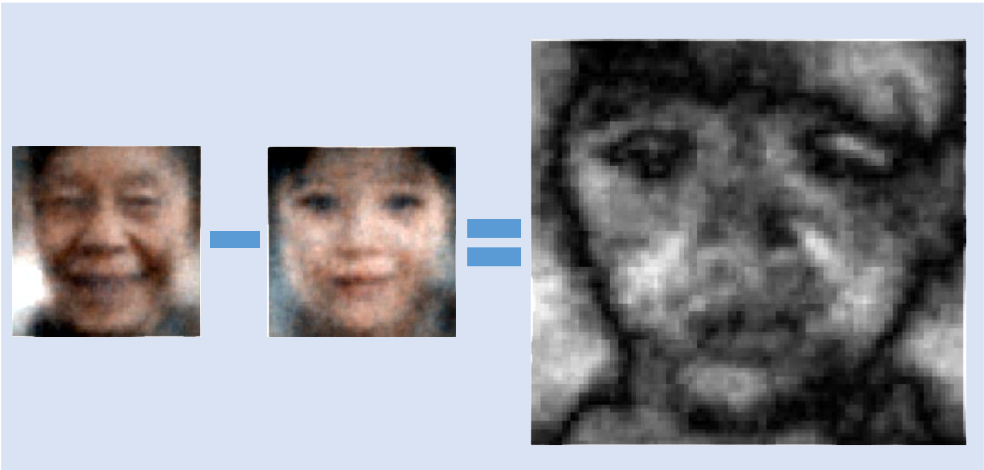


Figure 30 - Example of saliency map for perception visualization

5 Experiments and discussion

In this section, we will explain in detail how our work was tested and what methodologies were used both from the conceptual point of view and the practical one.

5.1 General framework

The experiments that will be described loosely follow the evolution of our studies and tackle progressively more challenging problems. Some tests are to be taken as more explorative, and their results might be less significant.

All experiments have been carried out using MATLAB and its deep learning toolbox. The use of external libraries was minimal; other custom functions have mostly been developed from scratch. All other references to datasets and implementation details will be described for the single experiments. Training was performed on a single Nvidia GeForce GTX1080 GPU using the parallel computation toolbox from MATLAB.

5.2 Experiments on MNIST dataset

The focus of this experiment was to understand whether the proposed ideas regarding ranking problems was feasible in practice. The objective is to train a network based on our Siamese-decoder architecture able to rank handwritten digits in the MNIST dataset, where the ordering is given by the natural greater than ($>$) relationship.

This experiment makes only use of derivates of *Algorithm 1*, which consists of perturbing the latent space, never acting on the input image. This gives a perception visualization explanation but does not generate an adversarial attack; the results are therefore indicative as they will not be grounded to the classical explainability techniques. However, as the problem is very simple, the results can still give us great insight on the perception of these kind of networks.

We must note that handwritten digits, as the ones seen in the MNIST dataset, are not natural images, as they are composed of artificial features. This makes the task harder for our explainability technique on a CNN; regardless, we will use these results only to establish whether our line of work is feasible and to lay the foundations for later experiments.

Due to the investigational purposes of this first set of experiments, visual results will be shown only for the latest architecture. For sake of completeness, we will explain in detail the functioning and architecture for all the conducted trials, but as results were poor in the very first tests, for those experiments we will only report the model's performances in text.

5.2.1 Dataset

The initial dataset used in these experiments is the standard MNIST dataset, however, each image was converted from raw binary data (its form in the original dataset) to jpg for easier use with our models.

Each image in the dataset is a grayscale, 28×28 pixel image with 8 bit depth. The dataset was reorganized in ten folders, each containing all samples relative to one specific digit.

Because the network always takes two samples as input and a value that refers to their ranking, the dataset that is fed to the network is composed of pairs of images randomly taken from any of the ten folders. The ranking value for two samples A and B is computed as the normalized difference between the values V_A and V_B of the handwritten digits relative to the images, giving a number in the range $[0,1]$. Because the possible values are in the range $[0,9]$ and we must account for samples where the second value is both greater and lower than the first, the resulting target value will be given by:

$$t(A,B) = \frac{V_A - V_B}{18} + \frac{1}{2} \qquad E\ 32$$

These values can be generated on the fly during network training and testing but, for performance reasons, a temporary dataset was precomputed containing a large number of these samples of pairs of images and relative ranking. This allows to test with different architectures more quickly in early experiments but may induce bias in the training process as the same dataset is used on multiple model iterations. Final models are to be validated on newly baked sets.

5.2.2 Autoencoder to Siamese network

The first set of experiments aim at exploring different approaches to the problem in terms of practical architectures and training methods. The very first methodology that was tested consists of training an autoencoder to reconstruct images in the dataset (handwritten digits in our case). This encoder can later be reused as the Siamese part of the main architecture; the pre-trained decoder can then be used to construct the perception visualizations. The remaining portion of the network is the tail (fully connected) section of the Siamese network, which can be trained after having frozen the parameters of the Siamese branches coming from the pre-trained encoder. Freezing the parameters of the encoder means that the reconstruction capabilities of the network remain the same of those of the autoencoder, as the embedding network is the same. However, this results in very scarce ranking performance, as the network was trained on a different task and might suppress information needed to also solve the comparison.

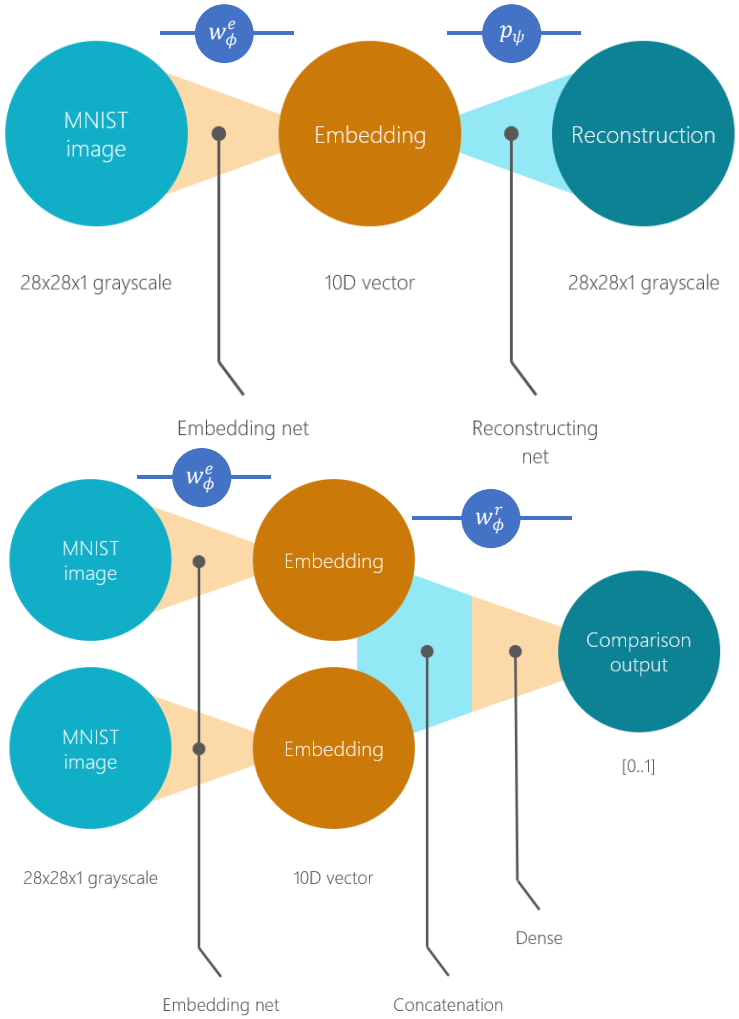


Figure 31 - The encoder is used to compute embeddings for the Siamese network

Autoencoder

An autoencoder is trained to embed and reconstruct images from the dataset; the encoder and decoder parts of the network can be used independently.

Encoding is done with a ten-dimensional vector, but this parameter can be changed.

Siamese network

The same encoder is now used as both branches of the Siamese network and it computes the embeddings for the remaining portion of the network, which outputs the comparison.

The reconstruction for the perception visualization can be obtained again by using the decoder trained before.

In this way, we are forced to face a trade-off between model performance in solving the comparison task and ability of generating perception visualizations that are clear. This highlights the need for a *model-specific* explainability framework; as networks are not naturally prone to give unambiguous results, and they might even hide or discard information that may be useful for the user in understanding the network. We hence want to train our networks in giving correct results but simultaneously provide good explanations.

The first step in pushing the network to have good performance while being able to give good reconstructions consist in the regularization of the latent space during the autoencoder's training process. In our experiment, we slightly modified the autoencoder loss to take in account the reconstruction quality but also the norm of the latent embedding and forcing it to be close to one.

$$\mathcal{L}_{autoencoder} = MSE\left(A, p_\psi\left(w_\phi^e(A)\right)\right) + \left(1 - ||w_\phi^e(A)||_2\right)^2 \qquad E\ 33$$

This idea stems from the more general concept of variational autoencoders, which are the next logical steps in this direction. Variational autoencoders work by reconstructing

samples that are taken from a normal distribution whose parameters are defined by the embedding generated by the encoder. This allows for "automatic" regularization of the latent space when the embeddings are forced to be Gaussians of mean zero and variance one.

Both explicit regularization and variational autoencoder were tested in our experiments, but the results were not promising, as the models were never able to be more accurate than average ± 0.2 on the range $[0,1]$. It is clear that the methodologies used for latent space normalization are not enough to overcome the trade-off between performance in solving one of the two tasks.

Another different route that can be taken is fine-tuning the parameters of the Siamese branches after having copied them from the pre-trained autoencoder, thus allowing the network to optimize to also solve the comparison task. This is the polar opposite to the trade-off regarding the methods previously explained, as this technique will surely degrade the reconstruction performance. As we will see, however, this path has not been explored as a more radical solution was developed, able to greatly increase performance.

5.2.3 Autoencoder mirroring

Following a similar idea to that of [23], in this experiment we explore a different way to train an autoencoder to reconstruct the perceptions of the Siamese network. We do so by firstly training a Siamese network from scratch to solve the comparison task; finally, we train an autoencoder and force it to generate embeddings that are similar to those generated by the Siamese network. The autoencoder is however allowed to optimize its latent space to correctly reconstruct the input. In this setup, the Siamese network computes:

$$siamese: w_\phi^r\left(w_\phi^e(A,B)\right) \qquad \text{E 34}$$

And the autoencoder computes:

$$autoencoder: p_\psi\left(w_\phi^e(A)\right) \qquad \text{E 35}$$

To drive the encoder to have similar embeddings to the ones of the Siamese network, we modify the standard loss of the autoencoder to account for the mean square error between the embedding it generates and the embedding the Siamese network generates for the same sample.

$$\begin{gathered} e_{siamese} = w_\phi^e(A) \text{ for the Siamese network} \\ e_{encoder} = w_\phi^e(A) \text{ for the autoencoder} \\ \mathcal{L}_{autoencoder} = MSE\left(A, p_\psi\left(w_\phi^e(A)\right)\right) + k * MSE(e_{siamese}, e_{encoder}) \end{gathered} \qquad \text{E 36}$$

In this way, the Siamese network is trained at the best of its abilities in the comparison task, while the autoencoder parameters have more freedom to be tuned for the reconstruction task. If training is long enough, we can safely assume that the latent space of the autoencoder is somewhat similar to that of the Siamese network, and thus it can be used to explain it.

Implementation details

For this experiment, it was important that the embedding section of the autoencoder was replicated identically as to mimic the behaviour of the Siamese embedding branches. We allow the embedding size to be variable and call it $size_e$, but once chosen it will be the same for both autoencoder and Siamese network.

The architecture for the autoencoder and therefore for the Siamese branches (encoder portion) is as follows.

Layer	Type	Input size	Properties	
conv1	convolutional	$28 \times 28 \times 1$	filter size 5×5, 64 filters	ENCODER / SIAMESE
relu1	ReLU	$28 \times 28 \times 64$		
maxpool1	max pooling	$28 \times 28 \times 64$	size 2×2, stride 2	
conv2	convolutional	$14 \times 14 \times 64$	filter size 3×3, 128 filters	
relu2	ReLU	$14 \times 14 \times 128$		
maxpool2	max pooling	$28 \times 28 \times 128$	size 2×2, stride 2	
conv3	convolutional	$7 \times 7 \times 128$	filter size 3×3, 128 filters	
relu3	ReLU	$7 \times 7 \times 128$		
flatten	convolutional	$7 \times 7 \times 128$	filter size 7×7, $size_e$ filters	
tconv1	transposed conv.	$1 \times 1 \times size_e$	filter size 7×7, 128 filters	DECODER
trelu1	ReLU	$7 \times 7 \times 128$		
tconv2	transposed conv.	$7 \times 7 \times 128$	filter size 2×2, 128 filters	
trelu2	ReLU	$14 \times 14 \times 128$		
tconv3	transposed conv.	$14 \times 14 \times 128$	filter size 2×2, 1 filter	
output	clipped ReLU	$28 \times 28 \times 1$		

Table 1 - Autoencoder architecture (MNIST)

The remainder of the Siamese network is composed of a fully connected portion with one hidden layer and an output layer with one single neuron with sigmoid activation.

Despite the shortcomings of this architecture and its asymmetries, we are able to generate valid reconstructions through perturbation analysis. The procedure forces the network to generate different perception visualizations and comparison predictions than the ones it initially provides given a specific sample, while keeping the embedding as similar to the original as possible. We do so by following the gradient of the output with respect to the embedding at the end of the Siamese branches. After the embedding has been modified, the resulting feature vector can be fed through the decoder that will output an image representing the perception visualization of the perturbed embedding (*Algorithm 1*).

5.2.4 Simultaneous training

With this experiment, we flesh out the final, general form of the architecture that will then be tuned in later tests. We remove the need for an autoencoder and train a Siamese network to also generate reconstructions directly. The specifics of the architecture will change in subsequent experiments and more optimized layer configurations will be used, but from now, all the models will follow the high-level architecture explained in *Figure 19*. The specific architecture for this experiment is detailed below. Like in previous experiments, the embedding size is variable.

SIAMESE BRANCH ARCHITECTURE

Layer	Type	Input size	Properties
conv1	convolutional	$28 \times 28 \times 1$	filter size 3×3, 32 filters, stride 2
relu1	ReLU	$14 \times 14 \times 32$	
conv2	convolutional	$14 \times 14 \times 32$	filter size 3×3, 64 filters, stride 2
relu2	ReLU	$7 \times 7 \times 64$	
flatten	convolutional	$7 \times 7 \times 64$	filter size 7×7, $size_e$ filters

Table 2 - Siamese branch architecture (MNIST)

SIAMESE TAIL SECTION

Layer	Type	Input size	Properties
dense1	fully connected	$size_e$	4096 neurons, sigmoid activation
output	fully connected	4096	1 neuron, sigmoid activation

Table 3 - Siamese tail architecture (MNIST)

DECODER

Layer	Type	Input size	Properties
tconv1	transposed conv.	$1 \times 1 \times size_e$	filter size 7×7, 128 filters
relu1	ReLU	$7 \times 7 \times 128$	
conv2	transposed conv.	$7 \times 7 \times 128$	filter size 2×2, 128 filters, stride 2
relu2	ReLU	$14 \times 14 \times 128$	
tconv3	transposed conv.	$14 \times 14 \times 128$	filter size 2×2, 1 filter, stride 2
output	clipped ReLU	$28 \times 28 \times 1$	

Table 4 - Decoder architecture (MNIST)

Results

Results are promising and show how the network has developed a feature-rich latent space that is easily visualizable through our techniques. In practice, we would like to see what features change in the perception visualizations when we force the network to give a different result than the original one, and whether these changes reflect how humans perceive the order relationship ">" between digits.

Visual results for this experiment consist in a series of images showing the perception visualization taken at different points during the perturbation process. Example results will be accompanied by the two original input images; we remind the reader that only the embedding of the first of the two input images is subject to the perturbation, while the second is kept fixed.

Decreasing the value of a digit

In this example, we can see how forcing the network to give a lower prediction drives the reconstruction to resemble lower digits. Starting with the digit 8, the model morphs it in something similar to a 6, then a 5 and finally a 3.

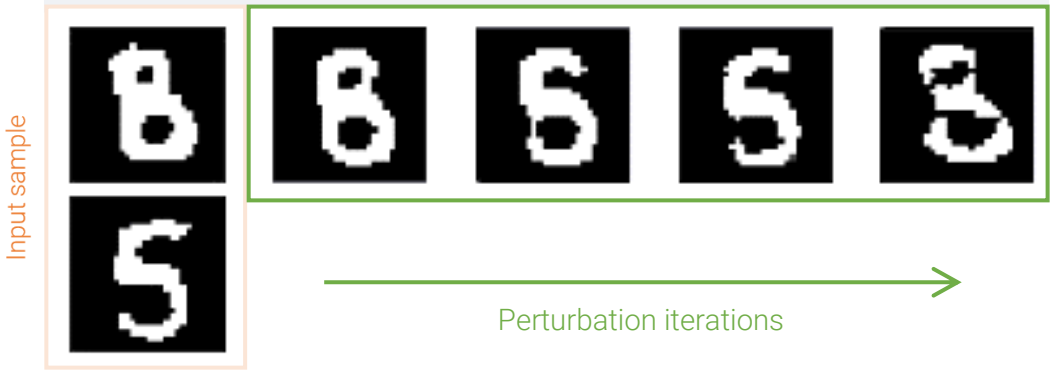


Figure 32 - Modifying a digit value through perturbation

In *Figure 32* we see how the reconstruction of the digit eight was perturbed; this result was achieved by forcing the network to give a lower prediction. The perturbation drives the network to represent a lower valued digit, because it is trained to give an output relating to $A \overset{?}{\geqslant} B$. It is important to note that the digit five as seen in the input is not related to the same digits reached during the perturbation process.

This result is however somewhat deceiving, as it shows a fortuitous progression ranging several meaningful digits along the perturbation path. What we see here is not a behaviour to be expected in general, as the perturbation process always forces the sharpest change; this means that if possible, the perturbation will directly go towards the target which could be represented by a digit nine or one depending on the desired direction but ultimately could be an incomprehensible representation of the maximum or minimum digit as the network perceives it. The process could also halt in a local minimum and the result might stabilize before reaching the boundary of the digit range.

The example is otherwise instructional in telling us about the latent space of the model, and it shows how this sampling process seen in *Figure 32* can be used to peek inside the network inner workings. Indeed, this is evidence that, in the latent space of our specific model, the path of maximum steepness between the original eight digit and the resulting three intersects regions that also represent the digits six and five.

In the following examples, we will see how the perturbation affects the output both in terms of the prediction and in visual factors.

Figure 33 - Progression of transformation though perturbation

In *Figure 33* we see the progression of the transformation of a digit by means of our perturbation techniques. We also see the model changing prediction and how this prediction reflects the perturbation output in the samples taken. The number above each sample represents $(1-\mathcal{L})$, where $\mathcal{L}$ represents the loss computed during the perturbation process as described in the algorithms. Because we want to obtain smaller digits for the perturbed sample, the prediction will be forced to go towards lower values; thus, our perturbation loss will be the difference between the current prediction and zero, which is the best possible prediction when looking for low digit results.

It is possible to plot the target outcome of the comparisons with a digit five and compare them with our results. We obtain this plot by using the standard normalizing formula (*E 32*) $\frac{V_A - V_B}{18} + \frac{1}{2}$ and plugging in 5 into V_B. The target for a pair of images where the second image is a digit five will hence be computed as $\frac{V_A - 5}{18} + \frac{1}{2}$. We plot for all possible values of v_A and obviously obtain a line.

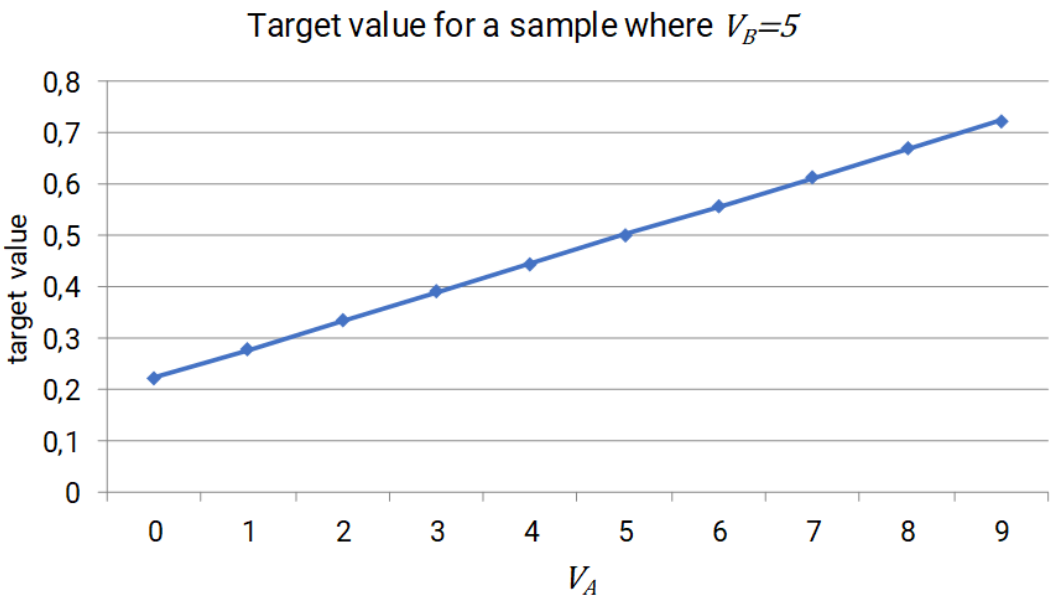


Figure 34 - Target value when comparing a digit to a digit 5

From *Figure 33* and *Figure 34* We can induce that the initial prediction $1 - 0.45 = 0.55$ is in line with a comparison between a six and a five on a range from zero to nine. The last prediction is instead 0.17; while we would expect a similar prediction to be out of range, and possibly represent an unintelligible image, the model produced something that can be assumed to be a three. This is due to the artificial nature of the generated image, as comparison with a "natural" three would most likely result in a prediction close to 0.39 as depicted in the chart. We can however use this to demonstrate that the latent space of the model represents the training set domain also when pushed to its boundaries.

However, this highlights a drawback of either our methodologies or our architecture: while the results are informative and always generate digits that shift towards the desired direction (larger digits when going towards higher prediction and vice versa), we see a great loss of prediction accuracy during the perturbation process. This means that we may not use, at least with the current framework, the prediction result after the perturbation to obtain information about what specific value the result represents. We can still obtain information about how sensitive the model is to a specific sample by analysing how many iterations it takes for the prediction to significantly change, and by relating this alteration to the feature change in the resulting reconstructions.

In the literature, analysis of the latent space has already been performed to see whether image interpolation was possible, for example when dealing with generative networks. Previous studies explored whether intermediate points between two latent vectors could be assimilated as interpolation between the resulting images. A possible critic of our work is that a simple interpolation or extrapolation between two samples may give the same result as those of our work. We want to dissolve these arguments both theoretically and with example results. Our algorithms, in fact, do not rely on latent space vector difference to determine the direction of movement, but they are only

driven by the gradient information. This means that the exploration is much richer than linear interpolation.

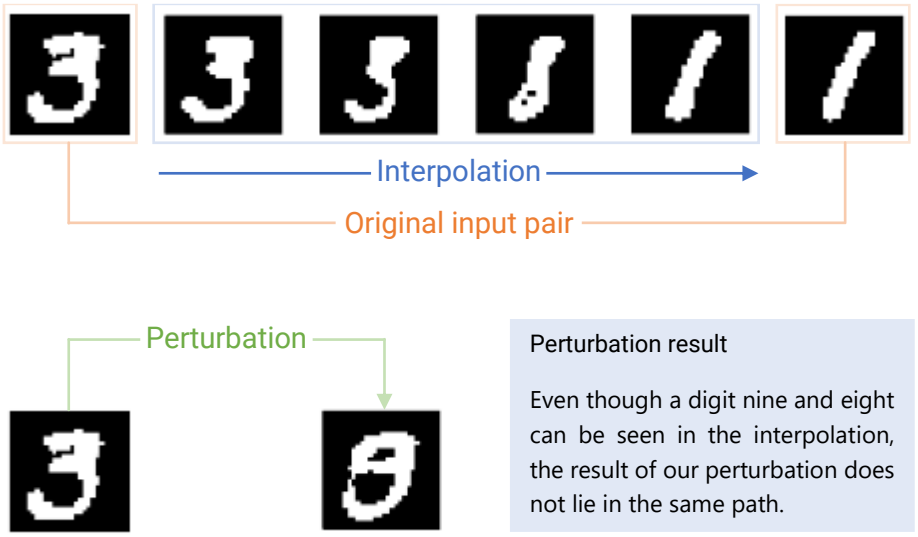


Figure 35 - Comparison between interpolation and perception visualization

In *Figure 35*, we show a possible latent space interpolation of a pair of samples and show how the interpolation is akin to the standard image interpolation seen for other well-posed latent spaces. The result of the perturbation following our methodologies gives instead a result that does not show a combination of the features of the input. This both shows that our methodologies can provide some novel knowledge outside of that of interpolation and proves that the latent space that was generated with our model is well suited for image interpolation. In other words, this means that the latent space is sufficiently smooth, and that we can expect continuous changes in the output reconstruction when performing continuous perturbation in the latent space.

5.3 Experiments on UTKFace dataset

In this series of experiments, we build upon the findings of the previous experiments, but tackle a much more complex task, enabled by architectural improvements of our model. The aim is to apply the techniques developed using the MNIST database to pairs of human faces, then teach the network to recognize the order relationship "$older\ than$". We will use similar methods to generate samples and target values to then feed to the network; the main difference in the dataset is however that faces represent a much more natural set of images than grayscale, handwritten digits. The focus is indeed to see how a CNN tackles a task that resides in its "area of expertise".

5.3.1 Dataset

The UTKFace dataset [24] is a large-scale face dataset with long age span (people from 0 to 116 years old) and consists of over 20 thousand face images. The pictures cover a large variation of pose, facial expression, ethnicity, illumination, and more. Images are taken in the wild and are then aligned and cropped to contain one single face per sample. Images are labelled by age, gender, and ethnicity, although our experiments will only make use of the age attribute.

Figure 36 - Sample images from the UTKFace dataset

Similarly to what was done for the MNIST dataset, a training set composed of pre-processed pairs of images was compiled by randomly selecting couples of images from the original dataset. Finally, a target value was assigned to each pair by taking the normalized difference, this time from the range $[0,116]$.

$$t(A,B) = \frac{V_A - V_B}{116 * 2} + \frac{1}{2} \qquad E\ 37$$

5.3.2 Architecture improvements

The architecture was further improved for these experiments to resolve asymmetries between the encoder and the decoder's architectures, and to solve problems relative to overlapping filters (due to wrongly sized kernels and stride values). Finally, with this architecture, we make use of batch normalization ($batchnorm$) layers between each $ReLU$ layer and the subsequent $conv2D$ layer. This allows for much better image quality after reconstruction and avoids checkerboard artifacts. However, due to hardware limitations, models are still relatively simple and result in blurry images. Sharper reconstructions might be achieved by larger models as shown by the current state of the art for autoencoders. Regardless, the main facial features are still clearly visible even in reconstructions coming from our models.

SIAMESE BRANCH ARCHITECTURE

Layer	Type	Input size	Properties
conv1	convolutional	$112 \times 112 \times 3$	filter size 4×4, 32 filters, stride 2
relu1	ReLU	$56 \times 56 \times 32$	
batchnorm1	batch norm.		
conv2	convolutional	$56 \times 56 \times 32$	filter size 4×4, 64 filters, stride 2
relu2	ReLU	$28 \times 28 \times 64$	
batchnorm2	batch norm.		
conv3	convolutional	$28 \times 28 \times 64$	filter size 4×4, 128 filters, stride 2
relu3	ReLU	$14 \times 14 \times 128$	
batchnorm3	batch norm.		
conv4	convolutional	$14 \times 14 \times 128$	filter size 4×4, 256 filters, stride 2
relu4	ReLU	$7 \times 7 \times 256$	
batchnorm4	batch norm.		
flatten	convolutional	$7 \times 7 \times 256$	filter size 7×7, $size_e$ filters

Table 5 - Siamese branch architecture (UTKFace)

SIAMESE TAIL SECTION

Layer	Type	Input size	Properties
dense1	fully connected	$size_e$	$size_e * 2$ neurons, sigmoid activation
output	fully connected	$size_e * 2$	1 neuron, sigmoid activation

Table 6 - Siamese tail architecture (UTKFace)

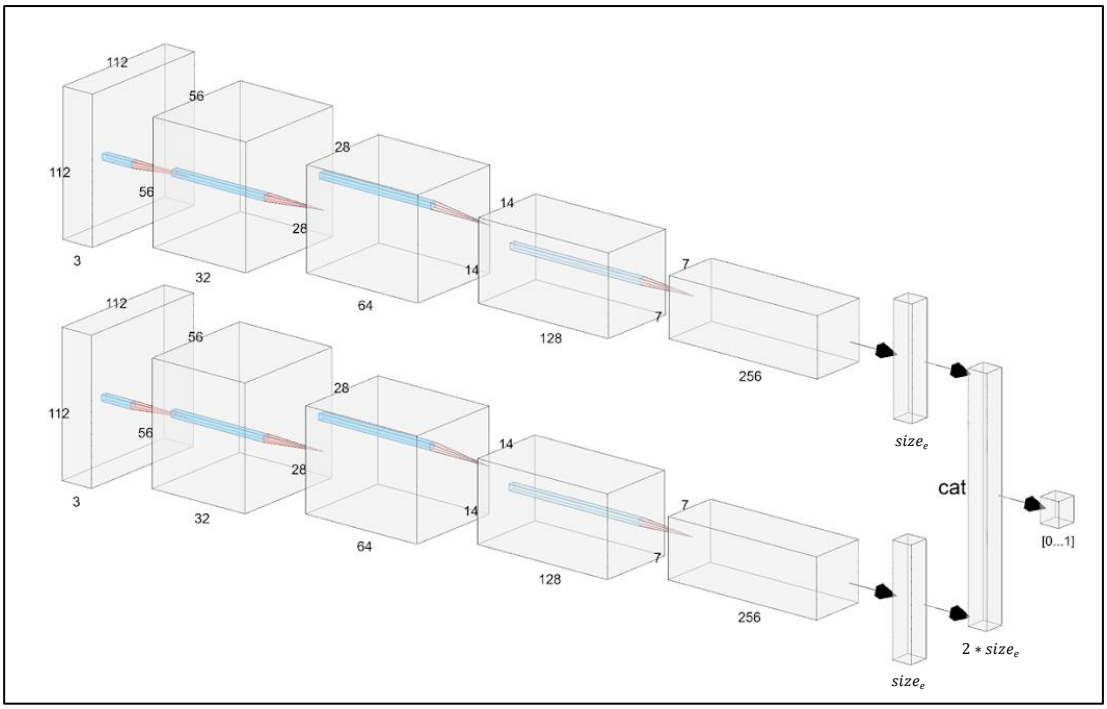


Figure 37 - Architecture of the Siamese and joint sections

DECODER

Layer	Type	Input size	Properties
tconv1	transp. conv.	$1 \times 1 \times size_e$	filter size 7×7, 256 filters
trelu1	ReLU	$7 \times 7 \times 256$	
tbatchnorm1	batch norm.		
tconv2	transp. conv.	$7 \times 7 \times 256$	filter size 4×4, 128 filters, stride 2
trelu2	ReLU	$14 \times 14 \times 128$	
tbatchnorm2	batch norm.		
tconv3	transp. conv.	$14 \times 14 \times 128$	filter size 4×4, 64 filters, stride 2
trelu3	ReLU	$28 \times 28 \times 64$	
tbatchnorm3	batch norm.		
tconv4	transp. conv.	$28 \times 28 \times 64$	filter size 4×4, 32 filters, stride 2
trelu4	ReLU	$56 \times 56 \times 32$	
tbatchnorm4	batch norm.		
tconv5	transp. conv.	$56 \times 56 \times 32$	filter size 4×4, 3 filters, stride 2
output	clipped ReLU	$112 \times 112 \times 3$	

Table 7 - Decoder architecture (UTKFace)

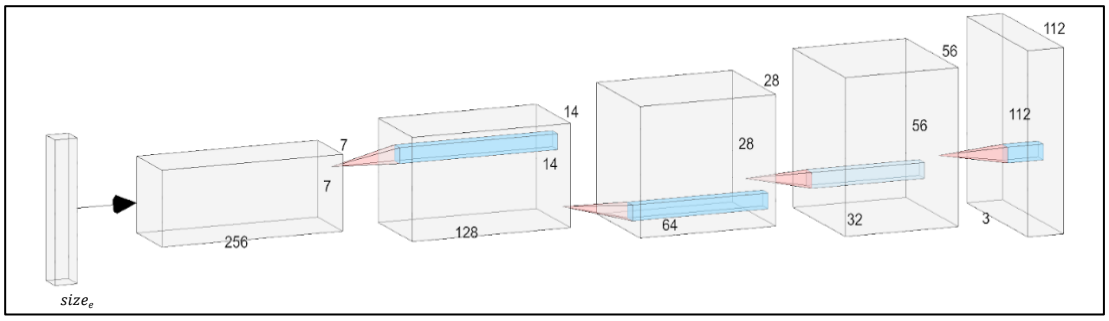


Figure 38 - Visualization of the architecture of the decoder

5.3.3 Face perturbation from perception visualizations

After training the model, the main experiment consisted in applying the techniques of perturbation analysis and perception visualization to this setup; this allows us to see what the network understands about age in visual terms. We will also simultaneously obtain saliency maps that will highlight which portion of the original image was most relevant for the decision. In subsequent sections, we will explore the capabilities of *Algorithm 3*, by fine tuning the loss function during perturbation using the *force* hyperparameters.

During the perturbation procedure, we can take samples of the output and see how it progressively changes throughout the process, similarly to what was done with the previous experiments on MNIST. This can help us achieve a better sense of which features are changing and which ones are not. As different samples require a different number of iterations to show the desired modifications, results will vary in number of pictures, as samples will be taken at regular iteration intervals.

Some visualizations for the results of this experiment will also be accompanied by the reconstruction obtained through perturbation recycling, which is run on the final result of the perturbation.

Results

All models where trained to at least 95% prediction accuracy, which means, for our purposes, that the average deviation between the ground truth and the model's prediction is on average of $\pm 5\%$ on the whole range $[0\ 1]$. For example, for a sample for which the target should be 0.5, a typical error might see the result in the range $[0.45\ 0.55]$. Performance regarding the reconstructions was not explicitly measured as such a metric would not be relevant for our purposes; visual quality of the reconstructions was instead manually evaluated, and the model accepted if deemed adequate.

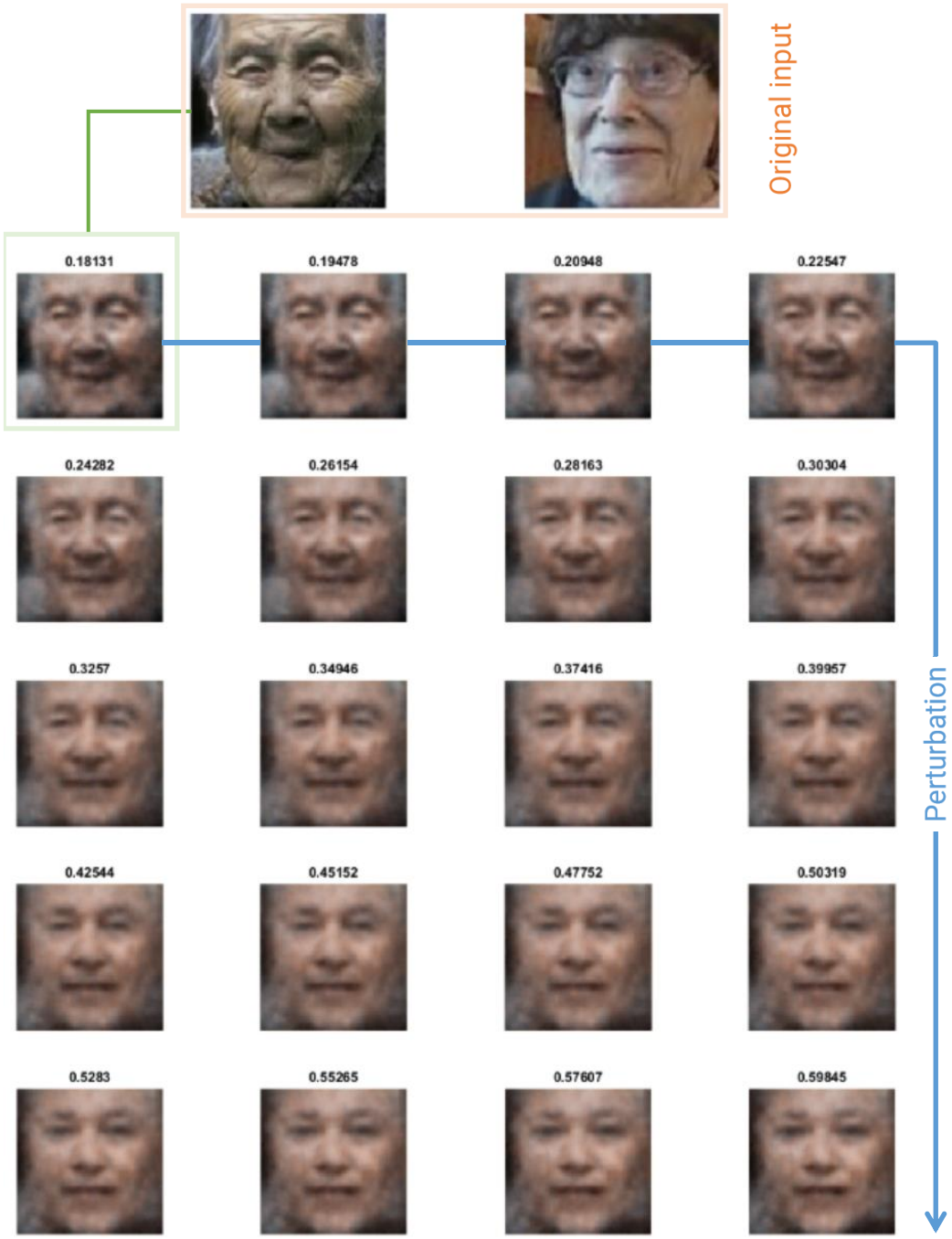


Figure 39 - Youngening of a lady through perception visualization

In *Figure 39* we see the results of our process when applied to a sample composed of images of two ladies. The left input from the original pair is perturbed to force the network to give a lower score. Like for the example for the MNIST dataset displayed in *Figure 33*, the label of the figures correspond to $(1 - \mathcal{L})$ where the loss is with respect to the target value, which in this case is zero, as we would like a lower prediction. Originally, the first sample shows an older figure than the second, we will hence obtain a very high prediction. Throughout the perturbation, we can see how features that are typical of old age such as nasolabial folds start to morph and the reconstruction shows a face that is progressively younger.

However, we still face the same issue of loss of prediction accuracy, as the resulting face gives a prediction of $1 - 0.60 = 0.40$ which would indicate a person which is only slightly younger than the second, non-perturbed sample. Instead, we are posed with a picture that is almost that of a baby.

We can also note that despite the shortcomings regarding prediction accuracy when presented with perturbed samples, our model is able to maintain the identity of the original person and generate a novel face that was never seen in the dataset. This is very important as it shows that our network really captured knowledge about what is age.

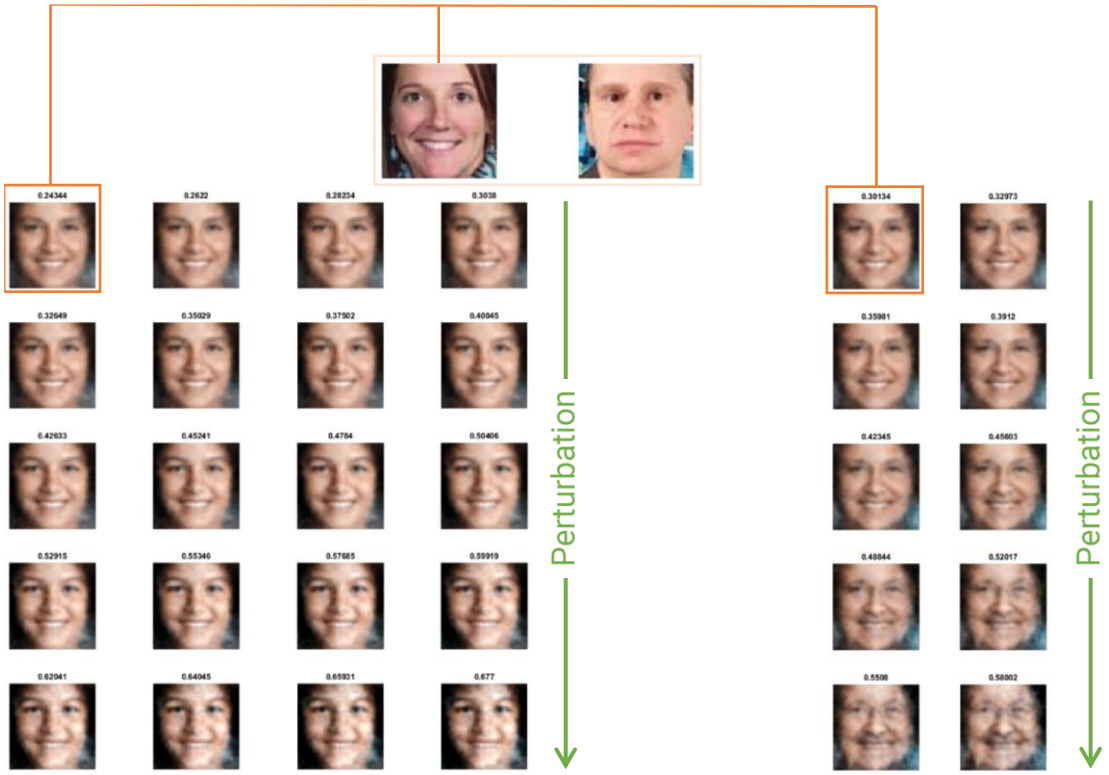


Figure 40 - Perturbation of a woman in both directions, younger and older

In *Figure 40* we show how a sample can be perturbed in both directions, as to make it both younger and older. Samples are taken at fixed intervals, the different quantity of samples to render a younger person rather than an older version means that it might not take the same number of steps to achieve these results.

This is true also between samples: as some regions of the latent space can be flatter than others, we may find samples that require less iterations than others to significantly change. We can however find a sweet spot that accounts for most samples or implement a stopping criterion based on how much the result deviates from the original. Because of the explorative nature of this portion of this work, we have limited our reach to choosing a number of iterations that allows most samples to change as much as needed, and to manually extend this range if the conditions require it. We will see that to base our experiments into a solid theoretical foundation, more well-defined

strategies will be developed to control the perturbation amount and the number of steps that need to be taken.

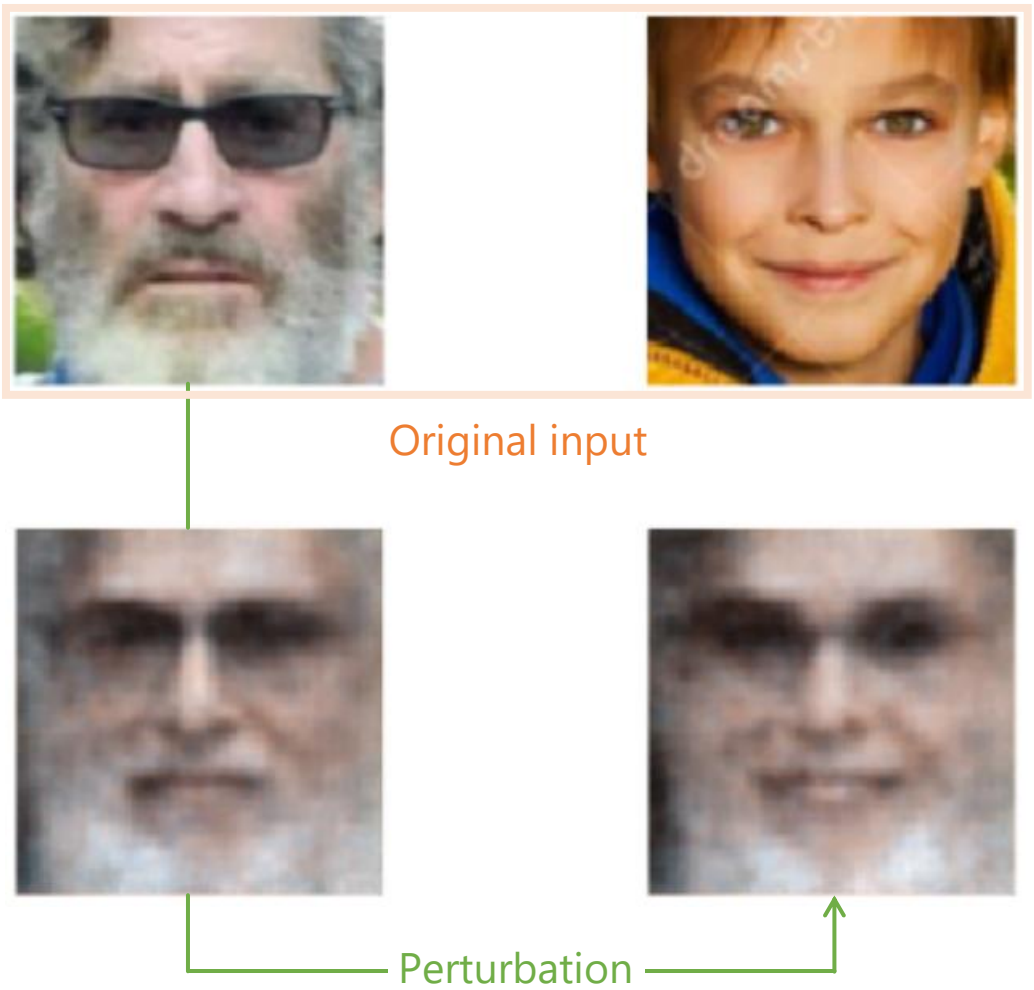
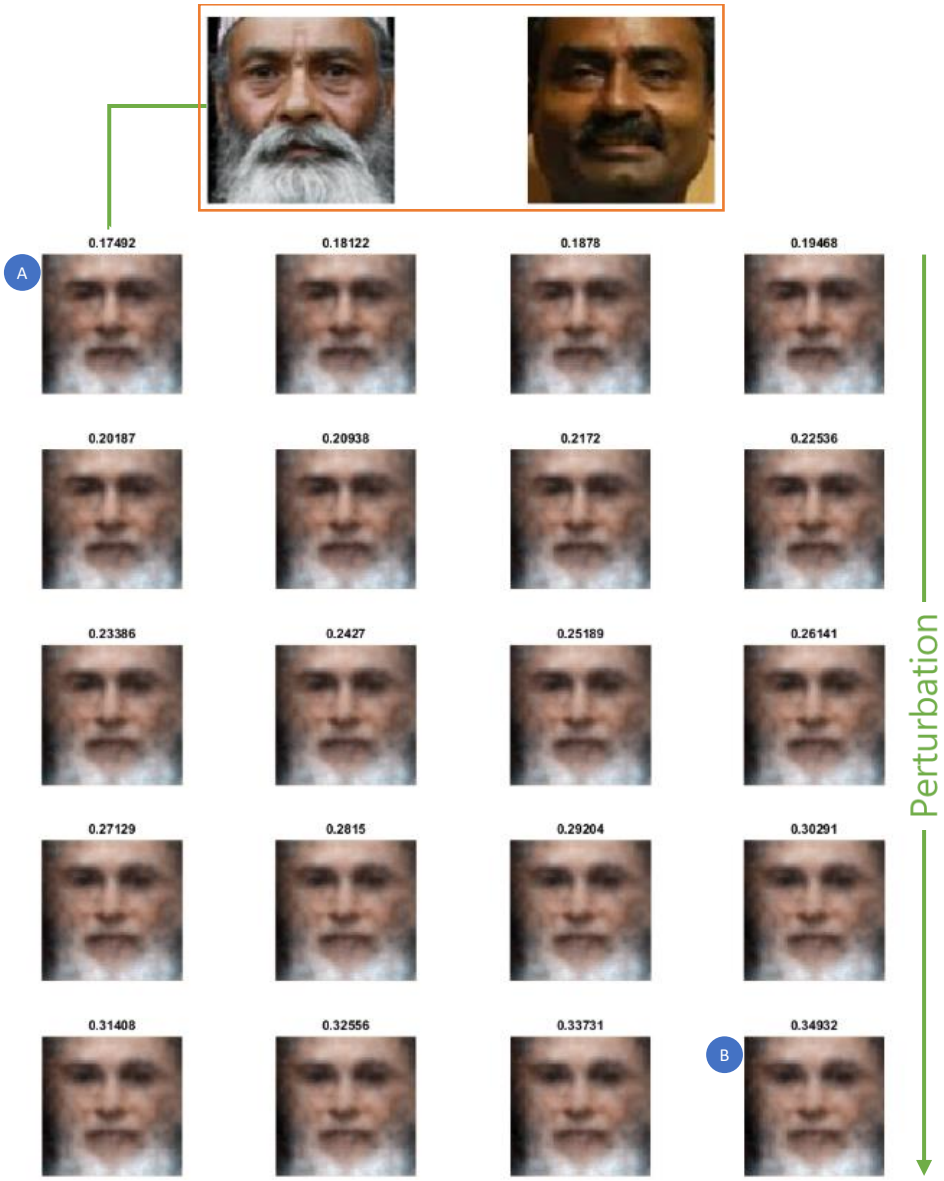


Figure 41 - Perturbation only affects the features that are relevant for the model

In *Figure 41* we show how our method uncovers the model's understanding of the task. In fact, we see how the eyes and the mouth region change during the perturbation, but the beard is unaltered also in the resulting sample. This demonstrates the utility of our techniques is exploring feature importance from a very human understandable point of view. We will also show a different example where this behaviour is coupled with a saliency map, further proving the validity of this concept. For explainability purposes, these perturbations can inform us on which features the model deemed relevant in semantic terms. By analysing which features change and which do not, we can infer which were the ones the model most focused on. Our model correctly identified and altered features that are typical of old age; this is a sign that the information needed to perceive age was correctly captured, and that the model had an organic understanding of the task.



Saliency map

This heat map represents the image difference between the original input image and the resulting input after the perturbation. The original image channels are averaged in the blue channel of this image, while in the red channel we show the changes that were applied during the perturbation.

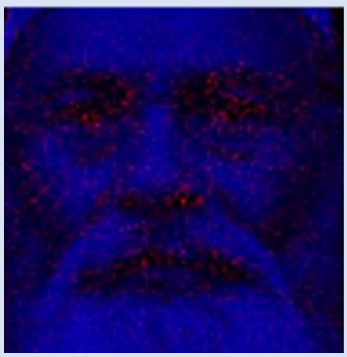


Figure 42 - Feature importance displayed by perception visualizations

In *Figure 42,* we again see how the beard feature was discarded by our model in interpreting age, and thus this part of the image was not changed throughout the perturbation process. In this example, we see the first instance of saliency map that can be generated with our methodologies, and we pair it with what we have said about feature importance.

The saliency map seen in the figure is obtain as described in *Algorithm 2* and in section *Explanation methodologies*, by computing the image difference between the original input that is responsible for the perception visualization seen in the top left (A), and the perturbed input, which is the adversarial attack that generates the perception visualization in the bottom right in the figure (B). To visualize the image difference and superimpose it over the original sample, we average the original image's RGB channels and put the result in the blue channel of the saliency map; we then take the pixel-wise difference between the two inputs and put the normalized result in the red channel of the image. In this example, we have used the barebones implementation of the algorithm and have not induced any bias in the generation of the adversarial attack. What we see here is only the result of the gradient descent method and only the change that most influences the network's result.

The main takeaway is that the changes in the output reflect the changes in the input; in fact, the beard feature that we have seen not to be modified during the perturbation is also not changed to generate the adversarial attack and thus does not show up in the saliency map. We can also see that the major sources of change are concentrated around relevant facial features, namely eyes, nose, and mouth. We also see some artifacts on the left border of the image, which are due to the rectification of the sample in the dataset; indeed, this region is originally totally black due to the warping needed for rectification and is therefore not representative of a natural image; in this cases the model might show unexpected behaviour.

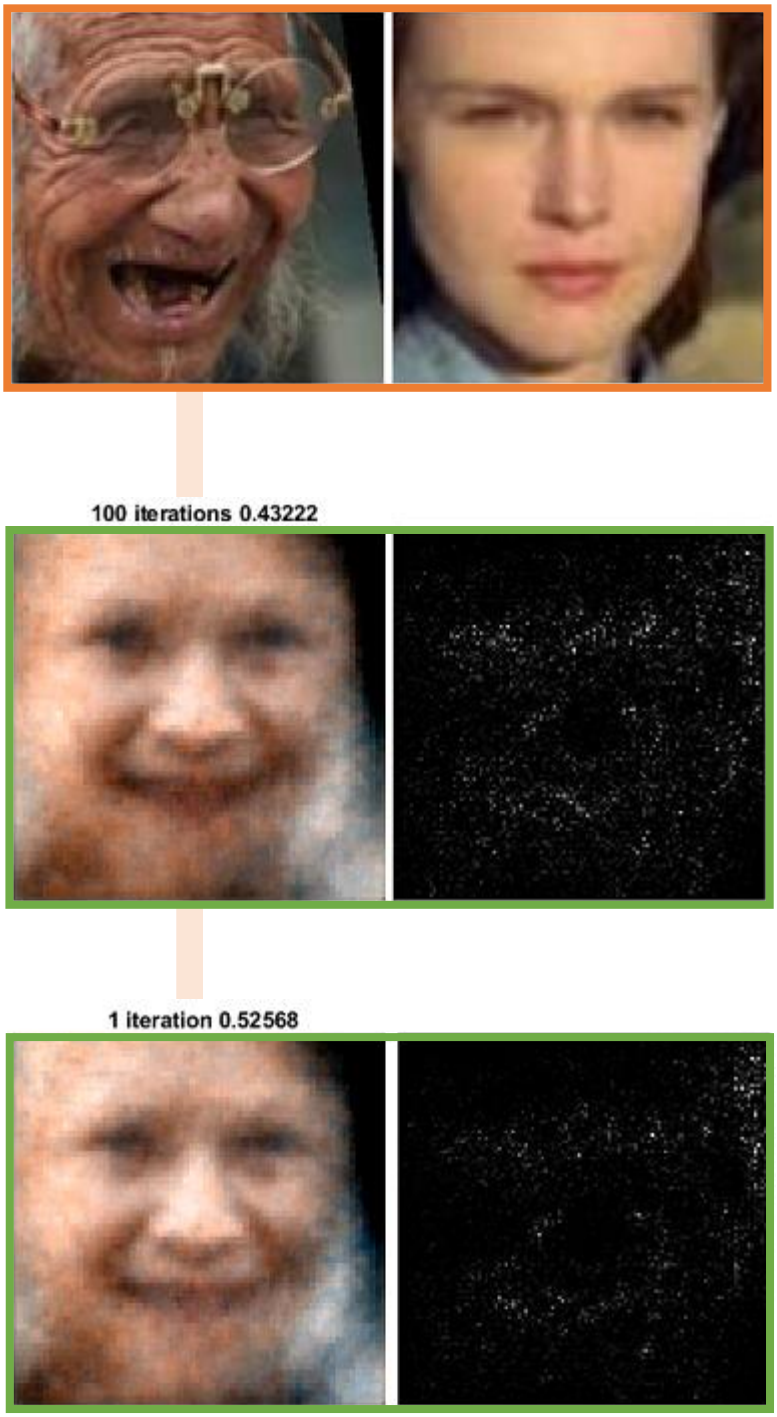
Most importantly, however, we note that the change that was induced on the input image through adversarial perturbation was mirrored in the perception visualizations; not only feature-wise, but semantically. Indeed, pushing the network towards higher or lower predictions generates reconstructions that are semantically relevant for the prediction, even though the adversarial attack only focused on the prediction gradient. This highlights the legitimacy of the adversarial attack as an explainability technique that can give relevant information regarding the network's perception and is proof of the validity of our research.

Finding such semantic coherence in the latent space is not a novel discovery: for example, in [25], researchers show how manipulation of vectors in the latent space can morph the result of a generative adversarial network and give outputs that show a continuous alteration along some semantic *axes*. This means that we can find "directions" in the latent space of a neural network that represent semantic features. In our case and in that of [25], examples of such features could be "age" or "gender". The main difference that we find between the literature and our work is that in the literature these axes are explicitly found by latent space analysis, while in our work they are

implicitly computed by our perturbation methods. In a standard setup, to compute the direction of a semantic feature, say "age", we need to compare the latent space representation of "young" and "old" people, and find transformations that allow to find the trajectory along which these two classes lie; the direction representing "age" will then be orthogonal to those two. In this thesis, instead, models are trained to give an output that tells us the position of the latent sample with respect to the feature range. In the running example, our model will give a low result for "young" people and a high result for "old" people; thus, performing perturbation with our methodologies will implicitly move the latent vector in the direction that semantically represents "age".

5.3.4 One-step attacks

In the previous section, we have shown how our iterated perturbation technique can be used to explore the feature space and obtain a general grasp of what the network learned regarding the task and which features it deemed more important. However, we still need to solve the problem regarding how our techniques relate to the original sample, and to discover whether the iterated perturbation methodology can give us informative results that strictly relate to the original sample. We will show that the iterated technique is indeed valid, and so are also the previous results.



Perturbation iterations

The first input image (top) has been perturbed with a many-iterations procedure in the second row, obtaining a prediction of 0.43 in one hundred iterations with a step size of 0.004.

After the procedure, the relative distance between the original image and the second was computed using standard vector distance by linearizing the image matrix. The result was 0.3684.

A second, one-step attack was finally performed on the same initial input, this time by using a step size equal to the perturbation amount of the previous attack. In the bottom row, we can see the results of this attack.

We discover that there is a strong correlation between the two attacks in terms of saliency map and regions that were modified, but also in perception visualizations. However, we show how the optimized attack generated by the first technique gives a prediction which is closer to the target.

Figure 43 - Correlation between many-step and one-step perturbations

In *Figure 43* we show correlation between the many-step and the one-step attack technique when the attack target is the range boundary 0. To obtain the two, we firstly run the standard many-steps attack as described in *Algorithm 2* on samples A, B and obtain the perturbed sample $A + \epsilon$. We then compute the distance travelled in input space by the algorithm $q = |\epsilon|_{L2}$, and save this value. We can now run the one-step attack and move in input space by the same amount q as for the previous attack. The only difference is that, in a one-step attack, the direction of movement is determined

only by the gradient of the prediction (with respect to the input space we are moving in) at the original image input location.

What we find is that the more efficient many-iterations attack obviously reaches a value which is closer to the target 0, even if it moved the same distance in input space. However, it does so by finding a path which is still in the general direction as the original gradient, and therefore the direction taken by the one-step attack. In the specific sample, in fact, we find that the spearman correlation coefficient between the two saliency maps is 0.71. We also find visual similarity both in the perception visualization and in the two saliency maps. Because our objective is to generate maps that are to be used by humans, this visual similarity further validates the correlation between the two techniques when shown to their final users.

More in general, on 20 trials we found the angle between the one-step and the many-steps perturbation vectors to be limited, being on average of $14°$. The spearman correlation is also very significant in general, with an average value of 0.75.

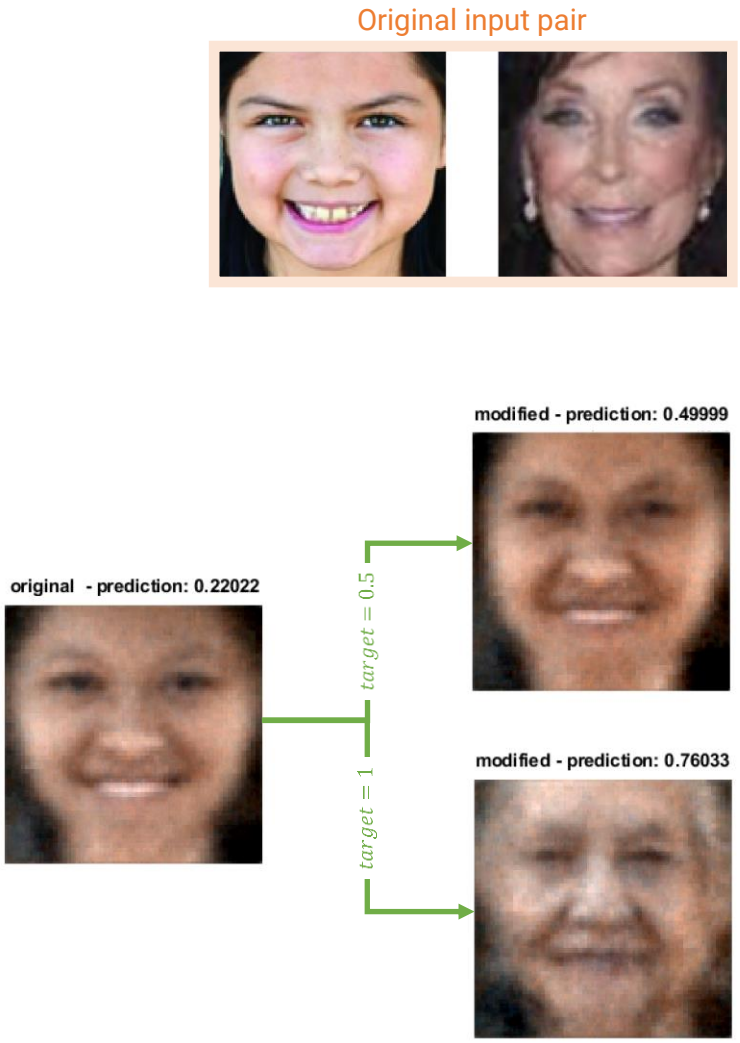
5.3.5 Mid-point attacks

In this section, we will explore our findings regarding mid-point attacks and how the explanations stemming from this technique can be used to link our work to the literature.

Mid-point adversarial attacks have been implemented in the same way as for face age perturbations, as described in section *Mid-point attacks*. The only difference from the other perturbation techniques is the target prediction, which is set to 0.5, instead of 0 or 1. The results of this experiments are obviously similar to the results of the previous section; the conceptual difference, however, lies in what the two different attack methodologies aim to explain. With the previous methodology, we have explored what the network perceives as age in general; now, we will analyse, for each sample, why the model predicted one face to be younger or older than another. In the next section, we will however show how the two concepts are strongly correlated, and that they can be used interchangeably.

Firstly, we feed the image pair to the network and obtain a prediction; we then force the prediction to obtain a mid-point value of 0.5 through iterated perturbations, as described in *Algorithm 2*. At this point, to compute the subset-minimal implicant set that makes up the global explanation, we compute the gradient of the prediction with respect to the perturbed input. All non-zero gradient values must be part of the set, because a perturbation of a feature that influences the output can change the prediction, as the prediction lies on the class boundary. For features for which the gradient is zero, we should check, for every other possible value assignment, if there exists a configuration such that the prediction switches class. If this is the case, then it means that our sample resides in a local stationary point for that specific feature, thus the gradient is zero, but another value exists such that this prediction changes If instead there is no other valid assignment such that the prediction changes, it means that the specific feature is irrelevant and thus not part of this explanation. However, we will see

that almost every feature will be relevant; this means that we can use the whole image as an explanation and ignore the few zero-gradient feature and still have a very accurate approximation of our global explanation, which visually resembles the actual subset-minimal feature set explanation.

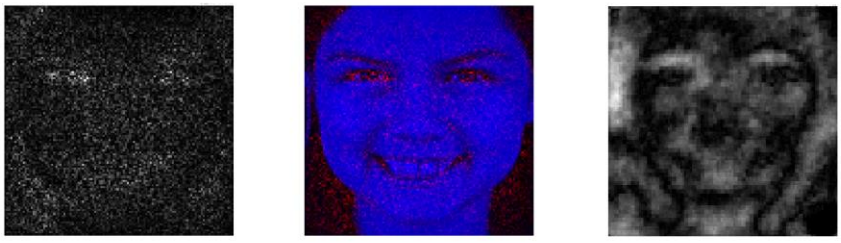


Mid-point attack

The original input pair is perturbed to obtain a modified version of the first sample image that results in a value of 0.5.

We show the perception visualizations stemming from these perturbations and compare the result of the mid-point attack also with the result of the standard attack where the target value for the prediction is set to 1.

On the bottom, we also show explanation maps for the modified sample that results in the top perception visualization and the prediction of 0.5.



Saliency map for adversarial attack

Saliency map for perception vis.

Figure 44 - Example result of a mid-point attack with relative explanation maps

In *Figure 44* we see the comparison between a mid-point attack and an attack which uses target value 1, and provide the saliency maps that can be generated from the perturbed sample resulting from the mid-point attack.

Even if the hypothesis of using the whole image as a counterexample were to fall, for example in cases where the model is completely unaffected by large regions of the input samples, the explanations that we finally obtain will only highlight regions that surely are part of the counterexample in question. This is because the highlighted regions relate to pixels for which the gradient is steepest and, therefore, regions the model is very sensitive to. Possible invariant regions in the input image will never be shown in an explanation generated through our methodologies.

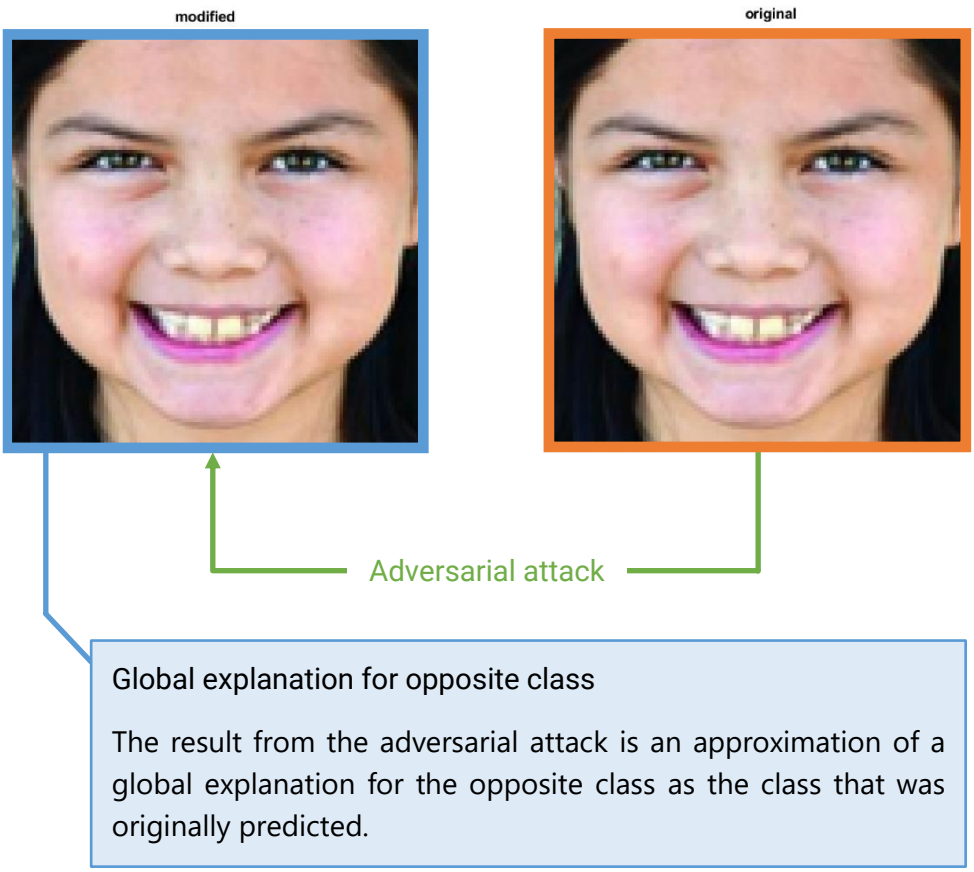


Figure 45 - Perturbation in the input space for the running sample

In *Figure 45* we show the two input images, before and after the mid-point attack. It is clear how imperceptible the difference between the two really is. This shows how we are able to find an explanation for the opposite class which is also local to our original sample, as the distance between the two samples is minimal.

5.3.6 On relating mid-point, one-step, and many-step attacks

With our experiments regarding one-step attacks, we have discovered that the hypersurface representing our model's decision when presented with an input sample is relatively smooth in any given local region. This is demonstrated by the small difference in perturbation direction between many-step and one-step attacks. This

smoothness property of the output surface is typical of well-trained deep learning models where no substantial overfitting occurs, as is the case for networks detailed in our work. It is to note that this property reveals that the many-step path will not only finish in a point close to that obtained with the one-step attack, but that throughout the perturbations, there will be no major direction changes, and that the perturbation vector at any given iteration will have a small angle with respect to the one-step resulting vector.

Extrapolating from these findings, we demonstrate how this correlation between one-step and many-step attack also involves mid-point explanations.

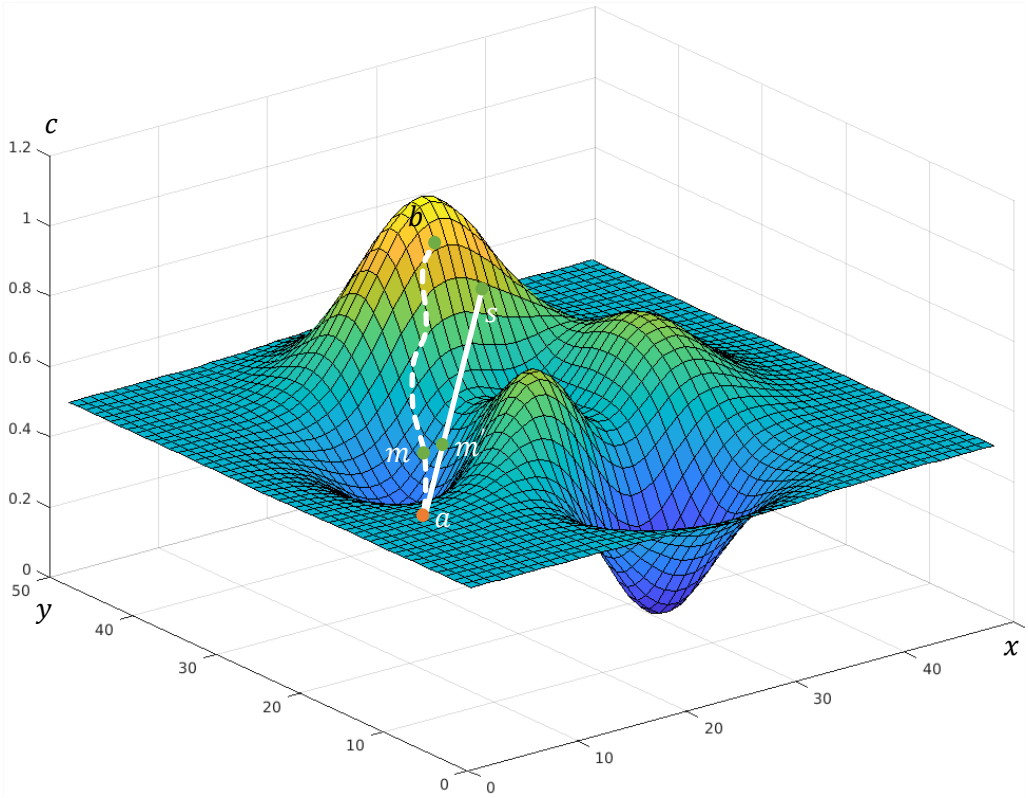


Figure 46 - Similarity between perturbations achieved with different techniques

In *Figure 46*, we see a visualization of what we are trying to show. In the figure, we propose an example surface that represents the output of our model. The horizontal axes represent the vector-space on which all possible input images reside, the vertical axis represents the model's prediction. The starting point of all perturbations is the sample image A identified by point a in the image. What we have demonstrated in the previous sections was that the difference $||b - s||_2$ between the perturbation obtained, starting from image sample a, with the one-step technique (full line) and the many-step technique (dashed line), respectively, is contained. This was experimentally proved by the small angle found between vectors $\overrightarrow{ab}$ and $\overrightarrow{as}$ (projected on the feature axes), which also explained the correlation between b and s.

We now also consider the mid-point perturbations, which are indicated in the figure by points m and m' for many-step and one-step techniques, respectively. These two points lie on the same path taken for their corresponding range-boundary attack because the algorithms used do not change when considering mid-point or range-boundary attack. Indeed, the many-step attack will reach the mid-point and bounce back and forth between the two steps that are closest to the target; the one-step attack instead requires fine tuning of its step amount to reach the required prediction target, but it will always move in the same direction as specified by the gradient of the surface in point a. Finally, we can show that due to the similarity between the two paths and the fact that the mid-points will lie on those paths, then points m and m' must be similar and the images that refer to those points must also be strongly correlated.

This relationship enables to use the results of many-step attacks as though they were coming from mid-point, one-step attacks for the purposes of generating explanation maps. This is because the only parameter that can change the saliency map for adversarial attack, when rescaled to be in the range $[0\ 1]$ for visualization, is the angle of the perturbation vector. In fact, the most salient features, as highlighted by our saliency maps, refer to the dimensions in image space over which this perturbation mostly takes place; but are rescaled to be shown in the visible range in an image. This means that the explanation map for the point m' in *Figure 46* will be the same as the map for point s. We can therefore use the perturbation generated from the boundary-range, one-step attack to describe the mid-point, one-step attack; the latter being the unspoiled source of explanation coming from our theoretical foundation. Furthermore, because the path of the many-step attack (its projection on the image features hyperplane) is quasi-linear, for the same reasons as for the one-step attack, we can use the perturbation generated by the boundary-range, many-step attack to have a good approximation of the map resulting from the mid-point, many-step attack. Finally, because m approximates m', we can use point b to generate explanations which are approximations of the theoretical explanation coming from m'.

We have therefore shown how many-step attacks (both in boundary-range and mid-point form) can be used to generate perturbations with our techniques which are very similar to those that would result from mid-point, one-step attacks. We have also experimentally seen, in the previous section, how the relationship shown here translates in visual similarity of the saliency maps resulting from those perturbations. Because the result is to be given to a user for inspection, our goal is that of visual similarity and we here show that it is reached. Therefore, we can safely use the saliency maps achieved by boundary-range, many step attack perturbations and still fall within the solid theoretical foundation of [20]. The reason to do so is that generating mid-point, one-step attacks is very time consuming, as tuning must be done in order to find the correct perturbation amount to reach the class boundary. The process is instead automatic when using many-step procedures; as stated before, for this kind of attacks, when the target is reached, the algorithm oscillates and therefore there is no need for tuning of perturbation range. Finally, we can also validate the results of the previous

experiments as correct explanations even if they were obtained using techniques different from mid-point, one-step attacks.

A possible alternative to using our standard many-step, mid-point attack is to use a variant where at each step we always proceed in the same direction specified by the gradient at the starting point, and stopping when the target 0.5 is reached. This would allow to automatically compute the perturbation amount, as it is the case for many-step attacks described before, but avoiding the slight direction changes that come with those techniques. This process is however rendered useless because, as we have shown above, we can obtain the same, untampered, mid-point, one step saliency map from a boundary-range, one-step attack.

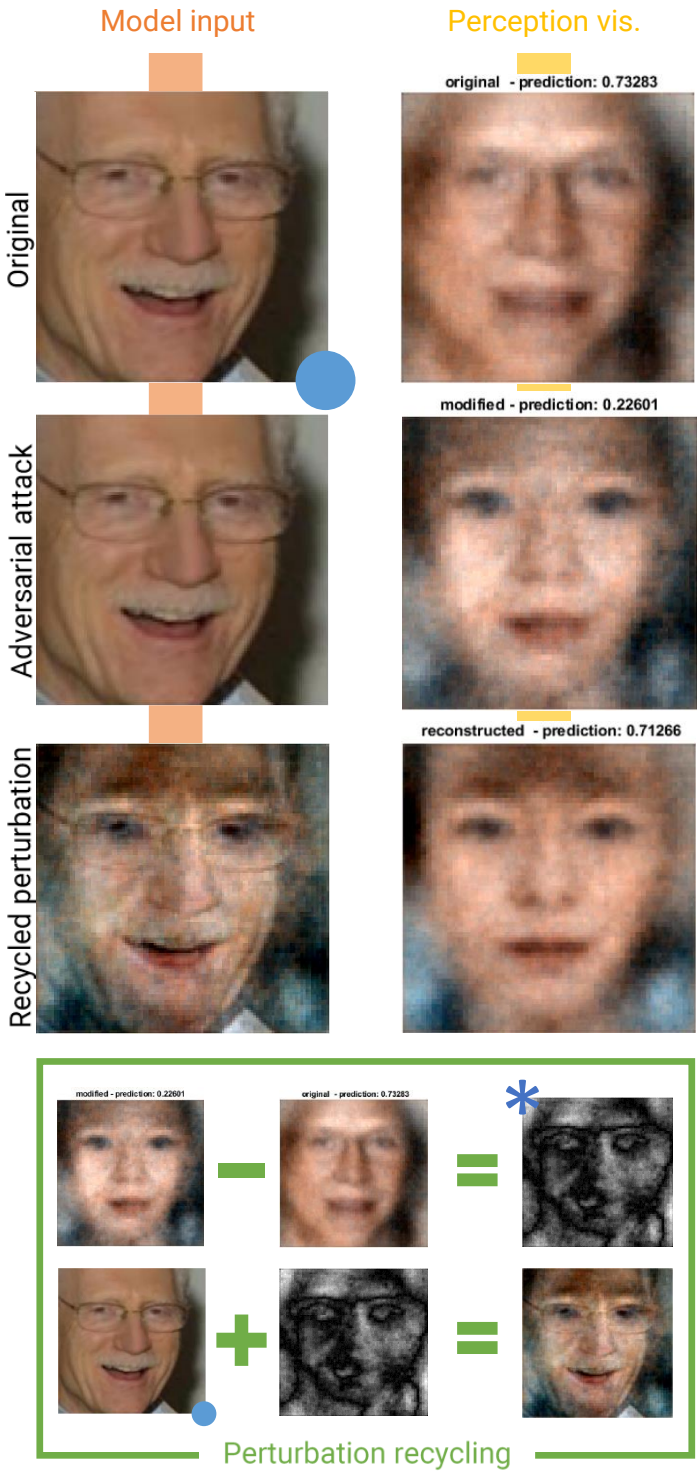
The reason for describing and analysing the various attack techniques is that although the saliency maps that we generate are strongly correlated, the reconstructions coming from their respective perception visualizations will not be equally similar. Obviously, moving in the feature space for different extents will translate in widely different reconstruction outputs. Opposite to what we have seen for saliency maps, the governing factors that drive how the reconstruction change are both the angle and the intensity of the perturbation vector. However, because the angle between the many-step and the one-step attack is rather small, we can assume that, if our focus lies within the examined methods, the amount of the perturbation is the main concern. This means that, independently of the step count, mid-point attacks will provide similar explanations, and the same is true for boundary-range attack, when the perturbation amount is the same. The similarity will however be stronger for mid-point attacks as the absolute distance will be smaller.

In the previous experiments, we have discovered that the feature space contains some notion of "age" that displays continuous behaviour along some direction. This property is typical of feature extractors like our neural network; we will use this to our advantage in order to choose which type of attack to use. If our focus is to validate the explanations that are obtained through perturbation analysis and adversarial attacks, we would like to emphasize whether our perturbation achieved the desired effect of deceiving the network into thinking that the input sample is of someone older or younger (in our specific case study) than the original input. In other words, we would like to maximize the effect the perturbation has on the latent space in order to show whether the network's perception matches the perturbation target. For this purpose, range-boundary attacks provide the greatest possible modification while generating saliency maps that strictly relate to the original gradient information, and hence to the original sample. In *Appendix A* and *Appendix B* we provide additional examples of how boundary-range, one-step attacks can generate meaningful explanations and also provide significant modification to the reconstructions that can be used to verify whether the network perception is coherent with the perturbation.

5.3.7 Perturbation recycling

We want to verify whether the methodology of perturbation recycling can tell something relevant about the model's decisions. We will do so by analysing the effect the procedure has on the perception visualization and compare it to results provided by the other perturbation techniques.

Following our formal setup, we would like to see whether there is correlation between $p_\psi(A+\epsilon)$ and $p_\psi(A+d)$, where $A+\epsilon$ is the perturbed input sample given as a result after the perturbation process through adversarial attack, and d is instead the difference $d = p_\psi(A+\epsilon) - p_\psi(A)$ between the perception visualization after and before the attack.



Perturbation recycling

In this example we see the effects of perturbation recycling. In the top row, we see the original sample and its relative perception visualization, in the middle row we can see how the adversarial attack, albeit imperceptible in the input, has sharply changed the model's perception. In the last row, we see how the recycled perturbation affects the model's perception, and it is clear that it does so in a way that is independent of the model's prediction.

At the very bottom, we display the algorithmic steps required by the perturbation recycling technique to achieve the sample used to obtain the result in the bottom row.

* The visualization shown here displays, as described in the *Explanation methodologies* section, the absolute value of the actual difference map that was used to generate the recycled perturbation, as negative values could not be visualized in an image.

Figure 47 - Perturbation recycling result example

What we find with perturbation recycling, both as seen in *Figure 47* and in general, is that the perception visualization resulting from the recycled perturbation $A + d$ strongly resembles the visualization obtained from the adversarially perturbed sample $A + \epsilon$. However, the model's prediction obtained from the former does not fall in line with what would be expected. Indeed, we discover that the prediction coming from recycled perturbations can widely differ from that obtained from the adversarially perturbed sample, and that it may sometimes also diverge to a different direction. For example, we have observed instances where although the adversarial attack brought the prediction towards a lower value, the recycled perturbation gave a higher prediction than the original; this was also observed for the opposite case.

The main result that can be taken away from these outcomes is that perturbation recycling generates a sample which is able to fool the network perception, and thus give us a perception visualization that resembles a younger/older person, but this sample is not in general able to fool the network regarding prediction. This can be explained by some ability of the network of disentangling the two tasks it was trained to solve: reconstructing the input images and comparing them. This is also clear when we look at the low correlation between d, which describes the region of the perception that were most perturbed, and the adversarial noise ϵ.

We finally find that perturbation recycling cannot be used to produce a relevant saliency map, however, the difference d between the perception visualization before and after the perturbation is still very useful to analyse and verify the procedures of explainability through adversarial attack. Indeed, we see that the visual changes in perception visualization always follow the model's prediction when presented with an adversarial attack, so that when the attack aims at making a sample older, the perception visualization will transition to a visually older reconstruction, and vice versa when making samples younger.

5.3.8 Advanced perturbations

As described in *Algorithm 3*, we can use the three contributions described in *E 25* to perform more targeted perturbations; to do so, we modify the adversarial attack objective function. To keep the analysis pure from biases that could generate from these procedures, the advanced perturbation techniques have not been applied for the other tests. In this section, we will analyse which are the effects of the $force$ factors and how the explanations change when we make use of those.

The results show that the parameters $DiffForce$ and $ZeroForce$, which respectively put importance on how much the sample differs from the original and how many pixels were left intact in the perturbation, actually do not have significant effects on the perturbation process. Instead, the parameter $TVForce$, which drives the perturbation to be composed of contiguous patches of modified pixels, has a very noticeable effect on the resulting perturbation.

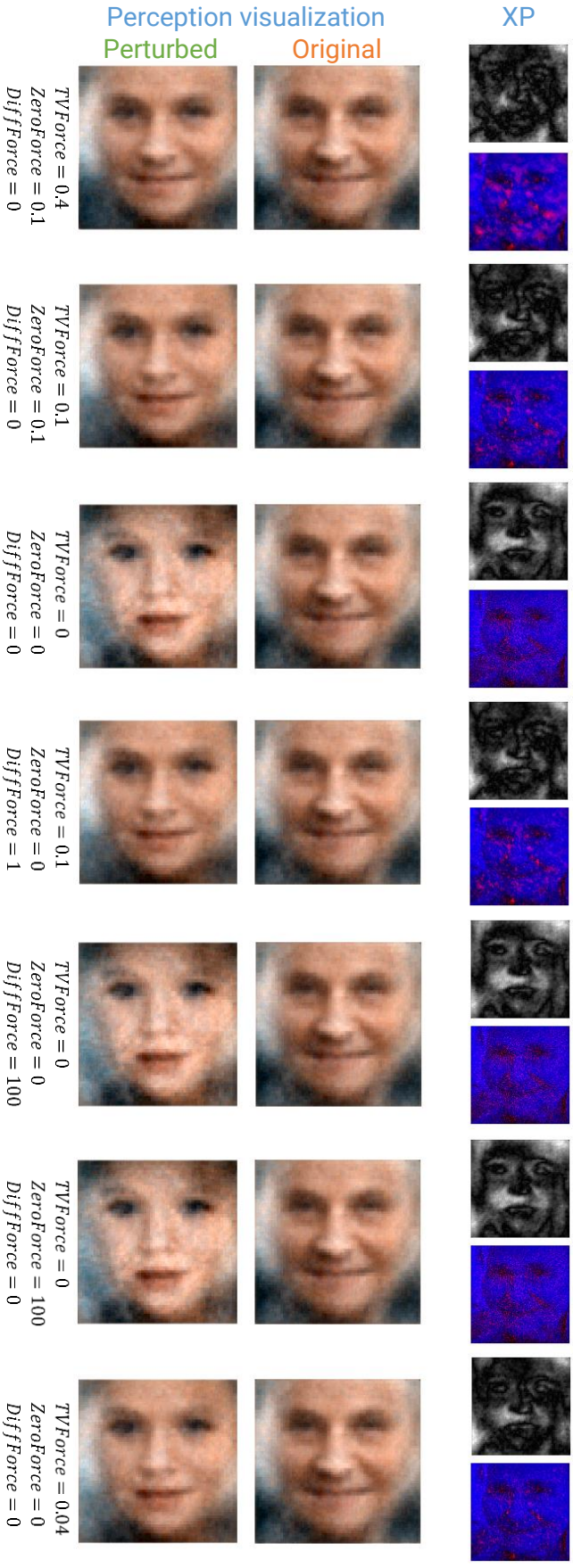


Figure 48 - Advanced perturbation results

From the results in *Figure 48*, it is clear that the explanations show sharp modifications only in the cases where $TVForce$ was used. Even when the other two parameters were used in extreme values, the explanation does not significantly change. Also, we discover that when $TVForce$ is used, the procedure gives a result which does not look as young as when it is not used.

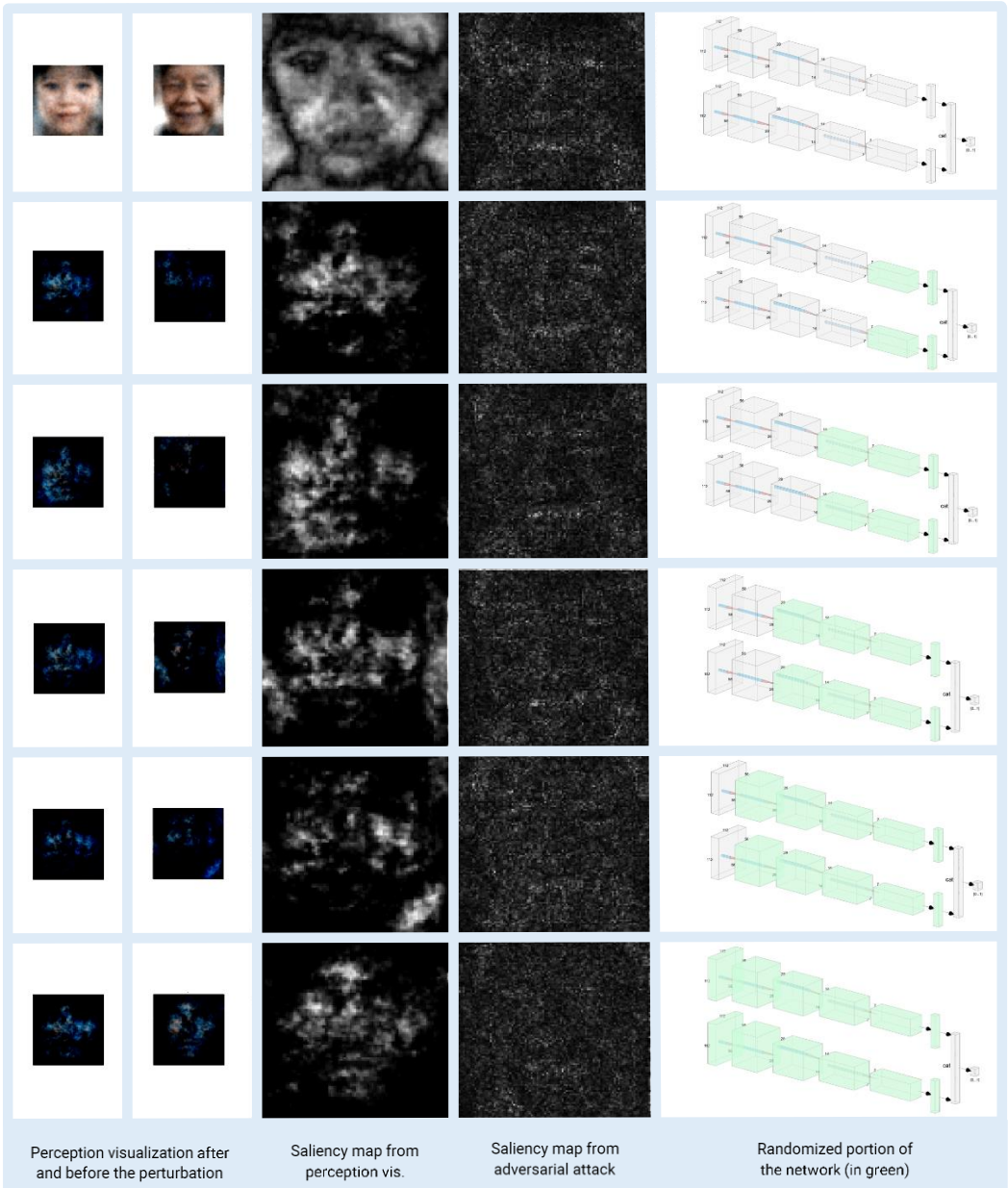
Due to this very fuzzy nature of the results, and the uselessness of two of the three parameters used, it is clear that the technique of advanced perturbations is not, at least in this setup, a viable technique to generate adversarial attacks. In our experiments we have indeed only used the standard unbiased perturbation technique, as we deemed this research path not valuable for our goals.

5.3.9 Validation

Sanity checks for saliency maps

We further verify the legitimacy of our methodologies by following the validation techniques described in [18] by performing the so called sanity checks for saliency maps. These consist in verifying whether the results of the visualization technique are model dependent and label dependent, thus confirming the thesis that the applied technique tells us something useful about what the model is perceiving. These tests have been performed against the saliency map generated with the edge-targeted, many-steps adversarial attacks. Following the findings of the previous sections, the results are valid also for maps generated with mid-point attacks and one-step attacks.

The first sanity check verifies whether the explainability technique is sensitive to the model; if it were not, then it would mean that the resulting visualization only depends on the input and can't be useful in understanding or troubleshooting the model. To perform this check, we take samples and run them through our trained model; we then construct another model with the same architecture but with randomized parameters, and feed to this model the same samples. We finally compare the results and compute correlation both visually and by using Spearman correlation. Randomization of the network's parameters is performed in a cascading fashion starting from the deepest layer ($flatten$) and then iteratively randomizing the other layers up to the first convolutional layer.



Layer weights and bias randomization

In this figure we show the results of the cascaded randomization test, where at each iteration we randomize increasingly large portions of our network starting from the deepest layers. We see the sharp decline in performance for perception visualization and the gradual loss of visual features also for explanation through the standard adversarial attack.

Figure 49 - Cascading randomization test

In *Figure 49*, we see the results of the parameter randomization test. In the figure, we show the different maps obtained through our various methodologies. The first two columns represent the perception visualization after and before the perturbation process; we see that the performance of this technique sharply declines already from the first step of randomization. This is easy to understand as the autoencoder capabilities of our model are very fragile and sensitive to the symmetry of the architecture, which is disrupted with randomization. From the difference between the first two columns we obtain the third column following the methodology of the saliency map for perception visualization as described in the previous sections. Also in this case, we can see a sudden decay in explainability performance; image features are progressively lost starting from the first iteration of randomization. In the fourth column, we present the saliency map computed from the adversarial attack that was generated through the perturbation process. Throughout the randomization iterations, we see a slow decline in visual features in the map, up to the last instance where we can barely see some relevant structure. However, we might still make out some facial features from these maps. The reason for columns three and four to still present some features is due to the intrinsic structure of CNN; in fact, these networks are built to solve visual tasks, hence they are very sensitive to edges in the input images. To discern between whether our explainability techniques are merely edge detectors or if they are able to tell us something more, we must analyse the differences between the maps at various randomization stages to see whether they are strongly correlated, in which case we must reject our methodologies. If instead we find no correlation between explanations at different steps in the randomization process, we can confidently assume that the information given by our techniques on a non-randomized network is relevant to the model's parameters. In the chart in *Figure 50*, we plot the correlation coefficient between the original adversarial attack saliency map and the same saliency map at every randomization iteration using spearman rank correlation.

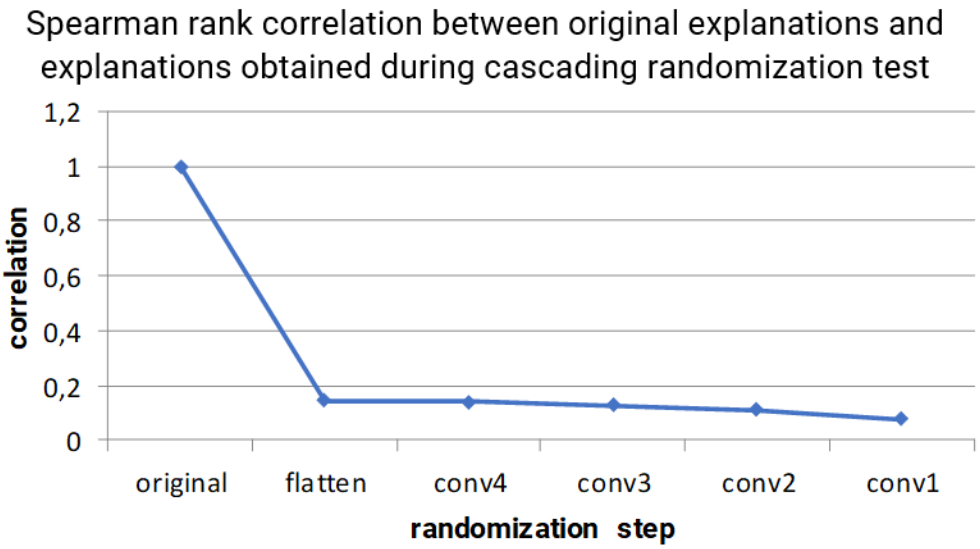
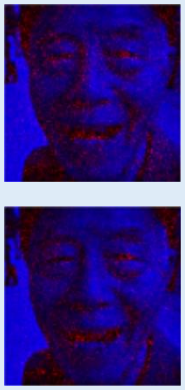


Figure 50 - Correlation for cascading randomization test

The takeaway from *Figure 50* is that there is no significant correlation between the original saliency map and the ones generated after randomization, even at the first step. We can hence confidently assume that this saliency map is sound from the perspective of the parameter randomization test. For the other explanation maps presented in *Figure 49*, visual inspection is enough to note the very visible differences between the results at different steps in the process.

The second sanity check regards label sensitivity; if explanations do not depend on the labels in the dataset, it means that they cannot retrieve knowledge regarding the decisions of the model and how those are based on information present in the dataset. To perform this test, we train a copy of our model on a modified version of the dataset where the labels (in this case the target normalized age gap values) are randomized. Finally, we feed a set of samples to both our original model and this randomized model and see whether the explanations differ. Similarly to what was done in the first check, verification of difference between explanation is done both visually and with Spearman correlation.



Label randomization test

In this example, the figure on top is the standard superimposition taken from the correctly trained model, while the bottom is taken from a model trained with random labels.

Figure 51 - Label randomization test example

Because even with random labels the model still has learned to reconstruct faces and correctly act as an autoencoder, the explanations will still present relevant features, as we see in *Figure 51*. However, it is curious to note how the differences in the two saliency maps reflect the fact that the second network is not able to discern age as well as the first. We can see this, for example, in the nasolabial fold and on the edge of the nose, which is much more relevant for the first network, as we see in the figure. This fact is also substantiated by how our original model focuses a lot of attention on these particular features when modifying the perception visualization. As the two images are, however, visually similar, we still refer to Spearman rank correlation to formally check for the relationship between the saliency map E obtained with the original model and E' obtained with the model trained on random labels, and find that it is not significant:

$$corr(E, E') = 0.1270 \qquad E\ 38$$

6 Conclusions and future work

6.1 Contributions

Our main contributions to the explainability research field lie in our newly developed explainability techniques and in the verification method for those, that takes form in perception visualization. We emphasize the great need for the artificial intelligence community of exploring the explainability research field and, with our work, we hope to have helped achieve this goal.

Between the various techniques that were analysed in this work, we focus on *saliency map for adversarial attack* (in its many forms) as the most successful of our developments regarding explanation techniques. The approach proved to be valid in giving novel and relevant information regarding the model's decisions, and it reached the explainability desiderata detailed in [13].

Not less important in our findings, hence the title of this thesis, is the technique of perception visualization. Even though similar latent-space exploration practices were developed in previous work, these never found their way into explainability until now. We strongly underline the importance of guaranteeing the quality of explanation methodologies from the point of view of the latent space, and of verifying them using the network's perception of the input.

While we have focused on these two main achievements, our otherwise extensive analysis of architectures and methodologies in the specific setup allowed to explore many possible research routes and distinguish the most promising from the rest. We think that having also detailed the failed methodologies will greatly help future readers in their research. It is not however to say that the routes that were not fully explored in this thesis are to be deemed invalid; in fact, we encourage the development of future work that may follow those paths, and provoke the research community in building upon our failures to find novel solutions to our shared problem.

6.2 Applicability

Even though our achievements show to be valid and novel, in this thesis we have only applied the developed techniques to non-critical tasks. We argue, however, that many critical applications can benefit from our findings and that the flexibility of our work allows the implementation of the methodologies seen here also to production environments.

An example of such scenario is that of authentication, like in cheque signature verification. In our work, Siamese networks have been used for comparison, but as stated before, they have extensively been used in the literature to compute similarity. One may think of applying our explainability methodologies to such type of models and obtain explanations of why the model thought that a specific signature was or was not written by the legitimate user. A saliency map that highlights the regions in the signature that are most suspicious may help a verifier user (the bank teller in this instance) in confirming whether the discrepancies found are irrelevant, or instead if

they are indicative of an actual fraud. This could help construct better models by allowing users to give feedback, and increase interaction between user and machine, but also point out possible acquisition issues with the signature input device.

While typically signatures are not provided as image input to Siamese networks, thus rendering perception visualization not applicable in this context, we may find other critical scenarios where the technique can be used to verify explainability techniques. The flexibility of perception visualization allows it to be used in any situation where we are trying to validate saliency maps for networks that process images. One such case is in medical imaging, where performing perception visualization on networks trained to recognize cancer cells may help in understanding the perception the network has of such cells. Similar techniques as those used in our thesis may be used to generate perturbations and finally saliency maps; in this context, perception visualization can help visualize the perturbation and see what features of the cell the network deemed important to determine its cancerous nature.

6.3 Further developments

There are also a series of improvements that can strictly relate to our work that could be developed to improve the scientific relevancy of this thesis. Most importantly, we highlight the very interesting property of perturbation recycling of generating attacks that semantically target the network perception independently of its prediction value. A more in depth study of these phenomena could lead to discoveries regarding the nature of adversarial attacks more in general and improve on the work started by [19]. Moreover, studies could tackle the inherent vulnerabilities of deep models to improve their security; in this regard, perturbation recycling could be a starting point to analyse attacks which are semantically relevant for the network.

Another section of our work that could be improved upon is that of advanced perturbations. Even though the research direction was discarded in our thesis, as we deemed that unbiased attacks were most relevant for explainability purposes, the technique of advanced perturbations might be another source of adversarial attack that could shed light, in future studies, on some of the weaknesses of deep neural networks. Future works could improve upon our findings and formulate better attack strategies to finally produce adversarial attacks that possess specific properties, such as those found using the $TVForce$ hyperparameter, and that can show different ways in which a network can be fooled. As highlighted in [3], this kind of semantic attacks pose a threat to our future technologies such as self-driving cars; therefore, there is a need to protect our models, and most importantly to know their enemy.

We could also think of building some additional form of explanation by using input masking. This methodology consists in blacking out specific regions of the input image to see how the output changes. Examining output differences in relation to the masked regions is akin to analysing the features contribution in a partial dependency plot: we can see how specific features affect the output and assign an importance value to each region and feature of the image. For a dataset like UTKFace, this could prove very useful as important face features like eyes and mouth are always found in similar locations in

the image; we can therefore black out specific face features and see how those changes reflect in the output.

Lastly, we recognize that our work has been developed without a specific end user target. The different maps that we have developed may in fact be more or less appropriate depending on the expertise level of the user. This translates in a loss of *customizability* of the explanation techniques. Because this work has an explorative focus, it was impossible to think of a specific user segment; therefore, the current solution is to present to the user the whole range of explanations. The drawback of this is a loss of *parsimony*: such a series of explanations might be overwhelming and redundant, making it more difficult to understand the explanations. In a production environment, we must profile the end user that will finally see the results of a model that makes use of our techniques. In this way, the explanations that are given to the user are reasonable with respect to their understanding of the model. It is however possible to devise, in future works, some way to classify the different maps developed in this thesis to help a system integrator in selecting the most appropriate for the user's skill level.

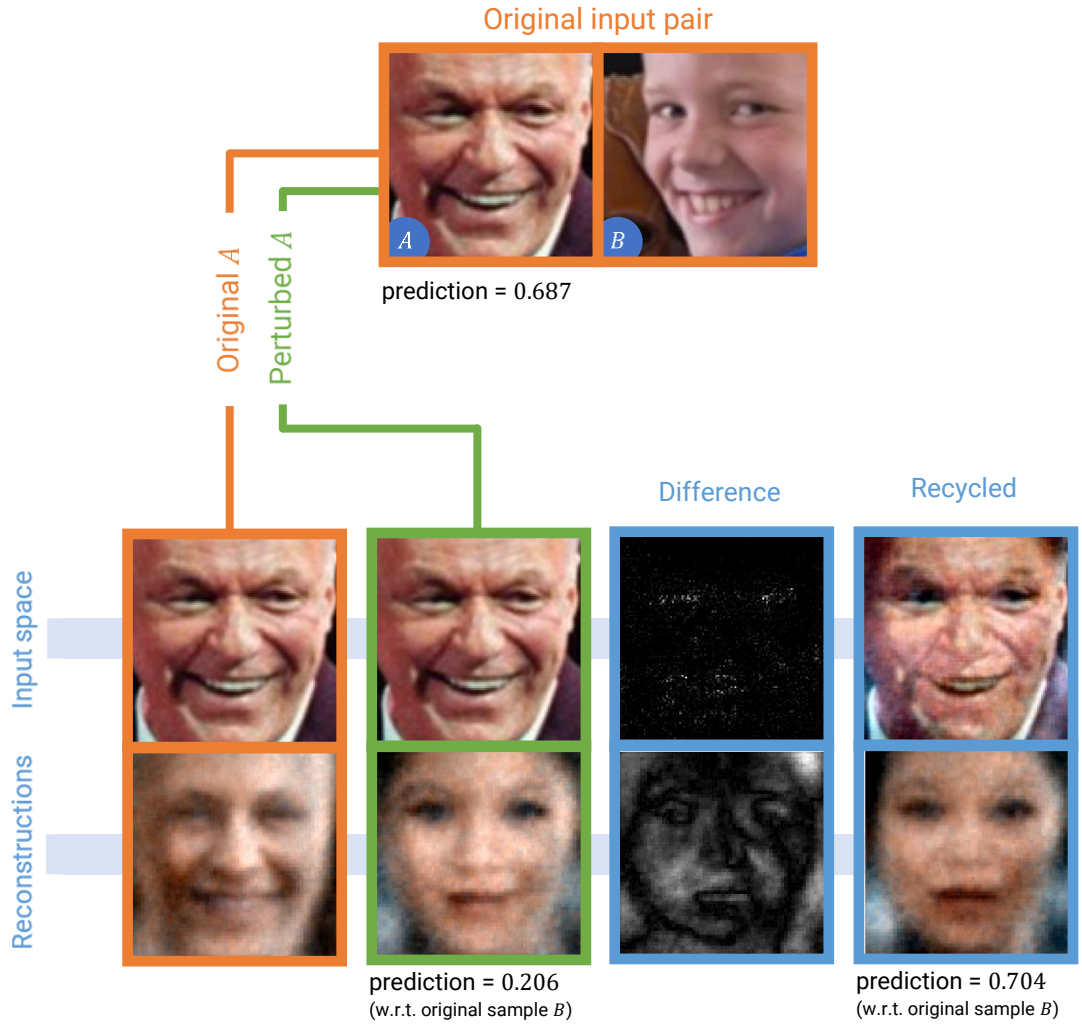
6.4 Drawbacks

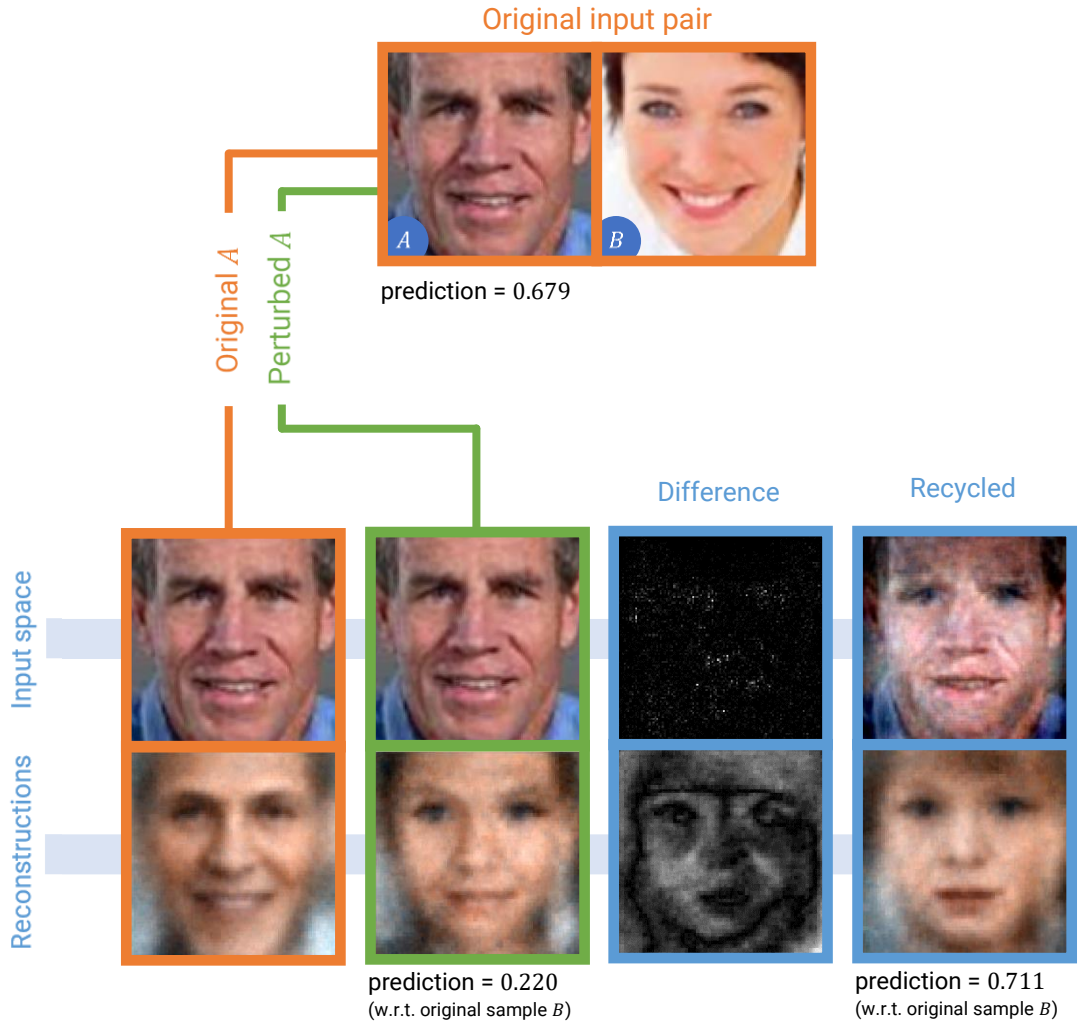
As always in any context, there is no such thing as a free lunch. Our work makes no exception, and as we have previously discovered, there is a trade-off between model performance and explainability potential of the model. In our case, this trade-off stems from the fact that we train the network to solve two tasks simultaneously; henceforth, the performance of the single tasks will naturally suffer. However, as long as the performance of a model remains satisfactory, as is the case for our studies, we argue that explainable models are much more valuable than their non-explainable counterparts. Another note that must be made is that an explainable model can be debugged by using the generated explanations, therefore, the performance dip can still be countered by analysis of the results together with the analysis of the explanations.

A limit of our work regarding perception visualization is instead the hypothesis of having a white box model. This restricts the flexibility of our techniques as it is not always possible to train a model from scratch. The explanations provided through our method, furthermore, leak essential information regarding the model's gradient. It is important, in case the end user of such explanations is the public, that the information revealed is thoroughly studied, in order to avoid security issues.

Appendix A

Additional explanation examples – attack target 0 (younger), one-step attack

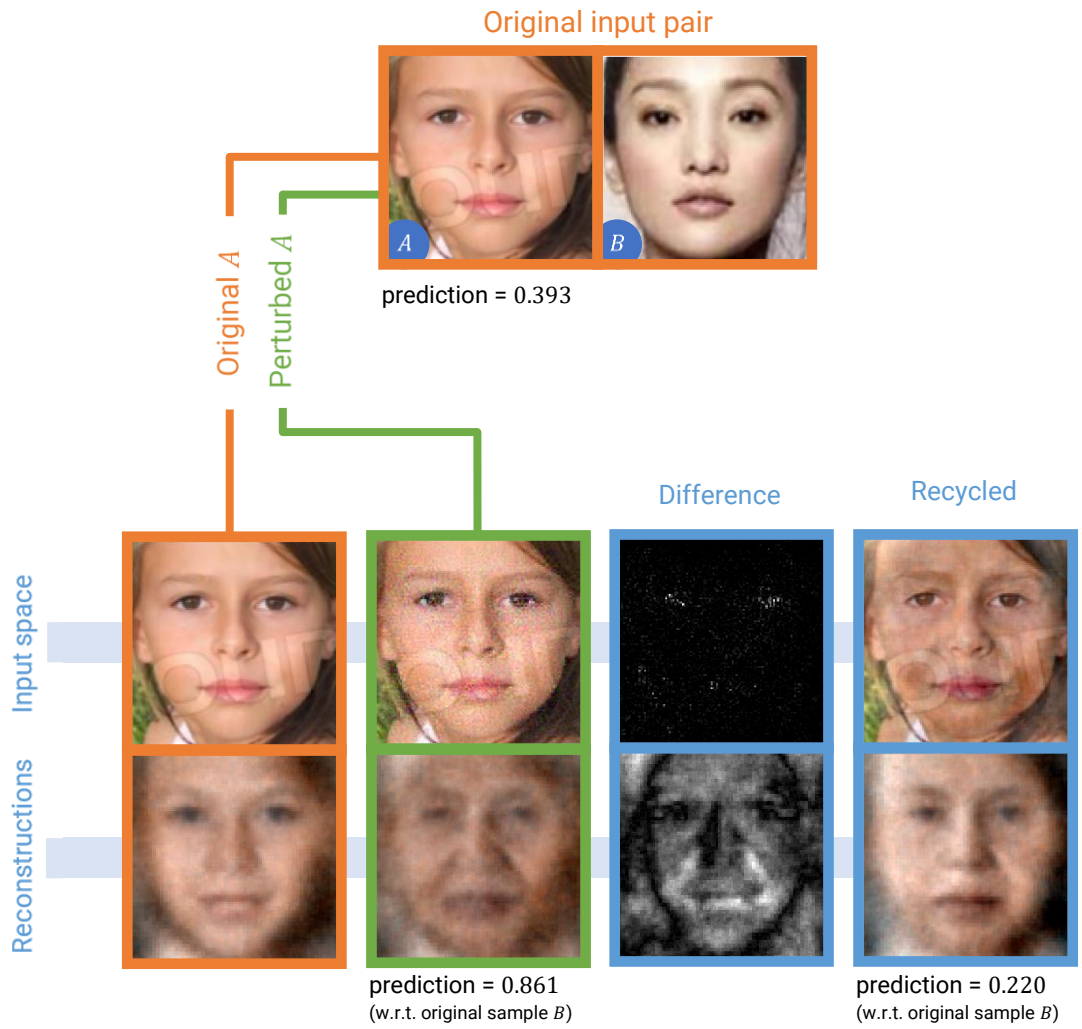


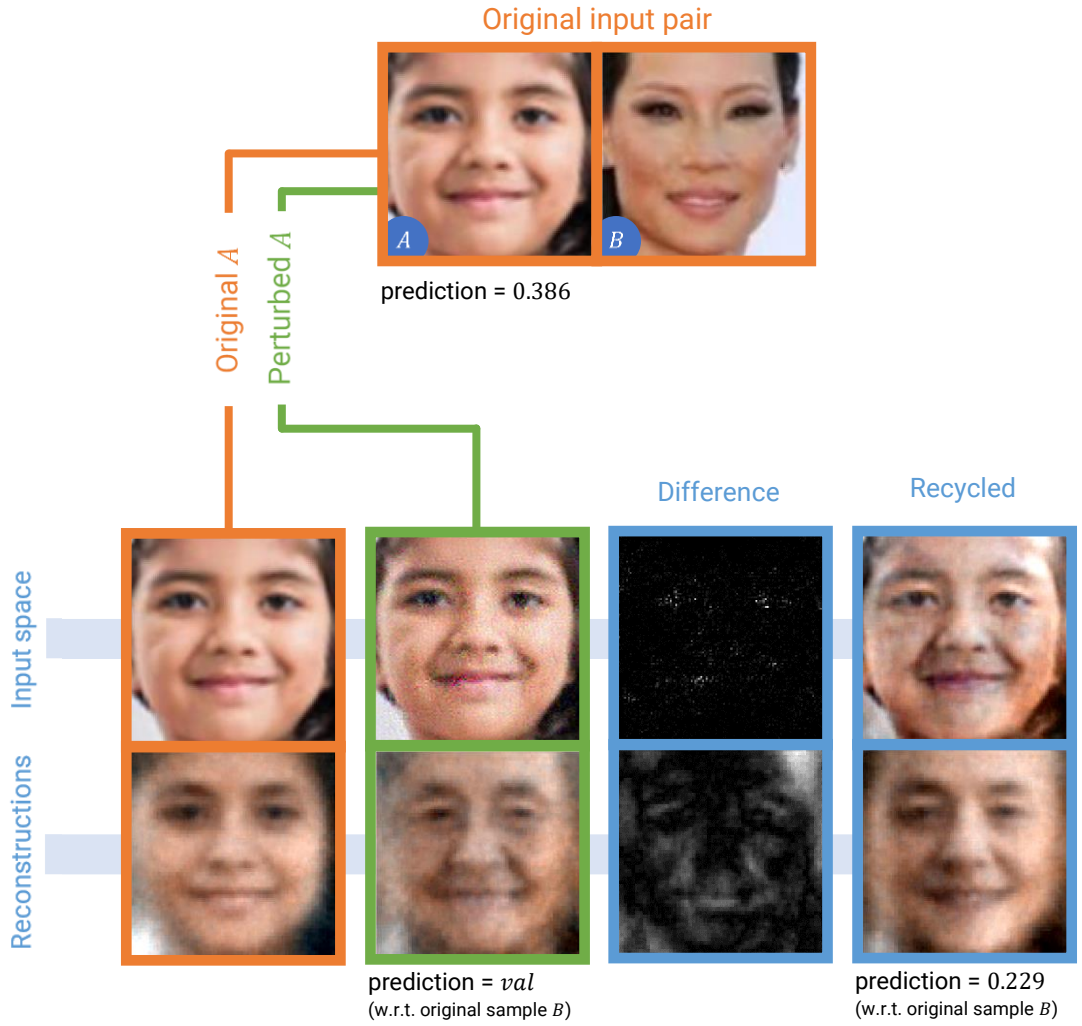

Original input pair
A
B
prediction = 0.679
Original A
Perturbed A
Input space
Reconstructions
Difference
Recycled
prediction = 0.220
(w.r.t. original sample B)
prediction = 0.711
(w.r.t. original sample B)


Resulting saliency maps
Superimpositions
Segmented

Appendix B

Additional explanation examples – attack target 1 (older), one-step attack




Original input pair
A
B
prediction = 0.386
Original A
Perturbed A
Input space
Reconstructions
Difference
Recycled
prediction = val
(w.r.t. original sample B)
prediction = 0.229
(w.r.t. original sample B)


Resulting saliency maps
Superimpositions
Segmented

Bibliography

[1] A. Das, P. Rad, Opportunities and Challenges in Explainable Artificial Intelligence (XAI): A Survey, arXiv:2006.11371v2.

[2] N. Morgulis, A. Kreines, S. Mendelowitz, Y. Weisglass, Fooling a Real Car with Adversarial Traffic Signs, arXiv:1907.00374v1.

[3] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, D. Song, Robust Physical-World Attacks on Deep Learning Visual Classification, arXiv:1707.08945v5.

[4] A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks In *NIPS*, 2012.

[5] D. G. Lowe, Object recognition from local scale-invariant features, Proceedings of the Seventh IEEE International Conference on Computer Vision, Kerkyra, Greece, 1999, pp. 1150-1157 vol.2, doi: 10.1109/ICCV.1999.790410.

[6] F. Sung, Y. Yang, L. Zhang, T. Xiang, P.H.S. Torr, T.M. Hospedales, Learning to Compare: Relation Network for Few-Shot Learning, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 1199-1208.

[7] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, G. Hullender, Learning to Rank using Gradient Descent, Learning to rank using gradient descent. In Proceedings of the 22nd international conference on Machine learning (ICML '05). Association for Computing Machinery, New York, NY, USA, 89–96. DOI:https://doi.org/10.1145/1102351.1102363.

[8] I. Melekhov, J. Kannala, E. Rahtu, Siamese Network Features for Image Matching, 2016 23rd International Conference on Pattern Recognition (ICPR), Cancun, 2016, pp. 378-383, doi: 10.1109/ICPR.2016.7899663.

[9] R. Hadsell, S. Chopra, Y. LeCun, Dimensionality Reduction by Learning an Invariant Mapping, 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), New York, NY, USA, 2006, pp. 1735-1742, doi: 10.1109/CVPR.2006.100.

[10] F. Schroff, D. Kalenichenko, J. Philbin, FaceNet: A Unified Embedding for Face Recognition and Clustering, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 815-823.

[11] AWS F1 Insights (https://aws.amazon.com/f1/)

[12] C. Zhang, S. Bengio, M. Hardt, B. Recht, O. Vinyals, Understanding deep learning requires rethinking generalization, arXiv:1611.03530v2.

[13] K. Sokol, P. Flach, Explainability Fact Sheets: A Framework for Systematic Assessment of Explainable Approaches, In Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency (FAT* '20). Association for Computing Machinery, New York, NY, USA, 56–67. DOI: https://doi.org/10.1145/3351095.3372870.

[14] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik and A. Swami, The Limitations of Deep Learning in Adversarial Settings, *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, Saarbrucken, 2016, pp. 372-387, DOI: 10.1109/EuroSP.2016.36.

[15] K. Grosse, N. Papernot, P. Manoharan, M. Backes, P. McDaniel, Adversarial Perturbations Against Deep Neural Networks for Malware Classification, arXiv:1606.04435

[16] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, A. Torralba, Learning Deep Features for Discriminative Localization, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2921-2929.

[17] R. R. Selvaraju, M. Cogswell, A. D, R. V, D. Parikh, D. Batra, Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization, Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017, pp. 618-626.

[18] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, B. Kim, Sanity Checks for Saliency Maps, Advances in Neural Information Processing Systems 31 (NIPS 2018).

[19] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, arXiv:1312.6199v4.

[20] A. Ignatiev, N. Narodytska, J. Marques-Silva, On relating explanation and adversarial examples, Advances in Neural Information Processing Systems 32 (NIPS 2019).

[21] W. Woods, J. Chen, C. Teuscher, Adversarial explanations for understanding image classification decisions and improved neural network robustness, Nat Mach Intell 1, 508–516 (2019). https://doi.org/10.1038/s42256-019-0104-6.

[22] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, S. Susstrunk, SLIC Superpixels Compared to State-of-the-art Superpixel Methods, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 11, pp. 2274-2282, Nov. 2012, doi: 10.1109/TPAMI.2012.120.

[23] L. V. Utkin, M. S. Kovalev, E. M. Kasimov, An explanation method for Siamese neural networks, arXiv:1911.07702v1.

[24] UTKFace dataset, https://susanqq.github.io/UTKFace/

[25] Y. Shen, J. Gu, X. Tang, B. Zhou, Interpreting the Latent Space of GANs for Semantic Face Editing, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 9243-9252.