UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

MASTER IN INNOVATION AND RESEARCH IN INFORMATICS

SPECIALIZATION IN DATA SCIENCE

On Explainability of Deep Neural Networks

Author: Álvaro Parafita Martínez

Advisor: Dr. Jordi Vitrià Marca
Affiliation: Mathematics and Computer Science Department, UB

Tutor: Dr. Lluís Belanche Muñoz
Affiliation: Computer Science Department, UPC

Barcelona, June 15, 2018

Abstract

Deep Neural Networks have attained state-of-the-art performance in a wide variety of Machine Learning fields during recent years, including Computer Vision and Natural Language Processing. However, it is still hard to ascertain what function each layer or neuron performs, or the reasoning behind each prediction. These models are normally considered black-box models due to the huge amount of parameters and operations they perform, too complex to be completely understood by a human supervisor.

This might seem irrelevant in contrast with the unprecedented leap in performance we have gained as a result, but for some applications this requirement for interpretability is unavoidable. Sensitive fields, like medical image recognition or credit risk scoring, require user trust on the model, that cannot be satisfied with just some performance metrics. The reasoning behind a prediction or additional information that motivates a particular decision is necessary for an application in these fields. Additionally, new legislation such as the new European GDPR demands the "right to explanation", a requirement of interpretable models for decision processes that might directly affect any person.

Interpretability encompasses many different techniques that we will discuss in the following pages, but two of them have received most of the attention by researchers, especially in Computer Vision applications. Feature visualization, on the one hand, tries to explain what each neuron or convolution channel is trying to detect. This is accomplished by automatically generating an image that contains the feature of interest for a certain objective. Attribution methods, on the other hand, look at any specific sample and highlight which components (pixels in the case of an image) are the most relevant in terms of the prediction for that input.

This project discusses both approaches, but focuses especially on attribution techniques used on Convolutional Neural Network image classifiers. After an extensive overview on the state of the art of both, we will discuss a novel method, that we called Path-mask attribution. The objective of this method is to answer the following attribution question: for an image classifier, what features in the image provide positive or negative evidence in the prediction of a certain class, with respect to another one? This comparison between two classes is essential; in contrast with most methods, that only highlight relevant features to a certain class, we compare two classes against one another, reflecting the differences between them. We show that this approach leads to more meaningful and interpretable results.

Our method is primarily based on a reconstruction of the input from a low-dimensional embedding space. This is done to ensure that the new images that we create with our method, decoded from these embedding points, belong

to the data manifold described by the original training dataset. With these embeddings, obtained with a Convolutional Variational Autoencoder, our method can optimize images directly at the embedding level, satisfying that the corresponding decoded images seem natural to the classifier network. This property is discussed extensively in the definition of the method and considered of vital importance for any interpretability strategy.

With these embedding codes, we define a second method, called Semantic Morphing, that creates an image very similar to the original one but predicted by the classifier as belonging to another class. In this way, we can compare both images to detect which features were more relevant for the change in prediction. This is accomplished by defining a transformation path between the two, obtained by decoding a path between the embeddings of both images, the original and the semantically-morphed one. Finally, the attribution method works by defining a mask, learned by an optimization procedure, that highlights which regions in the image canvas are responsible for the change of prediction along the previously defined path.

We tested this method on the MNIST and NotMNIST datasets with significant success: the mask is capable of highlighting regions that need to appear or disappear in our image in order to transform one class into the other. Additionally, we also try our approach on the FashionMNIST and CIFAR10 datasets, this time with limited results. The strengths and limitations of our method are extensively discussed along with these results to motivate further experiments and research lines that could extend its applicability to new and more complex problems. This topic and an evaluation on our research is included in the final chapter of this report.

Finally, for those readers not familiar with Convolutional Neural Networks and Variational Autoencoders, we include two appendixes with a short introduction to both of them, laying the ground work on which we develop our argument and define our own method.

Contents

Chapter 1

Introduction

At the core of any Artificial Intelligence project, or any Machine Learning project for that matter, the goal of *making stuff happen* has always been of paramount importance. We expect machines to achieve human or super-human abilities in the tasks we train them to do, and that, naturally, becomes our main focus. This normally translates in accomplishing a significant increase in a performance metric, such as accuracy in classification tasks, but some other aspects must be left behind in favour of that goal.

Things have changed in recent years. With the reawakening of Deep Learning technologies, we have finally achieved unprecedented results in a vast array of fields. This applies to Machine Learning, but also to other areas like Reinforcement Learning. Many fields, specially Computer Vision and Natural Language Processing (both in Text and Speech Processing), have finally shifted towards Deep Learning with a very significant leap in performance. Even Unsupervised Learning can benefit from advances in Autoencoders and Manifold Estimation.

We have finally *made stuff happen*, but what we are left with is a type of model that is usually considered non-interpretable and opaque: a black box. In our race for performance, we lost the interpretability of previous models, like Decision Trees. But **why is interpretability so important**?

Recent studies on ethical issues and Artificial Intelligence show that models that learn from data will learn the dataset biases, such as gender or racial prejudices. For example, in the problem of recidivism (the tendency of a convicted criminal to reoffend), the usage of non-interpretable models can hide biases (e.g. race bias) learned from the training data. Caliskan ([1]) also shows sexist associations learned by a model by working with real-world text downloaded from the Internet: ”man is to doctor, like woman is to nurse” is an example of a word association encoded in word embeddings that come as a result of training with a biased corpus. Without model interpretability, these prejudices could remain undetected. This problem raises concern among some fields, like law enforcement or credit risk assessment. Moreover, law regulations, like the European Union's new General Data Protection Regulation, will demand a ”right to explanation” for any algorithmic decision that might significantly affect users, as is the case in the aforementioned fields.

Model interpretability is also important when the model is trained in order to detect new causal relationships. Additionally, we must also consider user trust. In sensitive applications like medical diagnosis, user trust in the provided prediction is a definite requirement for the applicability of the model. Such trust can only come as a result of proper explainability of the prediction, not only by raw performance metrics.

In this context, it is clear that model interpretability is no longer a desirable secondary objective, but a definite priority in current and future predictive models.

This project will evaluate the latest advances in two of the most relevant research lines in interpretability, feature visualization and attribution, and propose a novel method for attribution that is capable of highlighting the pixels in an image responsible for the prediction of a certain label while discarding a complementary label. This method achieves significant success in detecting the most relevant features for prediction and motivates further research to extend its reach to more complex datasets and problems.

In the following sections, we will discuss what is an interpretable model and what kinds of artifacts we can use to analyze such a model. Finally, we will define both feature visualization and attribution in detail and state the objective of the project and the structure of this report.

1.1 Requirements of interpretable models

As one can see in the examples above, the implications and requirements of interpretable models are diverse. A precise definition of interpretability is missing, and even systems normally conceived as interpretable fail to fulfill some of the assumed requirements of an interpretable model.

Lipton ([2]) discusses this extensively, and provides a list of several traits of what we expect from such a model. One of them is **trust**: can we trust a system to perform the decisions it was created to make *autonomously*? Does the system correctly predict what humans are capable of predicting, or do we need a human supervisor for any possible mistakes? Trust also denotes confidence that the model will perform well with respect to our real objectives: sometimes our objective function to optimize during training is nothing but a proxy for our real objective, which is normally hard to explicit in a mathematical, optimizable expression.

The second requirement is **causality**: can we induce causal relationships hidden in the data? Researchers might use predictive models not to predict a precise outcome, but to uncover new associations in the data that can be later tested experimentally for causality. With opaque models, this task cannot be fulfilled.

We can also consider **transferability**: does the model behave appropriately when deployed in a real-world setting? We normally train and test our models with data sampled from the same distribution, but we need to consider evolving environments, or environments susceptible to the actual actions our model makes. In these contexts, our model could behave poorly, and without proper explanation of its behaviour, its performance is hard to ascertain.

In this sense one also needs to think of adversarial input. As pointed out by Szegedy in [3], Neural Networks normally learn highly discontinuous patterns in the frontiers of the manifold of study. These patterns could be exploited to force any wrong prediction with minimal changes in the input that are normally undetected by the human eye. Without proper explanation capabilities, adversarial input can pass as legitimate predictions.

Another important feature of interpretable models would be **informativeness**. Does the model provide additional information concerning a predicted output? An example of the type of information that a model could additionally provide is similar examples to the currently studied input that justify the final output of the prediction.

In the case of recidivism discussed before, the sole output of the system (will the offender reoffend?) does not provide any information apart from a prediction, that could always be wrong, even if by a low probability. In these cases it might be more interesting to obtain actual explanations of these predictions, which could provide real arguments for judges deciding on who to release.

This last case also applies to the last requirement, **fair and ethical decision-making**: by assessment of interpretations of the decision-making process, we could ensure that automatic decisions conform to ethical standards. Otherwise, bias hidden in the training dataset could be learned by our models.

1.2 Types of interpretability

Lipton also recognizes in [2] two main categories of interpretability objectives. The first one is **transparency**, which relates to the understanding of the inner workings of the studied model. We can study transparency at the level of individual components (individual neurons and its parameters), at the level of the entire model (a simultaneous understanding of the whole decision-making process) and at the level of the training algorithm (does it converge? does it respond well to new datasets?).

In this study, however, we will focus on the second category: **post-hoc interpretations**. These interpretations try to obtain additional information of the model's inner workings, even if we do not understand it completely. Examples of these are text explanations (verbal justifications of the decision), visualization (what type of input does each neuron respond to? what is the underlying structure of the input recovered by the model?), explanations by example (which examples are most similar to our current input, where similarity can be computed based on the layer activations in the network) and local explanations (which local changes affect the predicted output the most?).

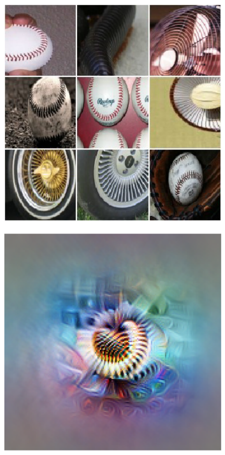
As stated before, the objective of this work was to evaluate the state of the art on feature visualization and attribution techniques. This is what the next sections will cover.

1.2.1 Feature visualization

In line with our objective of understanding the inner workings of our models, we could consider the following situation. Most neural networks nowadays work in the sequential paradigm. This means that the network is actually a sequence of layers, each dependent only on the outputs of the previous layer. The network sequentially computes hidden representations of the original input until we can finally obtain the desired output as the result of the last layer. Therefore, an option to understand the types of features each layer is capable of discovering is to perform what is called **feature visualization**.

The main idea is to obtain *interpretable* examples of inputs that activate the most each of the neurons of a layer (or channels in Convolutional layers). In that way, we can see what types of structures the layer has learned to recognize and what is their importance by analyzing the weights corresponding to each of these neurons (channels).

For example, if we have a neural network trained for object recognition in colored images, we might want to analyze what each channel in a convolutional layer has learned. Which is the type of input that mostly activates each channel?


(a) Stripes feature

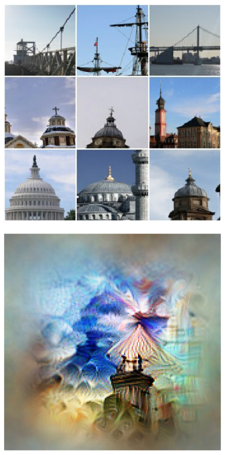

(b) Building (sky?) feature

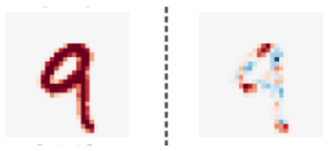
Figure 1.1: Examples of channel visualizations, from [4]

In order to do that, we could look in the dataset for the examples that mostly activate that channel. This is what the images in the top of figure 1.1 show. By analyzing the examples retrieved for the channel in 1.1a, we might think at first that the channel is sensitive to baseballs, but some of the examples show tires. We might induce then that it is actually recognizing stripe patterns. In 1.1b, on the other hand, we might assume that the channel looks for huge buildings, but it could also be possible that it is searching for clean blue skies.

As we can see here, these types of interpretations are absolutely dependent on the diversity of examples related to that particular feature that we are studying. We might assume that the channel is looking for a particular feature but in fact it activates with correlated stimuli. For example, eucalyptus leaves in the background of a koala bear image. In that case, with limited datasets, we might make wrong assumptions if we only have eucalyptus in images that contain koalas.

It is due to this problem that the approach for feature visualization does not rely on dataset examples. Instead, the common method is to take a white-noise image, pass it through the network, and optimize its pixels in order to maximize the channel activation that we are studying. In that way, and using proper regularization ([4]), we end up with images like the bottom ones in the previous figures. There, we can see that the left channel does indeed look for striped patterns. The right channel, on the other hand, is still ambiguous between buildings and the blue sky.

In any case, with perfected techniques in this aspect, one might understand what the network learns at each layer and try to obtain an actual explanation


(a) Saliency map example

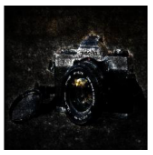

(b) Attribution by transparency

Figure 1.2: Examples of attribution visualizations, from [5] and [6]

of the final output based on the interpretations that we make of these feature visualizations.

1.2.2 Attribution

Attribution techniques try to answer the following question: what parts of the input have most affected the current prediction? In the previous example of object recognition in images, what regions in the image carry the most information to make the current prediction?

This particular case is normally visualized by what is called **saliency maps**, heatmaps plotted on top of an image with some transparency that highlights in red and blue the most relevant parts of the image, pixel by pixel. In red, we would find pixels that contribute as positive evidence, in favour of the prediction we are studying, while blue pixels signal negative evidence. An example of such a visualization can be seen in 1.2a. Another option is to take just positive attributions and plot them as transparency levels of the image; in that case, the unimportant parts of the image would be hidden by the attribution mask. Figure 1.2b shows that the relevant parts of this particular camera image are the top of the camera and the shape of the lens.

Attribution techniques, in contrast with feature visualization, opt for a more direct approach to interpretability, in the sense that the objective is to highlight parts of the input that are relevant for the prediction. In a way, proper attribution can provide a form of explanation of each specific prediction.

It is important to recognize that these techniques are not meant to be used in isolation, but as mere tools for the inspection of our predictions and models. Olah discusses in depth how we could use them as building blocks of potential interpretability systems in [7].

1.3 Project goals and report organization

We end this introduction with a short overview of the project and a quick summary of the report structure.

We will work on Computer Vision problems. Although Neural Networks are applicable to a vast array of fields, we opt for this constraint in order to focus on the objective of interpretability. As such, the proposed method will only work on Computer Vision problems, but its core ideas could be studied for extension to other fields in future projects.

Although we will cover both feature visualization and attribution, this project focuses more on attribution due to the limited results in the former and the more direct approach to prediction explanation of the latter. As such, we limit the exposition of feature visualization to a discussion on current techniques in the field. Then, we will move on to the state of the art of attribution methods and to the explanation of our own attribution technique.

Now, with the project in mind, we will discuss briefly the structure of this report.

Chapter 2 will discuss the state of the art in feature visualization and, specially, in attribution techniques, both in its possibilities and limitations. We will then move on to explain our proposal for a novel attribution technique: **path-mask attribution based on semantic morphing**.

In order to do that, we will first discuss the methodology of our work in chapter 3 and the actual method in chapter 4. Finally, we will analyze its results in chapter 5 and discuss its advantages, shortcomings and possible future experiments and improvements in chapter 6.

Additionally, we include two appendices with a short overview over Convolutional Neural Networks (A) and Variational Autoencoders (B). We also leave a final appendix (C) that details all model architectures defined in this project. For those readers not familiar with Convolutional layers or Variational Autoencoders, the reading of appendixes A and B is recommended before continuing with the following chapters.

Chapter 2

Related Work

In this chapter we will overview the state of the art in feature visualization and attribution techniques. We will start with a short overview on feature visualization and continue with a more extensive discussion on attribution methods.

2.1 Feature visualization

In the introduction of this report we briefly discussed what feature visualization consists of. The main objective is to understand what type of input or feature is activating the most a certain neuron in the network. This section will try to clarify the precise meaning of this statement. Most of the ideas summarized in this section are covered in more detail in [4].

Firstly, what do we mean by ”activate the most a certain neuron”? A neural network is essentially a network of neurons, linear or non-linear simple functions that take a series of numerical values as input and outputs a single value as a result. Therefore, maximally activating a neuron means finding the network input that, when passed through all the neurons in the network, results in the highest possible value for the output of that particular neuron that we are studying.

We should also briefly describe what a sequential model is. A sequential model is composed of an ordered series of layers. The defining property of a layer is that all the neurons belonging to that layer share the same set of inputs. In the sequential model, this *layer input* comes, actually, from the outputs of the neurons of a *previous* layer. That is why we call these types of models **sequential models**. What we end up with is a sequence of layers that take as input the output of the previous layer and output a result that is taken by the following layer. The input of the first layer is actually the network input (in Computer Vision problems, for example, the image we are studying) and the output of the last layer is the network output (in a image classification problem, for example, it could be the class logits for that particular image).

Since the hidden layers in a sequential model lose the actual input the network had received, it seems natural to think that the outputs of intermediate layers (the *hidden representations*) actually contain encoded features of the original input. Therefore, if we find inputs that maximize the output values of specific neurons, we might find what type of feature each neuron is looking for. That is the main assumption of feature visualization.

2.1.1 Feature visualization by optimization

In the introduction we stated that looking for specific examples in the dataset that maximize the activations of neurons has its drawbacks. The network might be looking for correlated stimuli (eucalyptus leaves in koala pictures) instead of the feature that we expect that neuron to look for. If the dataset has no examples of that correlated stimuli without the other confounding element, we might get a wrong interpretation of that neuron's feature.

The alternative to this process is to visualize by optimization. What this means it that, since we are looking for an input that maximizes a specific numerical quantity (the output of a neuron), we might as well optimize that quantity with respect to the input. That means using any optimization method that maximizes that quantity by modifying the input of the network. This idea was first proposed by Erhan ([8]), and named **Activation Maximization**.

The main strength of this approach is that we can take advantage of the actual network optimization process based on gradient computation by Backpropagation. In that sense, it only requires minor tweaks in most Deep Learning frameworks to compute these feature visualizations.

Now, the question that remains is, what do we actually optimize? There are several possible objectives for our optimization. In the case of image classification, for example using the ImageNet dataset with a Convolutional Neural Network, we could study feature visualization applied to any of these activations (see figure 2.1):

- **Class probability**: normally the softmax of the network outputs (the class logits). By optimizing the class probability of the particular class that we want to visualize, we obtain an example of what the network expects of that class.

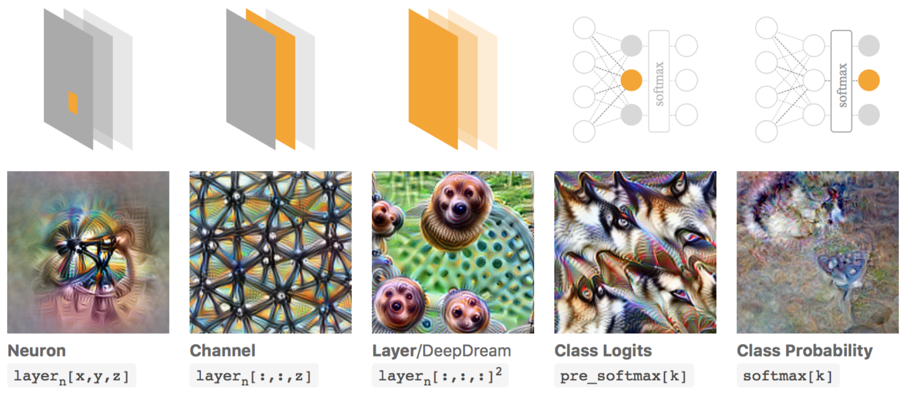


Figure 2.1: Optimization objectives for feature visualization, from [4]

- **Class logits**: in this case, instead of optimizing the probability of a class, we optimize its logit score (the value before the application of the softmax function). We might expect the same result as with class probabilities, but in fact, this objective usually performs better because one can increase the class probability of a given input by reducing the probability of all the other classes, instead of providing the features that make the particular class of study possible. That is why feature visualization research focuses on class logits, instead of class probabilities.
- **Neuron**: we can study any particular neuron in the network. However, this objective is so specific that it normally does not carry any relevant or interpretable meaning.
- **Channel**: any convolutional layer returns as output several *channels*, each containing a filtered version of our original image. By optimizing each of these channels, we generally uncover interpretable visual features (edges, patterns or even parts of objects).
- **Layers**: we can also feed an image to the network and then ask any of the layers to increase whatever that layer had detected in the original image, by optimizing the activations of the whole layer.

2.1.2 Adversarial input

Now that we know what objectives we can optimize, we choose what is the starting point for our optimization process. Normally, except for the layer objective described earlier, this initial input image is just a white noise image. If we now run the optimization method of choice, the result is not what we might expect.

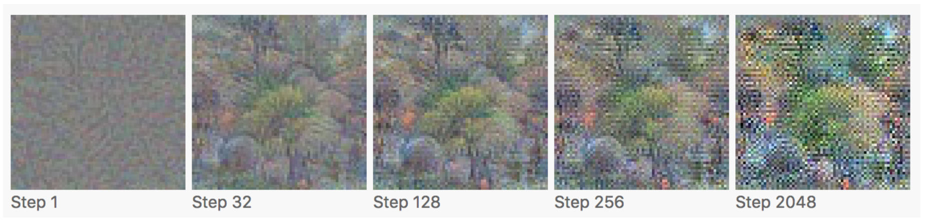


Figure 2.2: Example of optimization results, from [4]

Figure 2.2 shows the results of such an optimization. The problem with this image, although we might start seeing a hint of what we expected, is that it contains many high-frequency patterns that do not carry any meaningful information about what we want to visualize, but trick the network into increasing the objective value. This phenomenon, brought to light in [3] by Szegedy, is common in neural networks and it has been named **Adversarial examples**.

Adversarial examples are basically inputs specifically tweaked to maximize any given activation of a network by changing in ”unnatural ways” any initial input. This idea of unnaturality is important: the network has been trained on inputs sampled from a specific distribution. We might conceive this distribution as defining a manifold in a high-dimensional space, and the network is only adjusted to respond well in that manifold. As a result, if we optimize anything inside the network's activations without taking into account that manifold, we are bound to get wrong predictions on seemingly natural inputs, because we have unknowingly stepped out of the manifold.

This is the reason why regularization techniques are required. Here regularization must be understood as forcing the input to remain inside the *distribution of natural inputs*. There are several methods of regularization for feature visualization; we will briefly discuss them in the remaining of this section.

2.1.3 Regularization techniques for feature visualization

Olah ([4]) distinguishes three families of regularization techniques applied to feature visualization.

- **Frequency penalization**: directly penalizing high-frequency patterns, either by re-normalizing the input to the expected class norm at each step in the optimization (as proposed by Erhan in [8]), or by adding a Total Variation (TV) loss to the objective function of the optimization (suggested by Mahendran in [9]). This TV loss penalizes high differences in intensity between neighboring pixels.

  Alternatively, one can also use blurring filters between every optimization

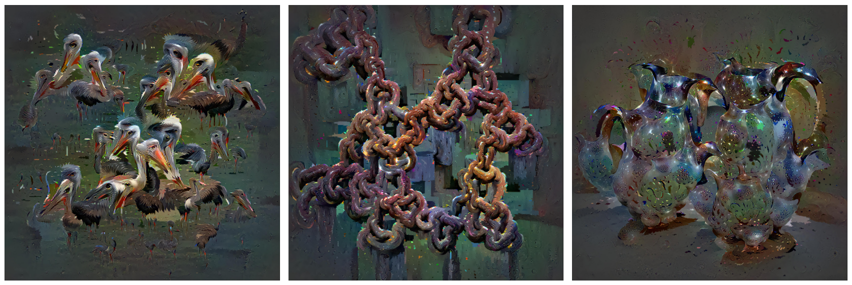


Figure 2.3: Feature visualization examples, from [10]

step, either with a Gaussian filter or, better, with Bilateral filters, in order to preserve edges in the image.

- **Transformation robustness**: slightly transforming the image between optimization steps with translations, rotations, scaling, color jittering, etc. These alterations try to ensure that the image generated at the end of the optimization process is not affected by minor transformations common in images, which effectively removes patterns that are only dependent on the network structure without conveying any real meaning. Examples of feature visualizations created with these techniques can be seen in [10], and figure 2.3 displays three of those results.
- **Learned priors**: the final option is to optimize while enforcing a specific prior distribution for the inputs. This can be done with a Variational Autoencoder (VAE) or a Generative Adversarial Network (GAN), that transforms an input embedding, a low-dimensional continuous code, into an image that belongs to the manifold extracted from the training data. Such an approach can be found in [11], with promising results.

These forms of regularization have led the field of Feature Visualization to better and promising results, but they are still subject to lots of tweaking: finding specific transformations or specific values for its hyperparameters. As such, the need for less hyperparameter dependency is paramount, lest we need to readjust every regularization step for each new dataset.

We now end this overview on feature visualization to focus on the most important part of our research: attribution techniques.

2.2 Attribution methods

The main objective in attribution methods is to distribute prediction relevance to every pixel in the input image. That means that, theoretically, without the knowledge of the most important pixels (according to the attribution score), the network would not be able to predict the correct class of the image.

One must note that, while feature visualization methods are used to study the network itself without needing any input in particular, attribution methods focus on explaining a specific input. Attribution methods are also called saliency methods, since they return what is called a *saliency map*, a map between an image and numerical scores for each of its pixels.

In this section, we will overview some of the latest works on attribution and discuss their main contributions and drawbacks.

2.2.1 Gradient-based methods

One of the first and most basic ideas in terms of attribution is the use of gradients. We will start with some simple definitions to justify this method.

Any Neural Network can be represented as a continuous function, differentiable almost everywhere. This means that the set of points where the function is not differentiable has measure 0. This can be easily proved by seeing that almost all types of layers in Neural Networks nowadays are all continuous and differentiable, or at least differentiable almost everywhere, as is the case of the ReLU activation layer, which is not differentiable just at $x = 0$. In cases where that is not true, researchers usually tweak the functionality of the layer so that its gradient becomes differentiable almost everywhere. Otherwise, training methods based on gradient descent would not be possible.

Being that the case, we can define the network just as a function

$$F\colon \mathbb{R}^{d_i} \to \mathbb{R}^{d_o}$$

where d_i is the dimension of the input (for example, the number of pixels in a grayscale image) and d_o is the dimension of the output (for example, the number of classes in a classification problem).

Now, let's consider instead that function F applied to a particular class j: F_j. That means $F_j(X_0)$ gives us the logit score of class j for an input image X_0. Note that by the first-order Taylor expansion of F_j at point X_0:

$$F_j(X) \approx F_j(X_0) + \nabla F_j(X_0)(X - X_0)$$

Now, if X_0 is the image of study, note that the gradient of F_j at X_0 gives us an attribution on each pixel of X_0. What are the pixels that, changing them

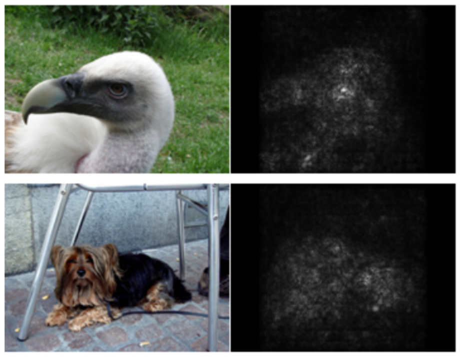


Figure 2.4: Attribution by gradient examples, from [12]

the least, produce the highest impact on $F(X)$? In other words, we can analyze the importance of each pixel as the impact of their changes in a neighborhood of the actual image.

This idea, put forward in [12], gives us an intuition on the importance of gradients in attribution. Moreover, the connection between Deconvolutional Networks, another visualization technique, and gradients has also been proved in [12] too.

Figure 2.4 shows saliency maps of two images computed by taking the gradients of the logic scores of the corresponding class with respect to each color-pixel (one gradient for each color channel), computing their absolute values and taking the maximum across each color channel, for each pixel. We can see that the object of interest contains most of the important pixels.

This method, however, does have some limitations. For starters, the relevance distribution is too noisy and doesn't really have much information about which parts in particular were important for the prediction. Additionally, we will discuss one particular limitation when talking about Counterfactual images in section 2.2.3.

As a final note, it is quite common, in terms of visualization, to multiply the obtained gradient with the input. As a result, attributions become sharper. This is usually referred to as *Gradient * Input* or GI.

2.2.2 SmoothGrad

Gradient techniques seem powerful in their simplicity, but their results are usually very noisy and, more often that not, not that informative. SmoothGrad, introduced by Smilkov in [13], briefly discusses the causes for this phenomenon

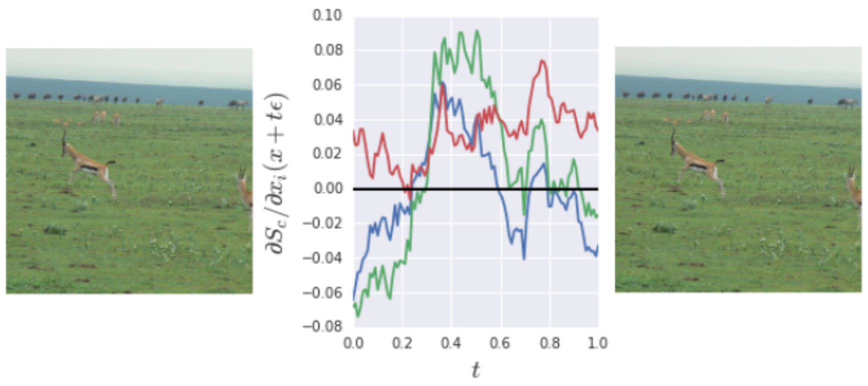


Figure 2.5: Gradient fluctuation experiment, from [13]

and proposes a method to mitigate this effect.

Smilkov argues that the reason for the noise found in saliency maps derived from gradients comes as a result of sharp fluctuations of the network function F in small scales. Nothing in the training procedure of Neural Networks imposes that gradients of the learned function should be smooth. In fact, Smilkov gives an example on gradient fluctuation by computing the gradients along a line segment defined as

$$x(t) = x_0 + t\epsilon$$

where x_0 is the image of study, ϵ is a random, small perturbation in the image, and $t \in [0, 1]$. Figure 2.5 is an example of this experiment. By taking any picture (left), and performing the described perturbation for several t's, we can compute the gradient of any pixel in the image. We take a pixel at random, and plot (center) the gradient wrt. that pixel for the red, green and blue channels. Finally, as a check on the effect of perturbation ϵ, the right picture is $x(t)$ for $t = 1$. One can see that such a small perturbation, with no apparent changes in the image, does change the gradient greatly.

Due to this behaviour, Smilkov argues that the actual gradient in the image is actually less informative than a Gaussian smoothing on the gradient. However, since computing this smoothing in such a high-dimensional space is intractable, we can use instead a stochastic approximation.

What we do instead is compute the gradient attributions for several images sampled from the distribution

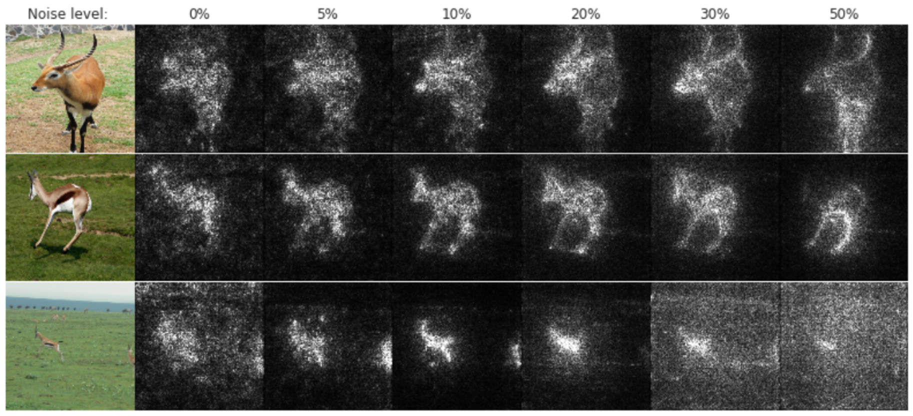


Figure 2.6: Smoothgrad examples, from [13]

$$X \sim img + \mathcal{N}(0, \sigma^2 I)$$

and finally compute the average of these gradients. As a result, what we do is actually smooth over the gradient distribution, and the resulting attribution maps do improve noticeably. Figure 2.6 shows the impact of σ on gradient-based attribution maps. The noise level reported on top of the images corresponds to $\sigma/(x_{max} - x_{min})$.

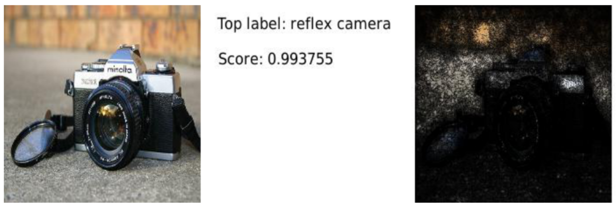
2.2.3 Integrated Gradients

Sundararajan continues building on top of gradient methods and describes an interesting phenomenon in [14].

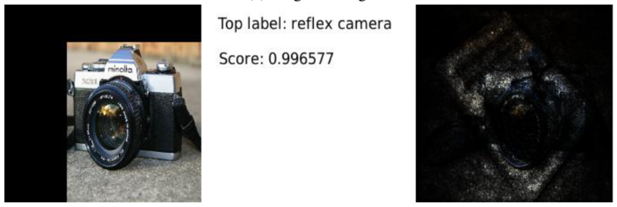
Counterfactual images

The top image on figure 2.7 shows the gradients of this image of a camera. One might think that for the network, these pixels are informative of the class of study. To test this hypothesis, a common option is to perform what is called **ablation**: replacing the supposedly important pixels by a constant color (normally black). Then, we can compute the classification score for the ablated picture to evaluate the effect of this modification. We see that not only it does not decrease, but it increases. Additionally, the new gradients seem to better focus on the camera.

This simple experiment seems to suggest that pure gradients, at least in this



(a) Original image.



(b) Ablated image.

Figure 2.7: Issues with the gradient, from [14]

image, do not carry the informative value that we expect. The main hypothesis for this behaviour argued in [13] is that the function flattens in the vicinity of this input (image). As such, the gradient wrt. the input in this vicinity would be tiny. This phenomenon is referred to as **saturation**.

In order to test this hypothesis, we can define the set of **Counterfactual images**:

$$\{\alpha \cdot img \mid \alpha \in [0,1]\}$$

What this does is to reduce the intensity of the pixels in the image, blackening them. We now run the following experiment (figure 2.8): if we compute the softmax (left) and pre-softmax (right) scores of the top scoring label along this interpolation path, we can see that at some point in $\alpha \in [0,1]$ the network saturates.

Even the hidden layer activations saturate too: in figure 2.9, we can see the L_2 distance and cosine distance, respectively, of the activations of the Counterfactual image (depending on α) and the real image along four layers of the Inception neural network, for 30 different images. As we see in the plots, all layers end up saturating at a certain α.

This observation leads to the following hypothesis: even if the network saturates, the gradient of important features is not saturated early in the training process, but after the model has been completely trained. As such, one can expect that, outside of the decision boundary, these gradients will still be informative.

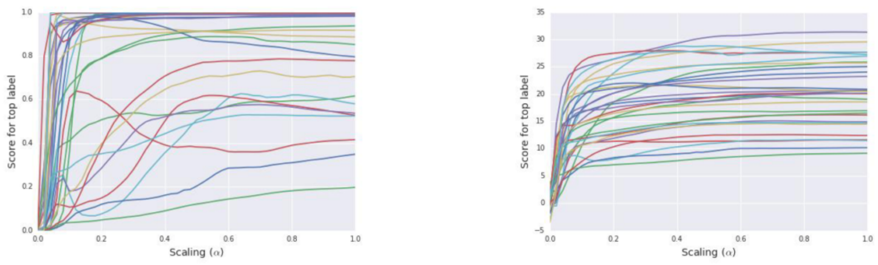


Figure 2.8: Saturation along Counterfactuals set, from [14]

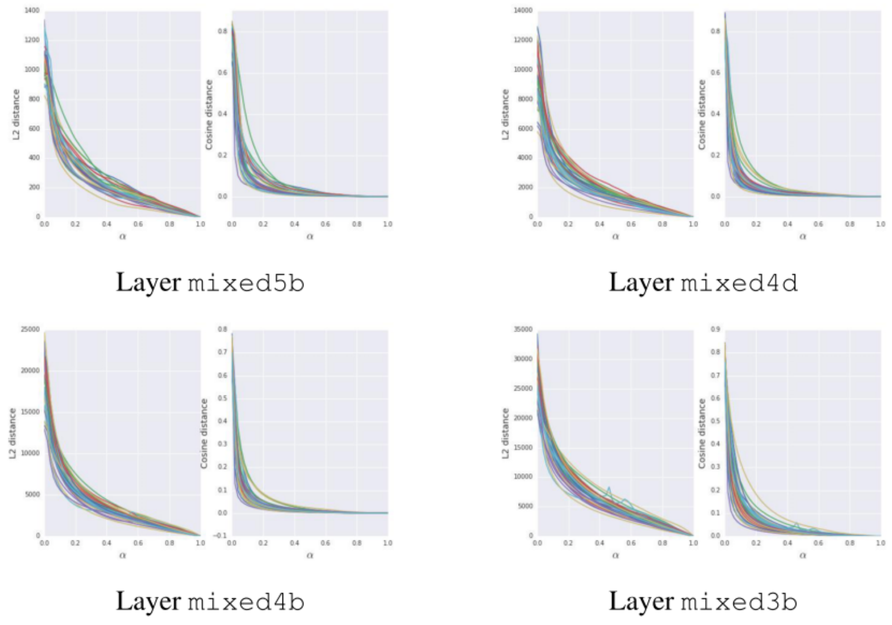


Figure 2.9: Saturation in hidden layers along Counterfactuals set, from [14]

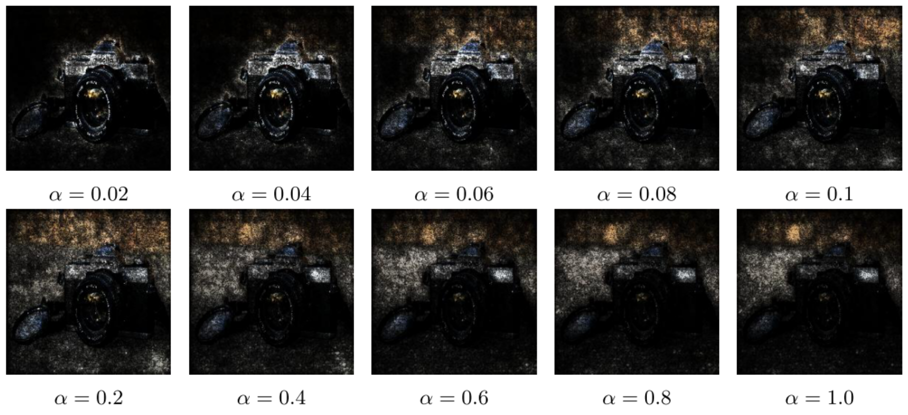


Figure 2.10: Saturation in hidden layers along the Counterfactuals set, from [14]

Smilkov ([13]) then suggests to compute the gradient along the path defined by the Counterfactual images. As stated above, we expect the important features to appear at some point in $\alpha \in [0, 1]$. Figure 2.10 shows the gradient saliency maps along this path for several values of alpha, used as transparency levels on top of the previous camera image. Values $\alpha < 0.6$ apparently give better results than the original gradients ($\alpha = 1$).

Integrated Gradients

Now that we know that Counterfactual images carry relevant information, we can introduce the Integrated Gradients method. [13] argues that by aggregating the gradients computed along this interpolation, we can define a proper attribution mask that takes advantage of the features detected by the Counterfactual images while also fulfilling an interesting property: **Completeness**.

Integrated gradients consists of computing the following integral:

$$IntegratedGradients_i(x) := \int_{\alpha=0}^{1} \frac{\partial F(\gamma(\alpha))}{\partial \gamma_i(\alpha)} \frac{\partial \gamma_i(\alpha)}{\partial \alpha} d\alpha$$

which returns the attribution of pixel i along interpolation path γ for image x. Here $\gamma_i(\alpha)$ represents the value of pixel i at step α. This integral, however, is not computed directly, but estimated by discrete steps:

$$IntegratedGradients_i^{approx}(x) := \sum_{k=1}^{m} \frac{\partial F(\gamma(k/m))}{\partial \gamma_i(\alpha)} (\gamma(k/m) - \gamma((k-1)/m))$$

which, in turn, for γ a linear interpolation path with equidistant steps ($\gamma_{x_0,x}(\alpha) := (1-\alpha) \cdot x_0 + \alpha \cdot x = x_0 + \alpha \cdot (x - x_0)$, for $\alpha \in [0,1]$) becomes:

$$IntegratedGradients_i^{approx}(x) := \frac{(x - x_0)}{m} \sum_{k=1}^{m} \frac{\partial F(x_0 + \alpha(x - x_0))}{\partial \gamma_i(\alpha)}$$

for $x_0 = 0$, a completely black image. x_0 is usually referred to as the baseline and used in other attribution methods, where it is not necessarily $x_0 = 0$. Nevertheless, the choice of this baseline is the standard.

Note that Integrated Gradients can be summarized as the mean of gradients along the Counterfactuals set, multiplied by the difference between the image and the baseline. Since the baseline in this case is 0, it actually is just multiplied by the original image. In a way, it resembles the previously described Gradient * Input method.

Properties of Integrated Gradients

As stated before, Integrated Gradients (IG) fulfill an interesting property called Completeness. This property means that the sum through all dimensions of the input (in the case of images, over every pixel) of the attributions obtained through IG (using the integral expressions) sums exactly to the difference between the prediction of x and the baseline x_0. In other words,

$$\sum_i IntegratedGradients_i(x, x_0, \gamma) = F(x) - F(x_0)$$

Naturally, with the approximated integral this property only holds approximately.

The rationale behind this integral is thus to compute attributions that fulfill this property. Additionally, Sundararajan ([6]) proves that it also fulfills **Implementation Invariance** and **Sensitivity**. Implementation Invariance means that the attribution method run over any network fulfilling the same function as F (same relationship input-output), no matter what actual operations the function performs "inside the box", would obtain the same result.

Sensitivity, on the other hand, means that any variable to which the function does not depend on will get zero attribution. These properties are all desirable for an attribution method and, thus, justify the use of Integrated Gradients.

In any case, the results of this method are limited, since they normally focus on edge artifacts and also tend to be noisy.

2.2.4 Layer-wise methods

We will now leave gradient-based methods and talk about another family of attribution techniques. This new approach could be called Layer-wise Propagation methods, and encompasses several methods, the first of which is Layer-wise Relevance Propagation (LRP). The general idea of this approach, proposed in [15], is to assign a relevance score at the top of the network (the output): 0s everywhere except for the dimension corresponding to the class that we are studying, where we will leave the original output of the image we are studying. Then, that ”relevance” will propagate, layer by layer, back to the input. The resulting relevance scores at the input level will conform the desired saliency map.

A general desirable property in these methods is that the sum of relevance scores at any given layer should always be equal to the relevance at the top of the network. This means that the relevance scores at the input level can be understood as signals of the effect of each input dimension (in the case of images, pixels) to the final prediction of the label to study.

The main difference between methods in this Layer-wise Propagation category is how to distribute, layer by layer, that relevance. In order to that, they assume that each layer can be reinterpreted as

$$x_j^{(l+1)} = g^{(l)}(\sum_i x_i^{(l)} w_{ij}^{(l)} + b_j^{(l)})$$

where $x_i^{(l)}$ is the i-th dimension in the input of layer l, $w^{(l)}$ are the corresponding weights for that layer, $b^{(l)}$ is the corresponding bias parameter, $g^{(l)}$ is the non-linearity function of that layer and $x_j^{(l+1)}$ is the corresponding j-th dimension of the activation of layer l, or, in other terms, the input of layer $l+1$. Note that linear layers, convolutional linear layers, batch normalization layers, etc., can be described in this manner. In cases where this is not possible, we can always apply a Taylor approximation of first order.

With this framework, the rule for relevance propagation in the case of LRP, is to compute the relevance of a given layer input, $R_i^{(l)}$, as the sum of the relative relevance between that layer and all activation dimensions for the next layer,

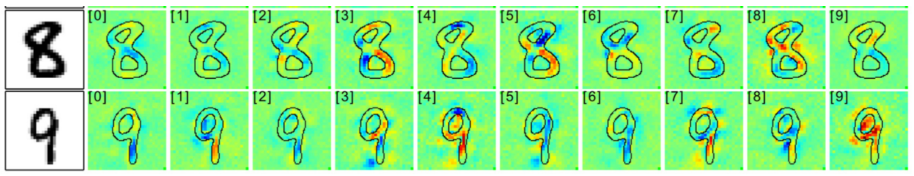


Figure 2.11: Attribution by Layer-wise Relevance Propagation, from [15]

$$R_i^{(l)} = \sum_j R_{ij}^{(l,l+1)}$$

Now, as for that relative relevance between layers we just compute a weighted average of the relevance scores in the next layer, weighted by the product between the inputs of layer l and their corresponding weights to each specific output j.

$$R_{ij}^{(l,l+1)} = \frac{x_i^{(l)} w_{ij}^{(l)}}{\sum_i x_i^{(l)} w_{ij}^{(l)} + b_j^{(l)}}$$

Note that we should also add some stabilizer ϵ in the denominator of the previous expression in order to avoid instability when that sum is close to 0, as suggested in [15].

Figure 2.11 shows examples of LRP attribution on MNIST numbers, without any stabilizer. Each column corresponds to the attribution on label i. We can see, for example, that for label 5 in the first row, we find that the top-right stroke is negative evidence of a 5 (it should not be there) while the bottom one counts as positive evidence; that part may be the most important one according to this method. The second row, on the other hand, could be confounded with a 4 as long as the top horizontal stroke disappeared.

Figure 2.12, on the other hand, shows us the results of this method with natural images. The first heatmap column corresponds to a low value ($\epsilon = 0.01$) in the aforementioned stabilizer, while the second column uses a much bigger value ($\epsilon = 100$). The results are clearly affected by this hyperparameter. In any case, we can see that the method acts somehow as an edge detector, except for the fact that only the relevant edges are finally highlighted.

Although, intuitively, the results seem to highlight the important portions of the image, it is clear that the effect of possible numerical instabilities does affect the result, as is proved with the effect of the stabilizer in the previous example. Another problem with this approach is that it does not take into account the influence of the non-linearity g, which could increase the relevance

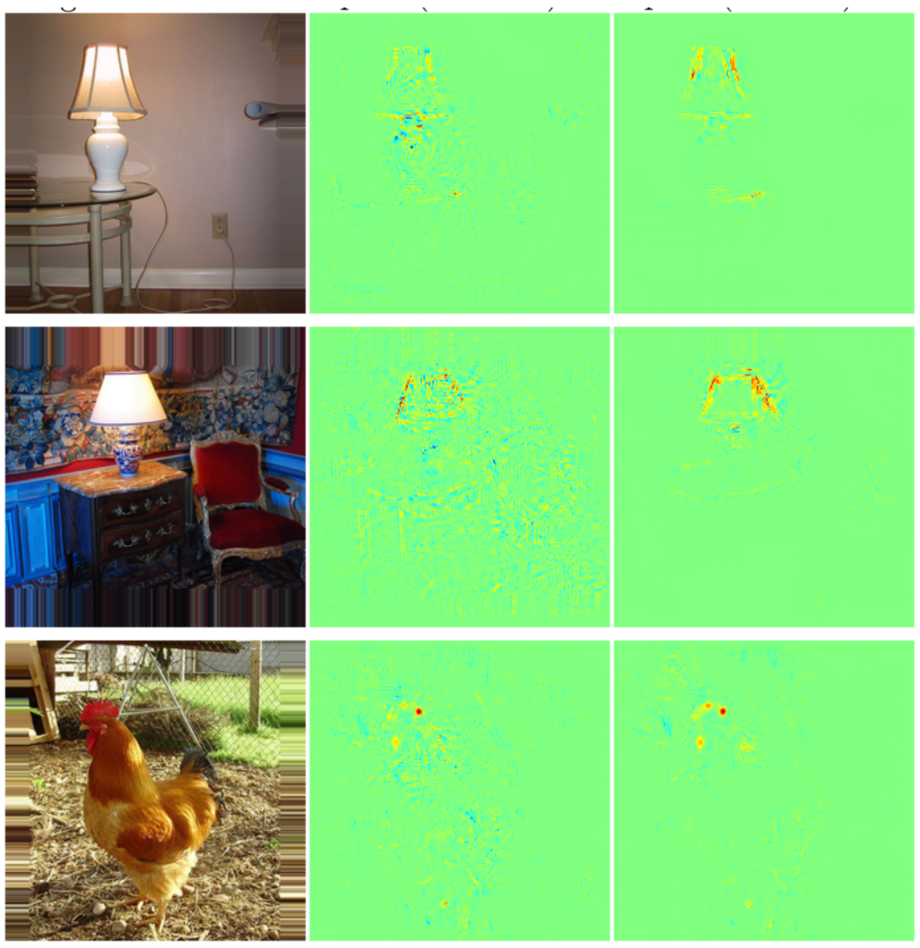


Figure 2.12: Attribution by Layer-wise Relevance Propagation, from [15]

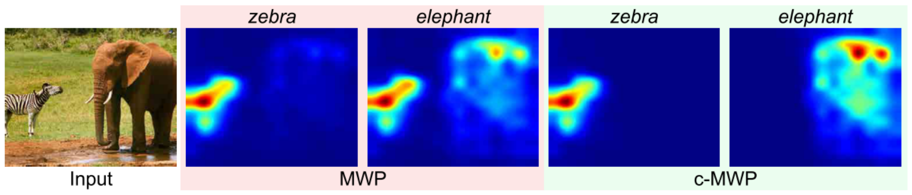


Figure 2.13: Comparison of EBP and c-EBP, from [16]

of some dimensions while reducing others, like in the sigmoid function, one the most basic non-linearities. Additionally, a problem with these layer-wise methods is that they require a specific implementation of the propagation rule for each type of layer; as such, its implementation with Deep Learning frameworks that abstract most of the details of each layer could be troublesome. In this respect, gradient-based methods are easier to apply and test.

Before finishing this section, it is worth mentioning another Layer-wise method: Excitation BackPropagation (EBP), proposed in [16]. The idea is very similar to LRP but there are two main assumptions that change the method substantially.

First, the output of every activation layer should be positive. If it is not, but it is lower-bounded, we can just shift it towards positive values for the computations in this method. Secondly, it assumes that activation neurons are tuned to detect features in the input. Additionally, their response should be positively correlated to the confidence of that detection.

With these two assumptions in mind, the method must change some details. Firstly, the bias in a layer is not considered, as it is not an input-related quantity, and we only consider non-negative weights, since those are assumed to be the ones that carry positive evidence about the class label that we are studying. Finally, the weighted average must take into account both these non-negative weights and the non-negative inputs of each layer (also assumed to be non-negative, as stated before).

The main strength of this method with respect to pure LRP is that we can extend it to what is called Contrastive Excitation Backpropagation (c-EBP), that is capable of highlighting the information in the input specifically related to the class of study, removing uninformative features by subtracting any evidence for the complementary prediction.

The comparison between EBP and c-EBP can be seen in figure 2.13. Apparently, there are some features shared by elephant and zebra that add noise to the EBP attribution for elephant. If we use c-EBP instead, we can discard these common features and focus on the ones that characterize each class.

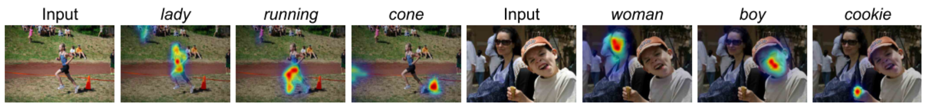


Figure 2.14: Attribution by c-EBP, from [16]

Additionally, we include in figure 2.14 other experiments with c-EBP that highlight its potential, at least, in identifying where each class is found in the image.

2.2.5 Learning methods

The previously described methods either depended on the simplicity of gradients or prescribed a series of rules to compute attribution. In the case of the following methods, the approach is to let an optimization procedure learn what a good explanation for the current input is.

Embeddings for Interpretability

This method is not really an attribution method, but it is worth mentioning because of the high interpretability of its results and its connections to attribution.

Qi proposed in [17] to create an embedding layer that can be attached to any intermediate layer in the network. However, the closer to the output, the more interpretable and meaningful the results. Note that this procedure, for the moment, is only suggested for binary classification, or 1 vs. the rest, in case of more classes.

This embedding layer is nothing but a lower-dimensional expression of the activations found on the layer that the embedding tries to replace. The objective, then, is to perform dimensionality reduction at the activations level, adding several constraints that justify their use for interpretation.

- **Prediction faithfulness**: the embedding layer will be reattached to the next layer in the network so that its values can traverse the network to the original prediction. We would like that new prediction not being very different to the original; thus, we add a prediction faithfulness loss (i.e., a simple L_2 loss) to the optimization objective.
- **Layer reconstruction**: the embedding layer should be able to reconstruct the original layer it is replacing. This reconstruction, however, does

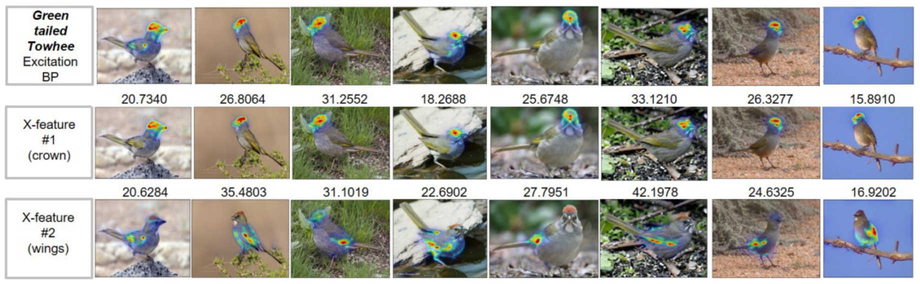


Figure 2.15: Examples of embedding explanations, from [17]

not need to be precise in every dimension. We should not waste space in the embedding to reconstruct neither correlated dimensions nor useless dimensions for the current prediction. Instead, Qi suggests adding a sparsity term that controls which dimensions in the activation of the original layer should we consider for reconstruction. By adding a regularization on this sparsity component, the network itself can decide which dimensions of the original activations to preserve.

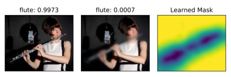
- **Embedding orthogonality**: finally, since we want to obtain interpretable dimensions in the embedding vector, we can enforce orthogonality in the embedding dimensions so that concepts can be isolated. This orthogonality can be enforced by adding an orthogonality loss to the optimization objective.

At the end of the optimization process, we have an embedding layer that should represent *orthogonal* concepts that add to the prediction of the network. By analyzing the contribution to the prediction of each of those concepts, we can find the most relevant. Finally, Qi suggests to use an attribution technique, Contrastive Excitation BackPropagation (discussed before), to see what pixels in the image were responsible for the activation of these important concepts.

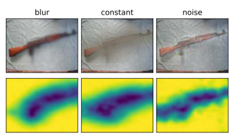
Figure 2.15 shows some examples of explanation with this technique. The first row shows the pure EBP results from the prediction, while the second and third rows show, respectively, the most important components of the embedding layer. We can see that each component in the embedding isolates a specific important feature for the prediction, while EBP shows all these features combined. This separability is indeed very important in terms of explainability.

Mask perturbation

Mask methods try to learn the saliency maps by considering them as the optimization input. We will discuss the method proposed in [18]. Consider a mask



(a) Example with blur mask



(b) Types of perturbation

Figure 2.16: Examples of attributions by mask perturbation, from [18]

m,

$$m\colon \{1, ..., W\} \times \{1, ..., H\} \rightarrow [0, 1]$$

that defines the amount of perturbation each pixel in the image should be inflicted. The procedure tries to learn the smallest perturbation (in terms of amount of perturbation suffered by the image) that affects the prediction of the image the most.

We denote by $x \odot y$ the Hadamard product (the element-wise product) between elements x and y. As a simplifying notation, we will use m both as a function and as the corresponding pixel mask matrix defined by function m interchangeably.

We now discuss the different possibilities for the aforementioned perturbation. Let $\Phi(x_0, m)$ be image x_0 transformed by mask m. The different perturbations proposed in [18] are:

- Constant value: the mask imposes a certain constant value.

  $\Phi(x_0, m) = m \odot x_0 + (1 - m) \cdot \mu_0$, where μ_0 is a constant value (normally 0).

  Note that this perturbation can perform ablation, if the mask only has values in $\{0, 1\}$.

- Noise: the mask imposes white noise.

  $\Phi(x_0, m) = m \odot x_0 + (1 - m) \odot \epsilon_0$, where ϵ_0 is a matrix formed by i.i.d. Gaussian noise, with the same dimensions as x_0.

  Note that the mask can control the amount of noise each pixel may have.

- Blur: the mask adds Gaussian blur to x_0, with $\sigma \cdot m$ as the corresponding noise for the Gaussian kernel for each pixel.

The procedure to learn this mask is a simple optimization process. The objective function is the reduction in softmax score for the class of interest added to some regularizing losses to avoid adversarial artifacts in the mask. These include the L_1 norm of the difference between the mask and a matrix of 1s (to force the mask to select a region as little as possible) and a Total Variation (TV) loss for the mask (as the one discussed in the Feature Visualization section). Fong ([18]) also uses a lower-resolution mask for the learning process, that is upsampled with a Gaussian kernel. With all these regularizations, the mask learns continuous patches of perturbation, as little as possible, that affect significantly the predictive capabilities of the model for the transformed image.

Figure 2.16a shows an example of attribution with a blur mask, while 2.16b compares the effect of different types of perturbation. In both cases, the mask learns to detect which is the object of interest and hides it from the network prediction.

Chapter 3

Methodology

In this chapter we will discuss the datasets with which we will perform our experiments and the reasons behind our evaluation strategy.

As stated before, the goal of this project was to study the state of the art on feature visualization and attribution methods, but we will now focus solely on attribution. The remaining chapters will describe, justify and evaluate our proposed attribution approach.

As a short note about our implementation, our work was done solely in Python Jupyter Notebooks using the pytorch Deep Learning framework. These notebooks are attached with this report and include all the experiments showcased here. The precise architectures of all networks used throughout this project can be found in appendix C. For those interested in replicating our experiments, the README file details the file structure of the project and which libraries to install.

3.1 Datasets

We will work with the image classification problem, using Convolutional Neural Networks. Our objective will be to define an attribution strategy for these kinds of networks.

For the sake of simplicity and the need to constrain the GPU requirements, we will not work with real-image datasets like the famous ImageNet dataset ([18]). Instead, we will focus on gray-scale, low-resolution image datasets: MNIST, notMNIST and, for the last experiments, FashionMNIST. We additionally include a last dataset, CIFAR10, to try our approach with a more

(a) MNIST examples, from [19]

(b) notMNIST examples, from [20]

complex problem, although the experiments will be limited.

MNIST

MNIST ([19]) is a dataset consisting of gray-scale images, 28x28 pixels, of arabic numbers. All numbers appear in white with a black background. This makes it a simple problem even for linear models, but it is interesting for interpretability problems because the evaluation of an attribution of numbers is easy for a human supervisor. The distinctions between the different numbers are clear to humans and we can easily interpret if our attributions highlight important features or not. Figure 3.1a shows some examples of MNIST numbers (inverted for better visualization).

As a final note, this dataset is already split in a train/test set of 60,000/10,000 samples.

notMNIST

notMNIST (from [20]) is a dataset very similar to MNIST. It consists of gray-scale images of 28x28 pixels showing latin letters from A to J (10 classes). The objective of the dataset is to extend MNIST to more complicated shapes, while preserving the structure of the original MNIST.

We replicate our method to notMNIST to evaluate its capabilities outside of the easier problem of pure MNIST. This dataset, however, is not as curated as the original MNIST: it contains many examples of non-letters and the fonts of each letter can be very different between samples.

The dataset is divided in two: the first, *small notMNIST*, consists of a small subset of the whole dataset, *large notMNIST*, with hand-cleaned samples. We will only use the former for the evaluation of our method, because the latter contains many noisy instances and even non-letters. Figure 3.1b shows some examples of notMNIST *A* letters (inverted for better visualization), where we

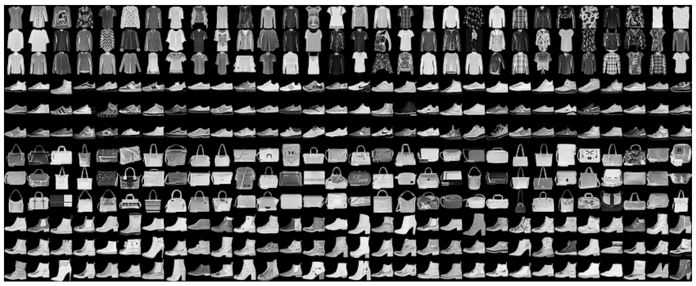


Figure 3.2: FashionMNIST examples, from [21]

can see the vast differences between samples.

FashionMNIST

FashionMNIST (from [21]) is another MNIST-like dataset, consisting of images about types of clothing, like shirts, pullovers, shoes or bags. As MNIST, its images are 28x28 gray-scale black-background, white object pictures with 10 different classes for prediction. It also consists of 60,000/10,000-sample training/test datasets.

This dataset is relevant because it is a real Computer Vision problem, more so than MNIST or notMNIST, so it is a good benchmark for our approach. Figure 3.2 shows examples of some of its categories.

CIFAR10

Finally, CIFAR10 (from [22]) is a different kind of dataset. Although it also includes a training/test set of 60,000/10,000 entries and 10 different classes (i.e., airplane, automobile, dog, cat, etc.), here the images are not gray-scale, but coloured, and they are 32x32.

We will use this dataset to evaluate our strategy on colour images and describe its shortcomings and required next steps for its extensibility to harder Computer Vision problems. Figure 3.3 shows examples of all classes in this dataset.

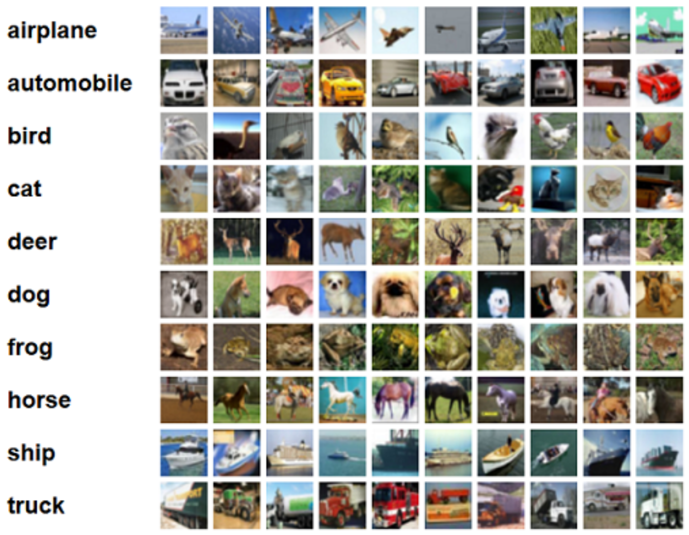


Figure 3.3: CIFAR10 examples, from [22]

3.2 Evaluation metrics

We will explain the project structure, experiments and results in the following chapters, but we will discuss first what kind of evaluation metrics are there for our purposes and the rationale behind our evaluation strategy.

Samek analyzes in [23] the problem of evaluating attribution maps in image object classification neural networks. The proposed metric tries to analyze the impact of the removal of the most important pixels (according to the analyzed attribution map) on the prediction of the label of study.

This means that, for any attribution map to which we want to compute its metric, we will sort all image pixels by their attribution scores descendingly. Then, for each pixel in that order, we ”hide” that pixel from the original picture and evaluate the prediction with our network. As a result, we get a prediction score (i.e., the softmax score for the object label) for that pixel removal. Then, we remove each following pixel sequentially, while computing the corresponding prediction score, until there are no more pixels to hide.

As a result of this process, we can plot a curve, called the **Most Relevant First** (MoRF) curve. Note that if the pixels we remove first (supposedly the most important in the picture) have a bigger impact on prediction, then the area over the MoRF curve will be bigger. Therefore, we can analyze this metric, called *Area Over the MoRF Perturbation Curve* (AOPC), to compare attribution methods and see which ones are better.

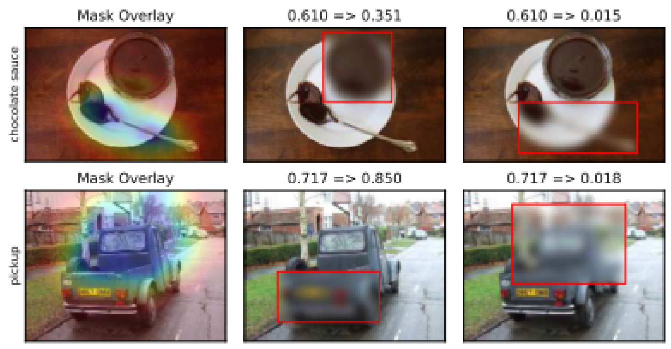


Figure 3.4: Image perturbation defined by attribution mask, from [18]

The last detail to discuss about this metric is what does ”hiding pixels” mean. Samek ([23]) suggests using any function on the pixel, or the neighborhood of the pixel, that hides the information it provides to the network. Some possibilities are using a constant value (i.e., a black pixel, what we called **ablation** before) or replace that pixel (or the neighborhood of that pixel) with noise from a uniform distribution. Ancona compares several methods in [24] with a similar ablation-based strategy called *Sensitivity-n*. Fong ([18]), on the other hand, suggested an attribution method based precisely on this idea, discussed before in section 2.2.5. Figure 3.4 shows two masks on two images and their impacts on predictive performance.

The resulting metric should theoretically help us in assessing which method is better at defining the attribution for each pixel. However, we argue that such a metric is based on a wrong assumption.

We know that Neural Networks learn from a training dataset the structure of a manifold, by means of several layers interconnected in a sequential architecture. At the end of these layers, one can use a simple classifier with the resulting activations, because the original manifold located in a bigger vector space has been projected into an easier, reduced form. These activations should then be considered as the final extracted features of this process.

We can tune our classifier to perform very well because the training and test data supposedly belongs to the same latent distribution. As such, what the network is in fact learning is how to perform its predictions in the manifold described by that distribution. However, nothing can ensure that its performance will not be affected when a point outside that manifold is studied. This is the case, in fact, of adversarial examples ([3]).

Being that the case, one must think what an image with ”hidden pixels” really is. Has the network ever seen an image with that behaviour? Not unless we had modified our original samples to behave that way during the training process. Therefore, if we had not done so, it would be plausible to think that the manifold discovered by the network should not contain our ablated images. The same applies to any kind of ”pixel hiding” that we would perform, like in the case of patches of gaussian blur, or pixel replacement by noise.

If that is the case, does evaluating the performance on these transformed images make sense? If the ablated images do not belong to the training manifold, as we think, then nothing ensures that they are not adversarial samples: the network should not be trusted with this kind of input. Therefore, we argue that AOPC, or any metric based on unnatural image transformations, should not be used.

What if we had trained our model with ablated images? This opens the discussion back, because then there would be nothing unnatural about these images. The problem, however, is whether it is even possible to do so. How to select patches of pixels that, all hidden, really affect the label of the image? If we did this randomly, we would probably not hide enough for the network to be affected and, therefore, when computing AOPC, the kinds of images the network would receive would be, again, unnatural. If we used human-transformed images, this increases the requirements of data. In any case, we do not deny the feasibility of this approach, but at the moment we will not consider it as an option.

Nevertheless, this leaves us with no alternatives. How do we confirm the good performance of our method? How do we compare which approach is the best? Our only option is by human inspection, which is purely subjective. Due to this, we risk choosing an explanation approach only because it confirms our expectations, not because it actually explains the network inner workings.

In any case, that will be our evaluation strategy. Ablation-based metrics are not reliable, and human inspection can discern between a meaningful attribution and a random one. The only constraint is that we cannot compare between different attribution methods to see which is best, but for our current objective, proposing new venues of research, this is not critical. Future work should focus on deriving proper evaluation strategies and extend our method in search for better results than the current state of the art.

Chapter 4

Proposed method

This chapter will explain the entirety of our attribution approach, discussing the motivating experiments and the implementation details. The final model will be evaluated in the following chapter with all datasets specified in chapter 3.

4.1 Distribution manifold

In the previous chapter we discussed the problem with unnatural transformations in the input that result in samples outside of the manifold described by the data distribution. Although we introduced this reasoning in the discussion about ablation metrics, its implications reach further.

We will begin by focusing on Integrated Gradients, a powerful method previously discussed in section 2.2.3. It computed the pixel attributions by means of a comparison between the image of study and a predefined baseline, the black image. A linear interpolation path is computed between these two extremes and then the gradients of each element in this path is computed and aggregated in a final integral approximation.

This idea comes as a result of the realization that Counterfactual images, images in the middle of this interpolation path, carry more informative gradients than the actual image. We argue, however, that this approach is flawed for the same reasons that ablation metrics are flawed. An image resulting from a blackening of the original picture (the result of the $\alpha \cdot img$ interpolation) should not belong to the training manifold, since we have not trained our CNN with that type of transformation. As a result, the predictive capabilities of our

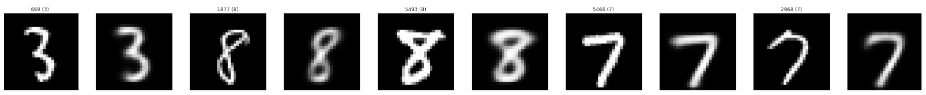


Figure 4.1: MNIST VAE reconstructions

model are not trust-worthy in these Counterfactual images, so the Integrated Gradients approach might not be reflecting the real reasoning inside the model.

We propose a different approach based on this idea of transforming one input into another, and taking advantage of the resulting transformation path for attribution. From this moment on, we will assume we are working with a MNIST-trained CNN classifier. This technique, however, can be extended to other image datasets, as we will see in chapter 5.

4.2 VAE manifold

If we want to work with a neighborhood of our image of study in order to understand the effect of local changes to the prediction score, we need a mechanism to work inside the data manifold. We propose the use of Variational Autoencoders (VAE), introduced in [25] and summarized in appendix B, to obtain an enconding of that space.

VAEs are a type of Autoencoder networks that force the resulting embedding codes to belong to an isotropic Gaussian distribution. This property is particularly interesting because it defines a region in the embedding space where points are more probably linked to the original data manifold. One only needs to use the density from that Gaussian distribution to ensure that the resulting decoded images belong to the original manifold.

For instance, if we wanted to study a particular image, we could use the VAE encoder to obtain an embedding representation, $c \in \mathbb{R}^d$, where d is the embedding dimensionality. If we now want to consider the neighborhood of that image, we can take neighbors in embedding space and decode them with the VAE decoder. If we only take codes with high density in our VAE distribution, we should be inside the data manifold.

We have trained a VAE for the MNIST dataset (the precise architecture for this or other models is detailed in chapter 5). Figure 4.1 shows pairs of original images and their reconstructions, respectively.

This approach, despite its simplicity, has its drawbacks. We must rely on a good VAE representation of the input. If the VAE we define is not complex enough, the quality of the decoded neighbors might be insufficient. Additionally,

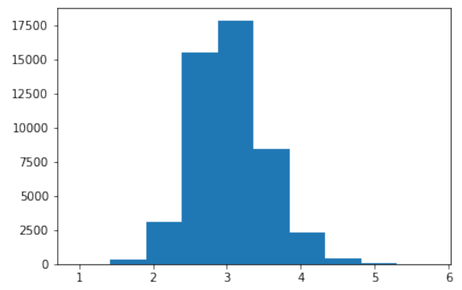


Figure 4.2: MNIST VAE embedding norm distribution

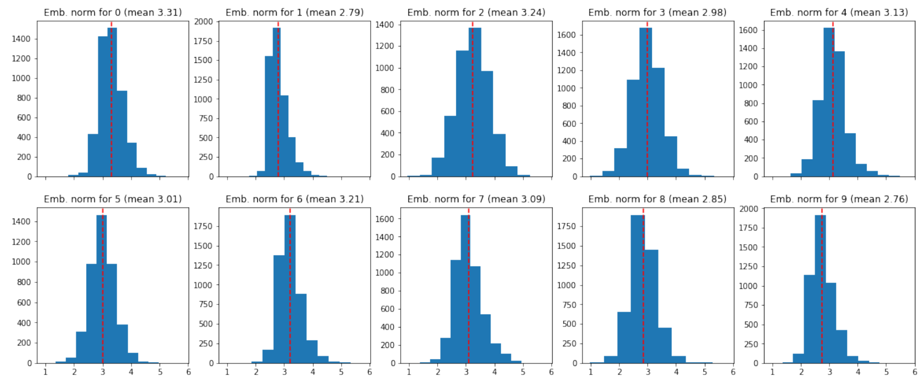


Figure 4.3: MNIST VAE embedding norm distribution, divided by class

the use of the embedding does not ensure that the resulting images are not adversarial.

We have an option to mitigate this, however. As said before, the embedding space defined by a Variational Autoencoder should follow (approximately) an isotropic Gaussian distribution. Instead of using the actual density values for that distribution, we can analyze, as a proxy, the norms of points. Figure 4.2 shows the histogram of encoding norms for all images in the MNIST training set. Figure 4.3 divides this histogram in a separate one for each digit in 0-9.

As we can see, the distribution of norms is not a normal distribution, as it is right-skewed, but it can serve as a proxy of what the network recognizes as real training input. Consequently, from now on we will use VAE's embedding with this norm distribution as an additional constraint to ensure that newly generated images belong to the data manifold. As a simplification, we will assume that each norm distribution per label is a Normal distribution, with parameters estimated from these samples. If that assumption held (although it is not really the case) the likelihood of a sample belonging to the class-specific norm distribution is proportional to the squared difference between the new image norm and the mean norm of that class. This is the reason for the inclusion of this term in the upcoming processes.

4.3 Choice of baseline and transformation path

Now that we know how to improve the likelihood of image samples, we can evaluate the second unclear assumption of Integrated Gradients. Why do we have to use the black image as a baseline? In the same way that Counterfactual images do not belong to the manifold, so does not the black image. Its choice, also, seems arbitrary. The only reason would be that the path between the baseline and the image represents the addition of features to the image so that we get our original input.

We propose an alternative approach. Instead of evaluating the global relevance of a feature in predicting a certain label, we can consider the following question: what features does this image have that makes the model predict class A and not B? Our method will try to evaluate feature relevance with respect to distinguishing two classes. Specifically, we will take the two most probable classes for this comparison. We believe that by focusing on attribution between two classes, we can discern the truly dichotomous features in the image.

Note that this method is particularly useful if it can explain what features made the model predict wrongly a certain input. If we ask the model which features made it consider our input as the wrong class with respect to the real class, we can evaluate strategies to overcome this source of failure. Additionally, this sort of focus can help in discerning the really relevant features for any pair of normally confounded classes.

With respect to this new problem perspective, the choice of baseline seems obvious. We will use as baseline an image that is as similar as possible to the original image while being predicted by the model as the secondary class. If that image belongs to the manifold (we want to avoid adversarial examples) then the interpolation path between both images would describe the required transformations to change the prediction of our model.

Four questions remain:

- How do we find this ”similar image with secondary label”? The first method one might try is to look at the training set for the most similar image belonging to the secondary class. The problem with this approach is that we are limited by the diversity of our dataset, which increases the data requirements of our approach. For the following experiments, we will work with this methodology for the sake of simplicity; figure 4.4 shows

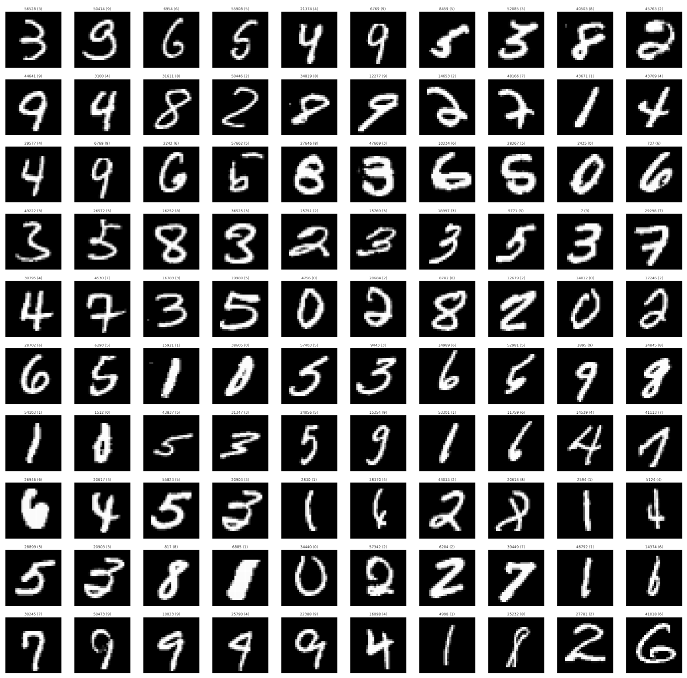


Figure 4.4: Pairs of images obtained with this method

examples of pairs derived from this method. In the upcoming section 4.5, we will discuss an alternative process to obtain these images.

- How do we define similarity between images? Taking advantage of the VAE embedding structure, we can use the embedding codes to compare images. The reason for this is that since the mapping in the encoder and decoder in a VAE is continuous, one should be able to compare similarity between the decoded images directly at the embedding level. Our choice of similarity function at the embedding level is the cosine similarity, since it evaluates the angle between the vectors defined by each code.

- How do we define this transformation path between both images? The approach in Integrated Gradients was to interpolate the image itself. We have argued that this approach leads to unnatural images. We will take advantage of the VAE embedding again and perform the interpolation directly at the embedding level. Then, with each coded step, we can decode with the VAE to obtain the intermediate representations.

  An additional consideration about this interpolation is required. Consider the case when the two points to be interpolated have the same norm (they are located in the same circumference around 0). Note that linear interpolation ($x_{x_0,x_1}(\alpha) = (1-\alpha)\cdot x_0 + \alpha \cdot x_1$) between these points would traverse regions with lower norm than the original ones. This, however,

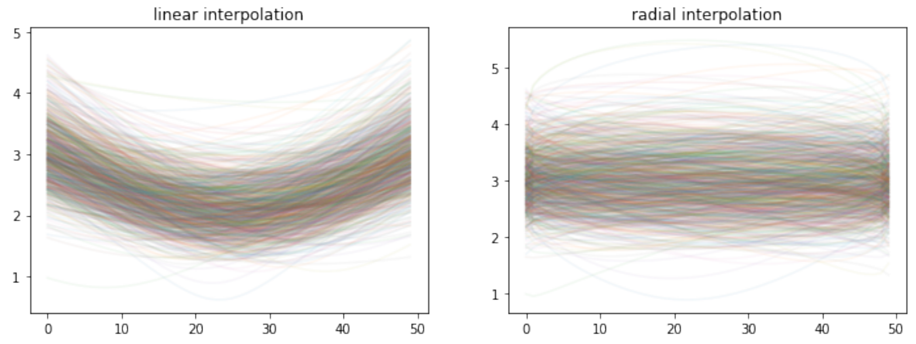


Figure 4.5: Interpolation - norm evolution

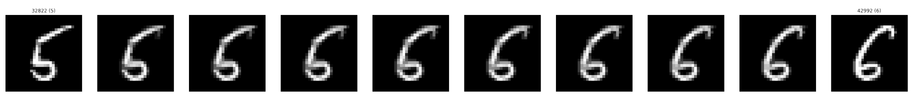


Figure 4.6: Interpolation path between a 5 and a 6

could lead our transformation path outside the manifold, since, as we saw before, the embedding norm distribution is right-skewed. Figure 4.5 (left) shows the evolution of the norm along a linear interpolation between embedding codes of random images in the training set. One can see clearly that linear interpolation could traverse regions with lower densities.

We propose an alternative interpolation method that preserves norms. We will call it **radial interpolation**. Its form is:

$$x_{x_0,x_1}(\alpha) = \sqrt{1-\alpha} \cdot x_0 + \sqrt{\alpha} \cdot x_1$$

Figure 4.5 (right) shows the same experiment as before, this time using radial interpolation. It is clear that the norm of the intermediate steps with this procedure is well-behaved in comparison with linear interpolation.

From now on, when talking about the interpolation path, we refer to this radial interpolation procedure. Figure 4.6 shows an example of interpolation between two images obtained by the aforementioned process.

- Will the gradients of the steps in the transformation path still be informative?

  Kingma ([25]) argued about the informativeness of Counterfactual gradients, but we do not know about the power of the gradients in our interpolation path. This is what the next section will analyze.

4.4 Entropy and gradient magnitude

Kingma argued that Counterfactual gradients are more informative because the gradients and activations of the real image are saturated. Since Counterfactual gradients are not, they should reflect which directions of transformation are more relevant for the prediction.

This statement, however, does not take into account what kinds of input the network may have learned from. If the Counterfactual samples are unnatural for the network, their behaviour could be unpredictable. Our transformation path step images should not be affected by this problem, since we have taken several measures to avoid escaping from the data manifold. However, this does not ensure that these images are informative. This is what we will analyze in this section.

Our principal observation with respect to this topic was that gradient magnitude (and, therefore, non-saturation) is related to classification entropy. If we consider the two labels of study in our transformation path (the original label of our image and the second most-probable label) we can compute the softmax of their logit scores (the result of the CNN) for these two labels, thus obtaining the probability of belonging to class A and the probability for class B. We can then compute the entropy between these two probabilities as:

$$H(p_A, p_B) = -p_A \log_2 p_A - p_B \log_2 p_B$$

This quantity measures the degree of ambiguity between both levels with respect to the network prediction, for each image. If we analyze the entropy along the transformation path for the 5-6 example in figure 4.6, we obtain the left plot from figure 4.7. We see that at some point, both classes are ambiguous for the network.

Now, we compute what we call the *mean entropy step* as the weighted step mean, with weights as the entropy values for each corresponding step. The result of this is the red dashed line at the center of the entropy peak. The two black dashed lines correspond to that average shifted by 1.96 times the *weighted entropy std*, another weighted average resembling the standard deviation formula. These bounds try to reflect the steps with increased entropy along the path.

If we now compute the gradient of the original label softmax with respect to the input image along the path, we obtain a series of 100 gradient values for each pixel in the image. If we plot those (the center plot in figure 4.7), we see that gradient magnitude radically increases in the bounds defined by the previous computations. Therefore, we might say that there is a relationship between gradient magnitude and entropy, which makes sense, considering that

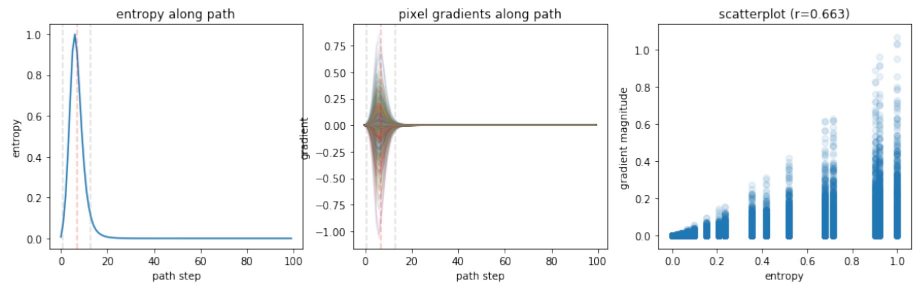


Figure 4.7: Entropy-gradient relationship for a specific pair

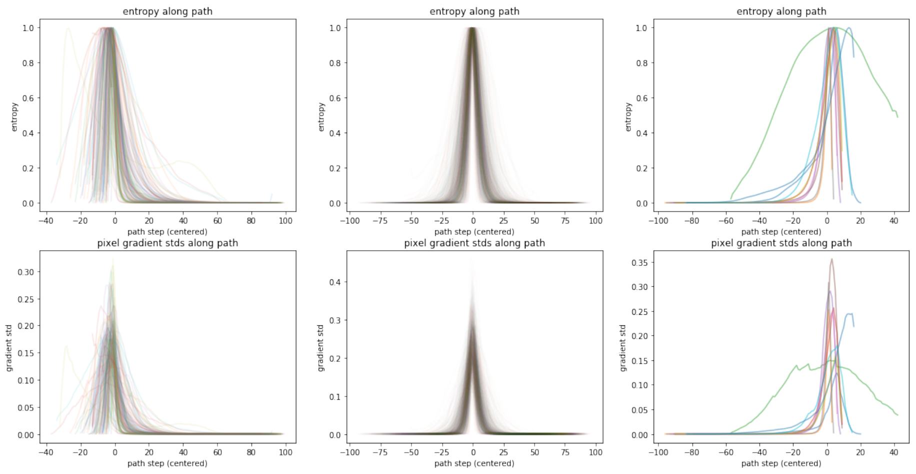


Figure 4.8: Entropy-gradient relationship for 1000 sample pairs

tiny perturbations in the input when the image is ambiguous might change the probability assignment radically. The right plot in figure 4.7 is a scatterplot of these entropy-gradient magnitude values, and also show an increasing trend in magnitude as entropy increases. In fact, the Pearson Correlation Coefficient is of 0.663 for this particular example.

Is this a particular behaviour of this example, or does it always happen? We replicate this process along 1,000 pairs of images from the training set and plot the results in figure 4.8. For clarity, we divided our experiments in left-center-right; if a pair is in the left plot, it means that the entropy window encompassed the first step in the path. That means that the left image was already ambiguous. The opposite behaviour can be analyzed in the right group. Finally, the remaining ones can be seen in the center plot.

In every one of these plots we can see that entropy is related to gradient magnitude, and the overall Pearson Correlation coefficient across all values computed in this experiment is of 0.593, which confirms our hypothesis of an upward trend between these two quantities.

What we conclude from this experiment is that our approach of computing the transformation path between these image pairs does highlight the important features that transform one label into the other. Entropy must increase as long as we change the class of an image into another, so one can expect gradients to increase in magnitude as well. Additionally, we should specially focus on those high-entropy steps, because those are the ones with the highest effect on the final prediction.

Even though this experiment was based on image gradients, we will not continue using them for the definition of our method, because their results were limited. Instead, we will define an approach based on this transformation path and on the idea put forward by Fong ([18], previously discussed in section 2.2.5).

4.5 Semantic Morphing

Before continuing discussing our approach, we need to solve a preliminary step to ensure better results at the end of the method. We stated that the transformation path between an image and another one, similar to the former but belonging to its second most-probable class, contained the most relevant features in predicting that image as belonging to a given class, in contrast with the secondary class. However, the approach for obtaining that secondary image is quite poor and dependent on the diversity of our dataset.

For simplicity, from now on we will refer to our original image as **self** (with label **self-label**) while the secondary image will be called **other** (with label **other-label**). Our objective is to define a method that is capable of *creating* **other**, an image similar to **self** but labelled with **other-label**, the second most probable class of **self**. We name this process **Semantic Morphing**.

This method will consist of an optimization process. The optimization objective is to maximize the logit score of **other-label**, and the optimization variable should be **other** itself. However, we must take into account several restrictions first.

- The resulting image should belong to the data manifold. Therefore, we should try to optimize the VAE embedding itself, instead of the raw pixels in the image. Additionally, since with the pure embedding we could step out of the manifold too, we can enforce the embedding norm restriction that we discussed in the previous sections.

  We will add a penalization loss to the optimization objective: the squared difference between the embedding norm and the mean norm of the **other-label** samples (as stated before, a term proportional to the likelihood of the norm belonging to the (assumed) gaussian distribution of norms. The resulting optimization objective is, then:

$$\max \quad lg_o - \alpha_{lk} \cdot (||c|| - ||\overline{c_o}||)^2 \quad \longleftrightarrow \quad \min \quad -lg_o + \alpha_{lk} \cdot (||c|| - ||\overline{c_o}||)^2$$

where lg_o is the logit score of class o, c is the embedding code (the optimization variable), c_o is the average code for all training instances belonging to class o, and α_{lk} is the hyperparameter that controls the effect of the likelihood regularization loss. In this case, o should be **other-label**.

- We should enforce similarity between the original image (**self**) and the new one. This could be done with an additional penalization loss to the objective, but we deemed preferable not to include it. In our experiments, the inclusion of this loss made it harder for the optimization to escape from the initial embedding. Moreover, if we did include this term, an additional hyperparameter for its corresponding loss would be necessary and the optimization process would end up being too dependent on hyperparameter selection.

  Instead, to make sure that the new image resembles (as much as possible) the original image, we will opt for early stopping. Since the gradient optimization is continuous and we use low values for the learning rate, as long as the optimization process does not last many epochs, we should not be very far away from the original embedding. Since the decoder is also continuous, one should expect that the two inputs are relatively similar. For the purposes of this project, this simplification is enough.

  The stopping criteria is to check if the entropy between the two classes of interest (**self-label** and **other-label**) is below a certain threshold (0.01 in our experiments), which would mean that the network has already decided without ambiguity for a certain label. However, this is already true for the original image, since the label tends to be non-ambiguous for them. Therefore, added to the entropy condition, we also check that the **other-label** has probability above 0.5.

Note that if we connect the VAE decoder to the CNN model, we obtain a differentiable model with respect to the loss above. We can use the optimization methods from any Deep Learning package to perform this optimization, as long as we can include the stopping criteria described above.

As a final note, for this and any future optimization procedures that we describe, we do a certain normalization step to express all loss terms in the same scale. During the optimization we compute a progressive average of each loss term and use that average to normalize all regularization losses to the scale of the optimization objective. For simplicity we do not include these terms in

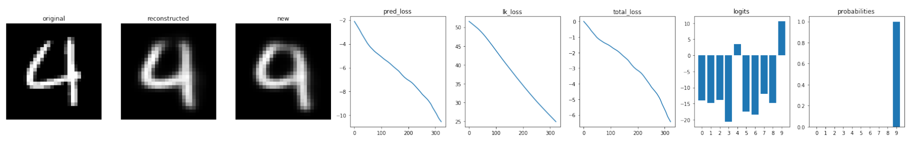


Figure 4.9: Semantic morphing example between a 4 (original) and a 9 (new)

the objective loss formulas, but one can multiply them to the corresponding regularization-control hyperparameters (in this case, α_{lk}).

Figure 4.9 shows an example of semantic morphing done with the MNIST dataset. Note the evolution of each loss term and the total loss, and the resulting logits and probabilities for the final image. This experiment was run in less than 350 epochs with an Adam optimizer of learning rate 10^{-3} and $\alpha_{lk} = 0.1$. For more examples of Semantic morphing and a proper analysis of its performance, refer to the next chapter, where we will evaluate these techniques with all the previously discussed datasets.

4.6 Path-Mask attribution

In this last section we will define our attribution method, based on the work of Fong ([18]). We call our method **Path-Mask attribution**, since it is based on a mask (resembling the one in [18]) that transforms the images in the transformation path defined with the previous Semantic Morphing and radial interpolation processes discussed before. This mask will try to maximize the gain in the prediction of the desired label (**self-label**) while trying to fulfill some regularization constraints. This will be achieved with another optimization model.

Fong used the mask to perform transformations on the input image as a way to hide some evidence that might affect the classifier prediction. We will not apply any such transformations because, as argued before, we consider them to derive in unnatural images. Instead, we will use the same operations the transformation path applies to the images step by step, in such a way that the impact of the transformations on the images should not be noticeable for their likelihood with respect to the data manifold, but their aggregated impact on the final prediction is noticeable.

We define the objective mask m as a matrix with the same shape as the original image. Its elements are numbers in $[0, 1]$ that reflect the importance of each pixel in the transformation path. Let $(x_i)_{i=1..n}$ be the transformation path between the original image (x_n) and the semantically transformed one (x_1). This path is formed of n steps. For every step $i < n$, we define its mask-transformation, denoted by $m(x_i, x_{i+1})$, as:

$$m(x_i, x_{i+1}) = (1 - m) \odot x_i + m \odot x_{i+1}$$

We can now define the objective function for this optimization:

$$\sum_{i=1..n-1} f_o(m(x_i, x_{i+1})) - f_o(x_i)$$

where $f_o(x)$ is the softmax probability of class o (**self-label**) for input image x. Notice that this optimizes the mask to highlight which pixels in the image need to be transformed (following the transformation strategy of the transformation path) to gain in probability of the class of interest.

This objective, without any constraints, could just highlight all pixels. We also want to enforce some regularization in this respect, so we include the L_1 norm of the optimized mask as a penalty term. We use the L_1 norm to achieve a sparser mask.

Finally, if we used this optimization directly, we would probably get a mask with high-frequency patterns. In order to avoid this, we could use additional regularization losses, like the Total Variation (TV) loss discussed before for Feature Visualization in section 2.1.3. Alternatively, in order to avoid unnecessary additional hyperparameters, we will include a Gaussian blur filtering between the image variable and the optimization procedure, so that the network learns a *blurrier* mask, less prone to these noisy artifacts.

Again, we run this optimization process with an Adam optimizer of learning rate 10^{-3}. Figure 4.12 shows an example of this method.

The top three images in this figure show the original image, its reconstruction with the VAE decoder and the result of the Semantic Morphing process with this reconstruction embedding. Note that the embedding is not the mean embedding offered by the VAE encoder, but a sample from the distribution the encoder returns that resembles the original image the most while still preserving its prediction with our CNN. That way we ensure that the image and its reconstruction are as similar as possible.

The next two rows of images show the transformation path between both images. Note in this case how the transformation path already shows us what features need to change to obtain a different class. This is by itself a powerful tool for human inspection of possible prediction errors.

Finally, the last row analyzes the mask and its effects on both images. The first plot shows the evolution of the loss at each epoch in the optimization. The last plot shows where in the transformation path are the gains located. Finally,

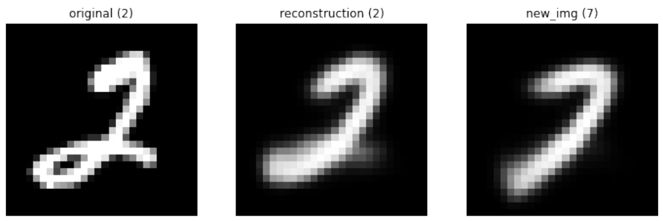


Figure 4.10: Semantic morphing

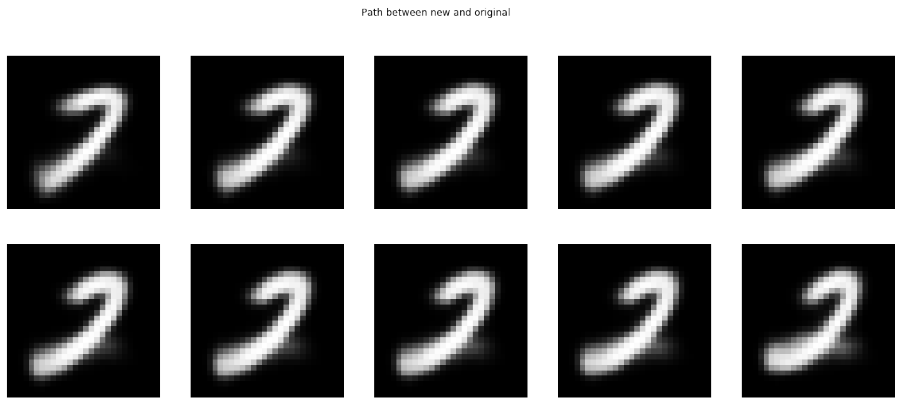


Figure 4.11: Transformation path

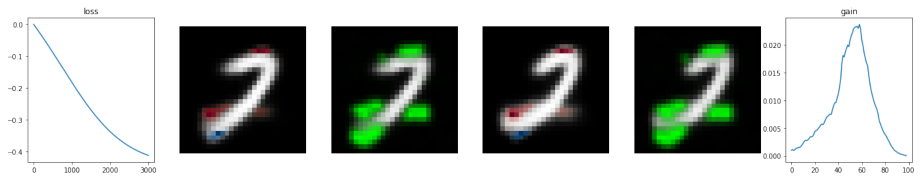


Figure 4.12: Path-mask attribution

the plots in the middle with green spots show, in green, where in the image is the mask. Note that this mask only tells us which parts of the image need to evolve like the transformation path does in order to obtain **self-label** from a similar image with **other-label**. The two plots in red and blue highlight these areas with red for an increase in pixel intensity and blue for a decrease. Therefore, the red spots indicate that we need to add the bottom horizontal stroke to achieve a 2, while we need to reduce the end of the downward stroke to create the curve that connects with the previous horizontal stroke.

This is just an example. In the following chapter, we will discuss the results of the presented methods in detail.

Chapter 5

Results

In this chapter we will evaluate our approach with the specified datasets and discuss its strengths and shortcomings so as to identify possible new paths of research. For each dataset, we will show examples of the VAE reconstructions, of the semantic morphing process and, finally, of the path-mask attribution method. Each of these experiments will help us understand the potential of our approach.

5.1 MNIST

We will discuss first the reconstruction capabilities of our (Convolutional) Variational Autoencoder applied to the MNIST dataset. The precise architecture of this and other models defined throughout this work can be found in Appendix C.

Figure 5.1 shows pairs original-reconstruction. One can see that the results are blurrier than the original images, but they manage to reconstruct the overall shape and stroke width for most cases. Notice the 8 in the second row that is reconstructed as a 7. This is a problem of the VAE approach. If the input is unnatural with respect to the rest of the dataset, the reconstruction can try to make it seem more natural, sometimes changing its label. Due to this problem, when performing the rest of the steps in the method, we choose the embedding code that, when decoded, results in the most similar image to the original input and that it is predicted by the CNN with the same two most probable classes as the original image. This can be done using the encoder of the VAE. The encoder of a Variational Autoencoder returns a mean and std of the distribution generated by the image to encode. What we do to obtain the aforementioned

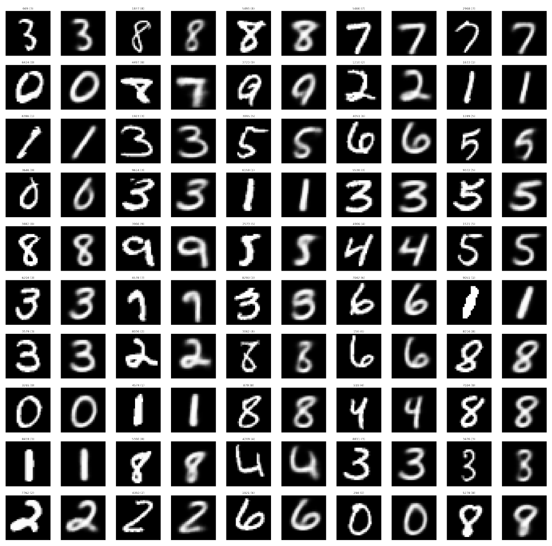


Figure 5.1: VAE reconstructions for MNIST dataset

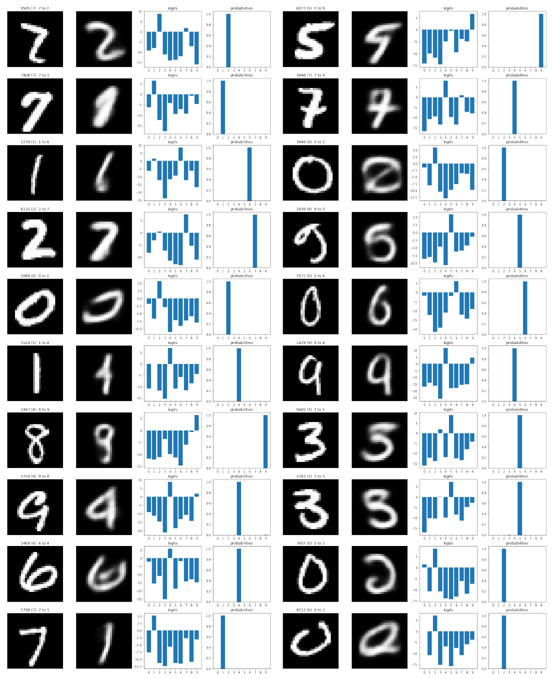


Figure 5.2: Semantic morphing pairs for MNIST dataset

image is to sample from this distribution, filter to the codes that, when decoded, return the same prediction as the original image and finally choose, from this subset, the image that is most similar (in L_1-norm) to the original input.

Now that we know that the VAE behaves appropriately, we move on to the Semantic Morphing step. Figure 5.2 shows pairs original-new, along with the logit and probabilities (softmax) of the new image. Note that the new images are, again, blurry (due to the VAE limitations) but also not always natural. As an example, take the 0-to-2 in the fifth row. The image is correctly predicted as a 2, but we know that it is not a natural image. This, again, highlights the importance of a good Semantic Morphing process capable of automatically generating the images while still making them seem natural and part of the data manifold.

One must note, however, that the result of this process is the result of an optimization, so there is no guarantee that we will finally obtain an image with the required class. In order to ensure the proper performance of this method, in terms of convergence to a solution, we test it with a 100 samples and compute the Semantic Morphing error rate, which is the ratio of samples that could not be semantically morphed (we did not find a new image with the required class). For the MNIST dataset, this error rate is just a 0.03 (3 entries in our 100 samples). The exact hyperparameters for this optimization are a maximum of 5000 epochs, with a 0.01 entropy threshold and a α_{lk} likelihood regularizer weight of 0.1.

These results look promising, so we can move on to the final step: path-mask attribution. For this process, we use 3,000 epochs with a mask L_1-regularization parameter of 1. Figure 5.3 shows examples of attribution for several samples. The precise meaning of these visualizations can be found in section 4.6.

It seems clear in all these cases that the method is capable of highlighting the features that really define the label of study (with respect to **other-label**). Note, however, in the last example (1 to 7) that the mask by itself is not that informative. The mask only highlights the important regions in the transformation path, but it does not tell us what kind of changes are required. That is where the red-blue heatmaps are important. This technique can be useful in this kind of "binary" image, but when dealing with more advanced datasets with coloured real images, with texture, shadows, etc., this process will not be enough. Further work with respect to this problem will be necessary to extend our method to more complex datasets.

We also include an example of a bad VAE reconstruction and how it affects the whole attribution method in figure 5.4. One can see that since the reconstruction is basically creating a 5, the attribution method is focused on highlight the features of a 5 instead of the ones we actually care about, the features of a

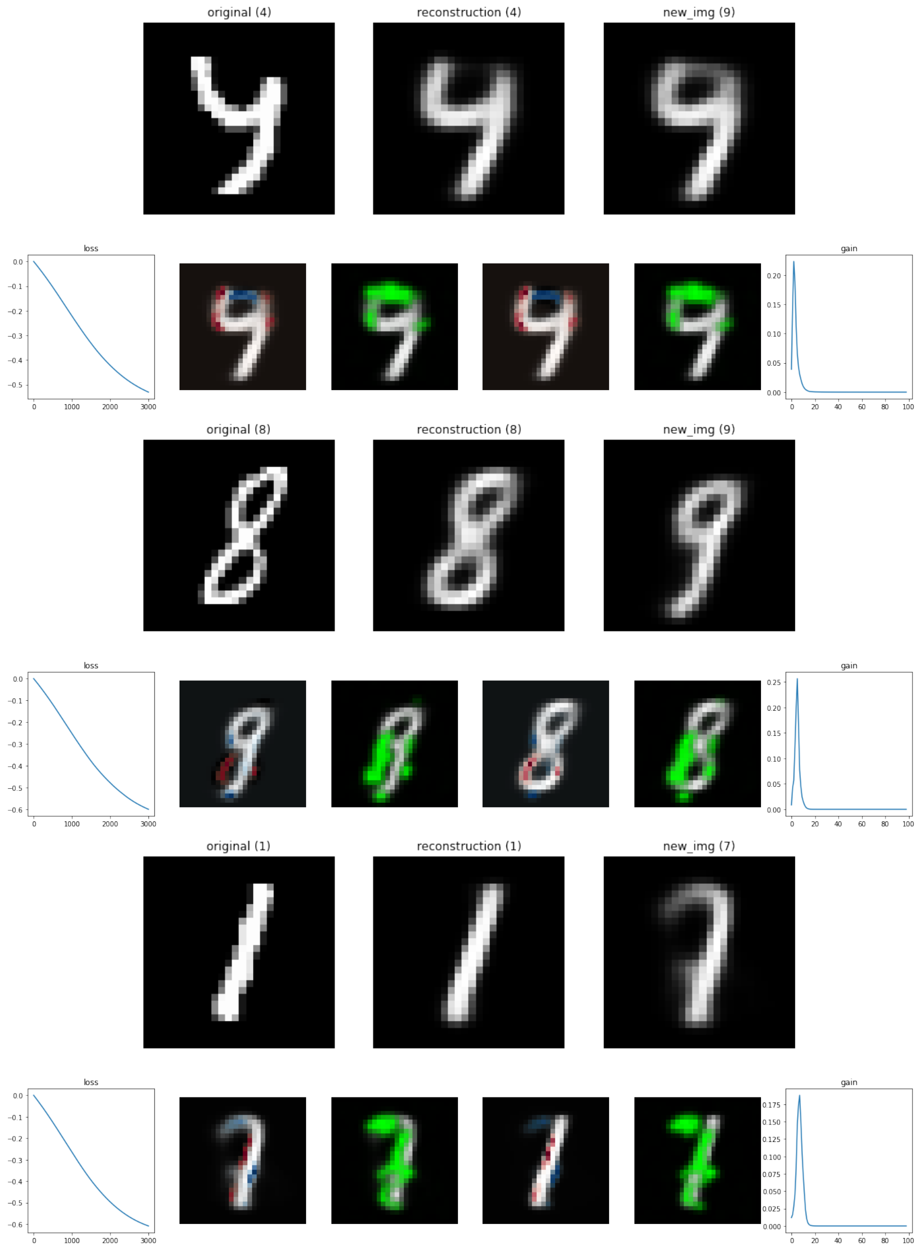


Figure 5.3: MNIST - Path-mask attribution examples

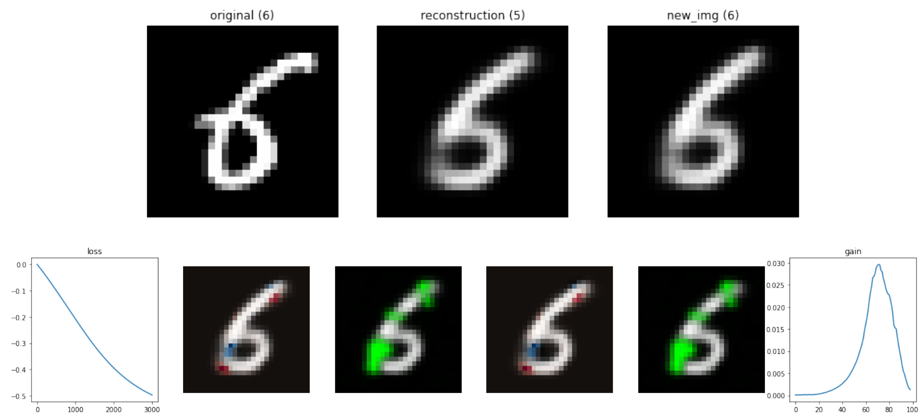


Figure 5.4: MNIST - Example of wrong reconstruction

6, which the image really is. This is definitely a shortcoming of our approach, since it is too dependent on the good performance of the previous steps (VAE reconstruction and Semantic Morphing).

Finally, we will end this section by analyzing the performance of our approach on images that were wrongly labelled by the CNN, in figure 5.5. In this case, the title on top of the original image shows the real label according to the dataset, the title on the reconstruction, the label the CNN predicts, and the new image, the label it was supposed to predict. Therefore, we now start with what should be the right label and advance through the transformation path to the reconstruction of the original image that is wrongly labelled. That way, we can analyze the reasons why an input was wrongly classified.

In the first case, the mask tells us that there is a vertical stroke at the bottom left part that confirmed the prediction of a 6. That is the reason for the wrong attribution. On the second picture, the length of the top horizontal stroke made the CNN confound it with a 7. Finally, in the last picture, the closing of the circle and the absence of a stroke at the center of the circle made the network think this was a 0. All these cases show the usefulness of our approach in detecting the reasons for a certain classification. However, as we already stated and will continue to see in harder datasets, the reliance on good VAE representations is paramount in the final performance of our model.

5.2 NotMNIST

We will start with the VAE reconstructions for the NotMNIST dataset, in figure 5.6.

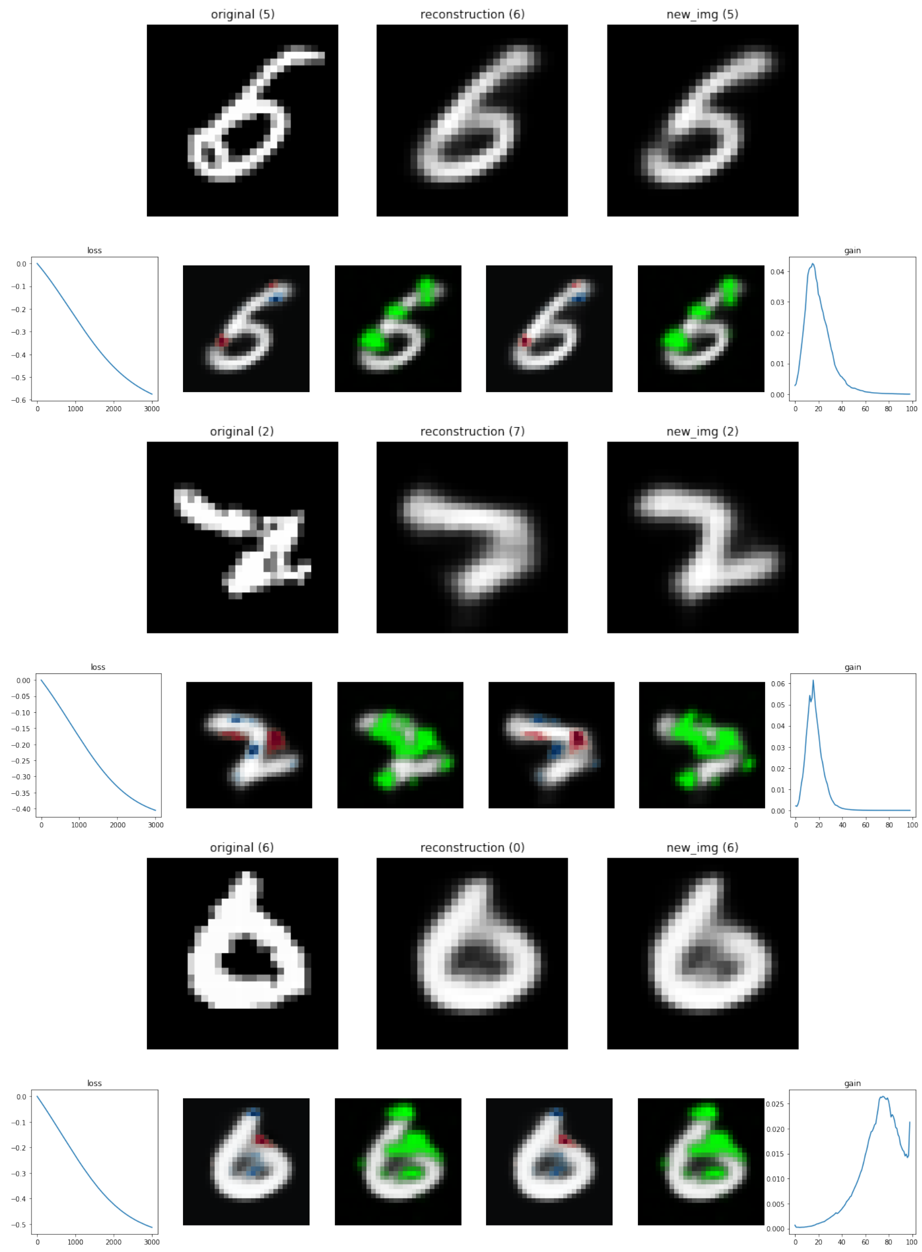


Figure 5.5: MNIST - Path-mask attribution examples with wrongly labelled samples

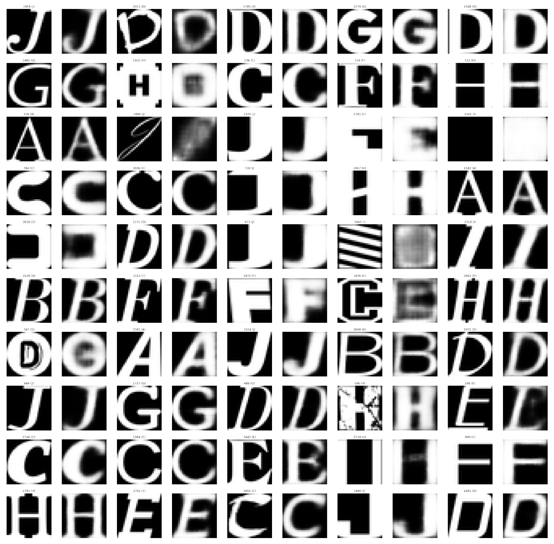


Figure 5.6: VAE reconstructions for NotMNIST dataset

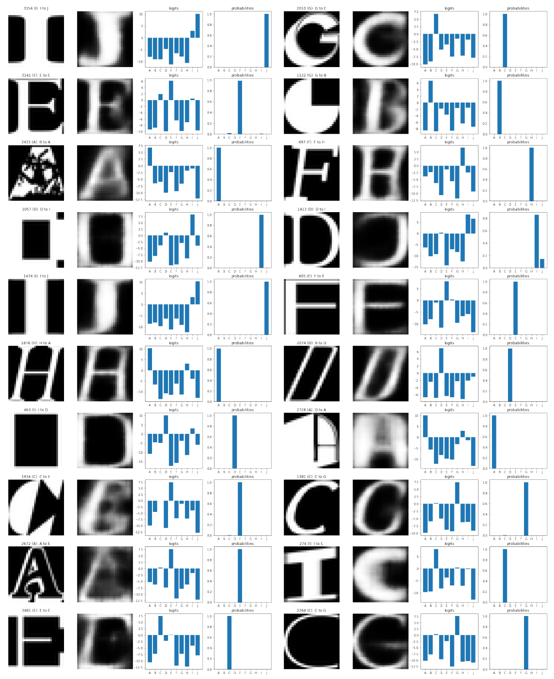


Figure 5.7: Semantic morphing pairs for NotMNIST dataset

Here we see a huge variety of fonts for letters A-Z, with different stroke widths, letters with all stroke and no fill, letters in white background, etc. The quality of the VAE representation is affected, specially when the font is very different to the rest, like in the third row, third column. Again, this supports the argument that we need better methods for input embedding in order to continue with our approach. Alternatives to the current VAE strategy can be found in chapter 6.

If we continue with the Semantic Morphing step, in figure 5.7, we see how most of the generated images are affected by this blurriness, even more than with MNIST. Again, this is a consequence of the limitations of the VAE approach. Nevertheless, the results are still sufficient, and if we run the performance experiment on Semantic Morphing like before, the error rate is of 0.02 (two errors in a sample of a hundred), which is still very good. The hyperparameters for Semantic Morphing in this case are a maximum of 5000 epochs again, 0.01 as the entropy threshold and a α_{lk} likelihood regularizer weight of 10. This noticeable increase in this weight comes as a result of the harder representations needed for this dataset, which implies that the results of this process are more sensitive to adversarial effects.

Finally, for path-mask attribution, we still work with the same hyperparameters and the results can be seen in figure 5.8. The mask is still capable of detecting the relevant features in the transformation path, even in the last example, where the only change required is the addition of a small white spot that hides the cut of the B.

The NotMNIST dataset still provides very good results, but there are more wrong attributions that we can analyze. Figure 5.9 shows three interesting examples of this wrong behaviour.

In the first example, although Semantic Morphing creates what the CNN sees as an H, it is not really an H for human supervisors. The attribution is still good with respect to this bad result, since it highlights both the positive and negative evidence, but does not say anything about the top and bottom horizontal strokes that should not be there, since the transformation path does not change them in any point. The second example shows us the limitations of the Convolutional VAE approach, since the reconstructions are quite limited and the transformation is not that different from that reconstruction. The mask is therefore affected, and here we cannot detect any relevant features because the transformation path is not relevant at all. Finally, the case of the last picture is actually a problem of the NotMNIST dataset. This is an example so different from the rest of the dataset that the network cannot predict it correctly, nor reconstruct it appropriately, nor semantically transform it. As a result, the mask attribution is insufficient too.

We argue that these results are not detrimental for our approach, but ac-

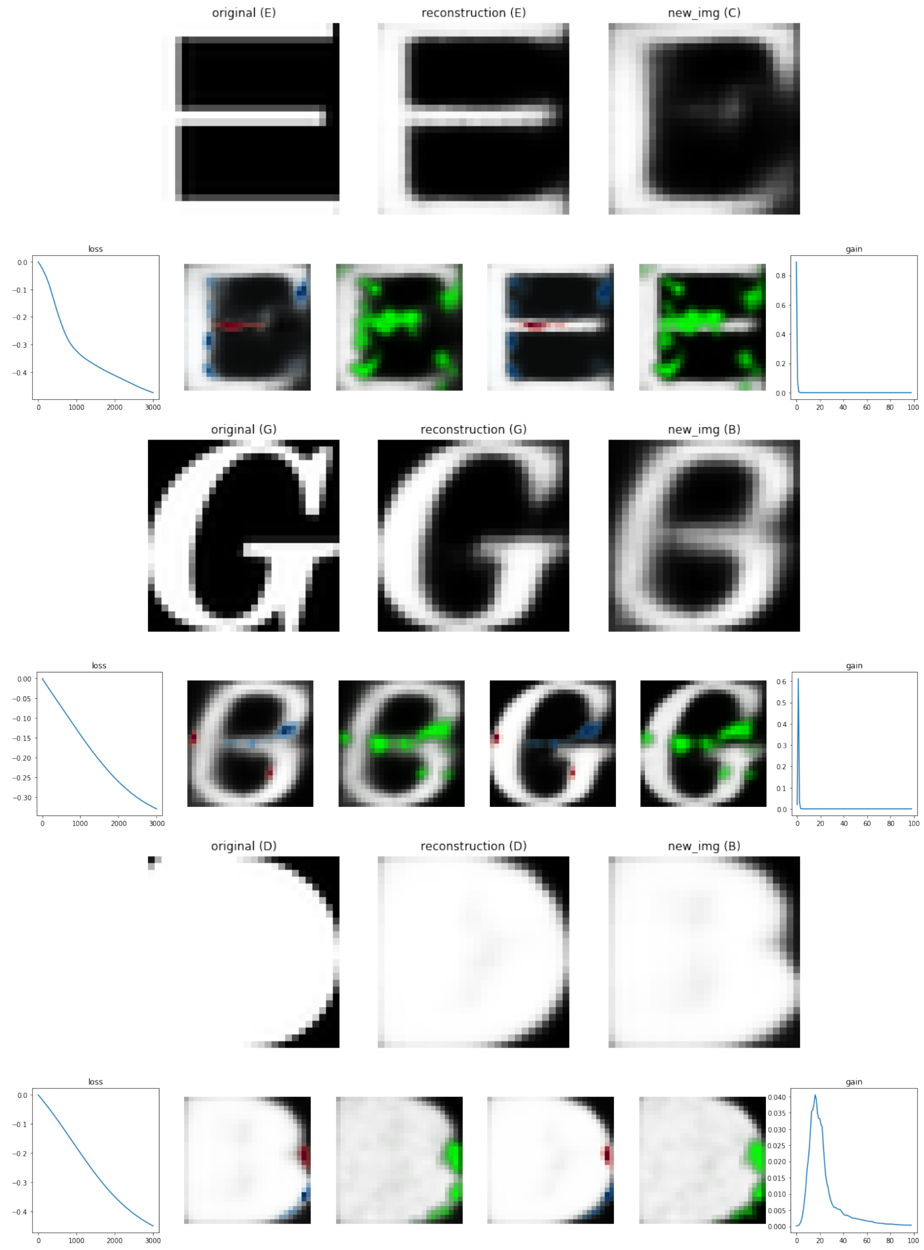


Figure 5.8: NotMNIST - Path-mask attribution examples

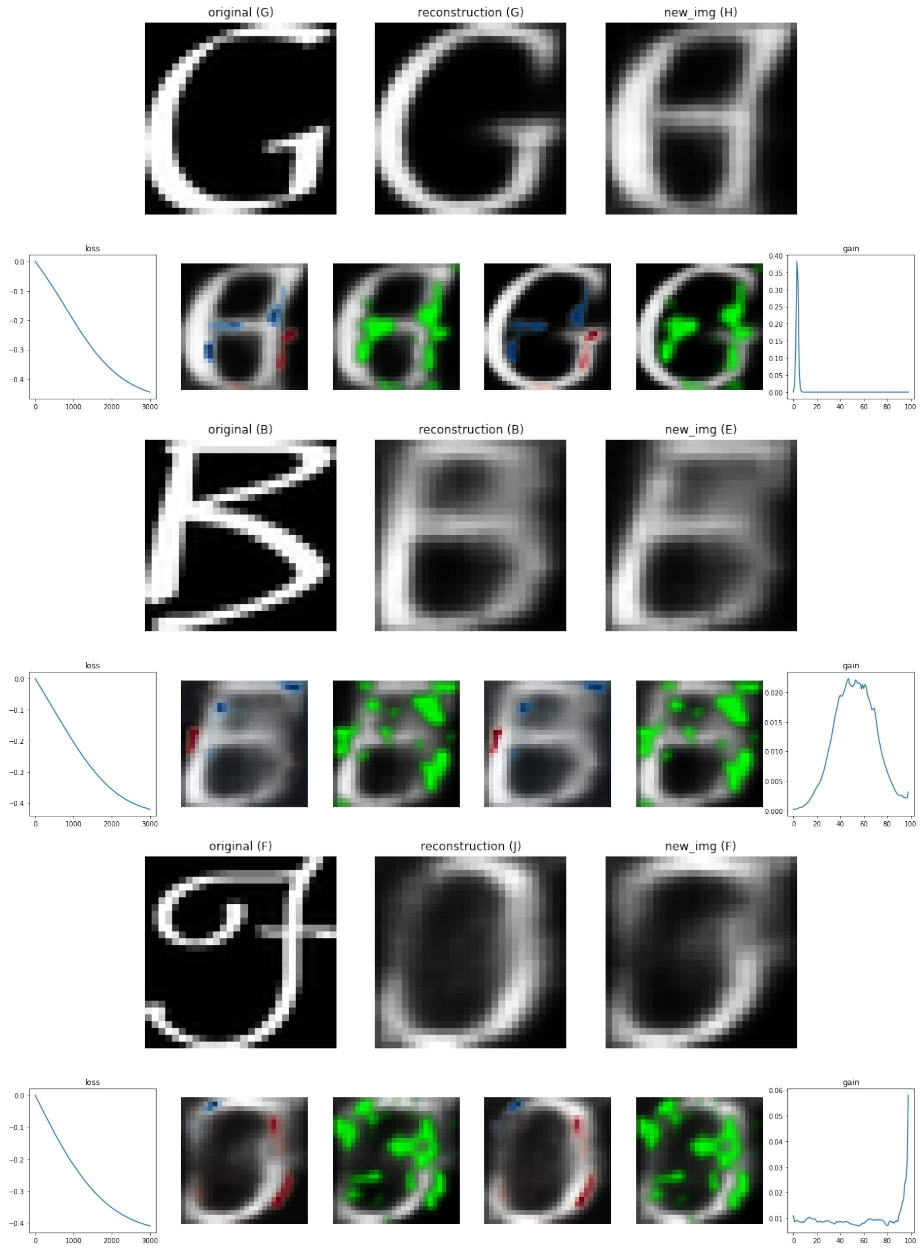


Figure 5.9: NotMNIST - Error examples

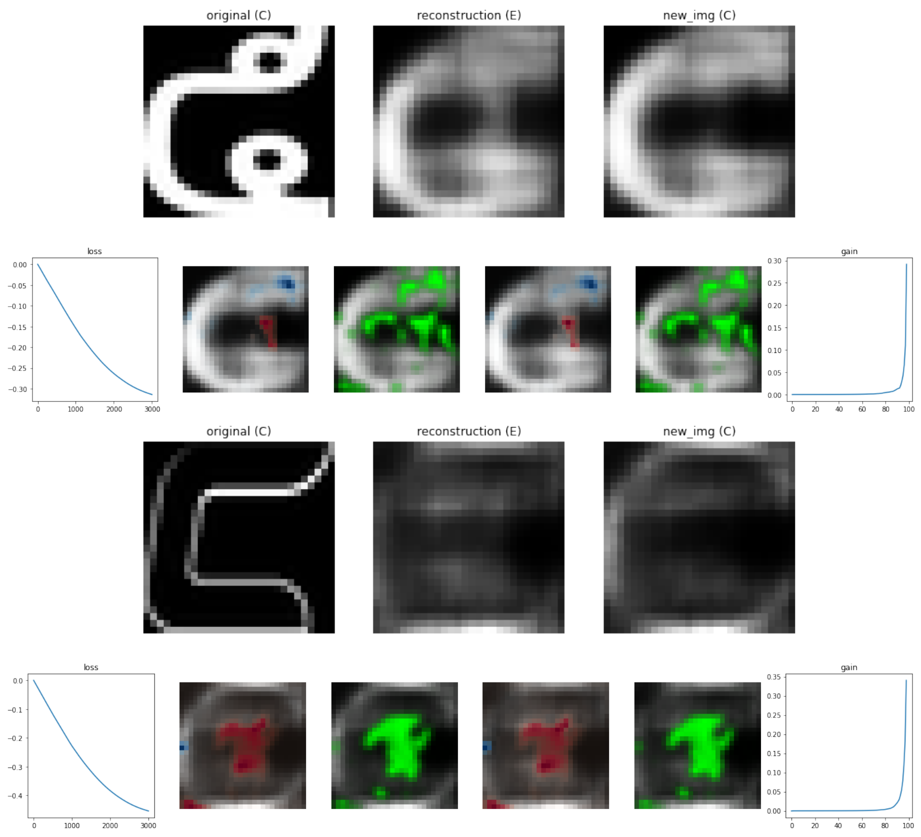


Figure 5.10: NotMNIST - Path-mask attribution examples with wrongly labelled samples

tually good constructive criticism on its weaknesses and ways to improve it. Again, this will be analyzed in more detail in chapter 6, so from this moment on, we will stop mentioning this dependency between the three systems for the good performance of our approach.

Finally, we will end with some wrongly-predicted examples that can be analyzed with our attribution approach. These two examples can be seen in figure 5.10. Both of them show us that the problem is that they seem unnatural to the CNN (and the VAE) with respect to the remaining of the dataset. Their reconstructions are quite poor, which also affects the Semantic Morphing process. However, the examples are more informative than it appears.

In the first case, the network apparently misinterprets the two circles at the center of the image as being part of the middle stroke of the E. That is the reason why the attribution highlights that area in red. On the other hand, the

C is confounded with another E because the network is not used to seeing ”all-border, no-fill” letters, as can be seen by the poor quality of the reconstruction, and this affects the whole process.

Despite the good results seen before, these error examples highlight the weaknesses of the method. However, now that we can identify the cause, we can propose new approaches to improve upon this. This will also be discussed in chapter 6.

5.3 Other datasets

The following two subsections will try to apply these techniques to harder datasets, FashionMNIST and CIFAR10, but with very poor results. We include here some of the experiments for completeness and to identify new problems we can work on.

FashionMNIST

We start again by the Convolutional Variational Autoencoder, in figure 5.11. Here the results are good in terms of the shape to reconstruct, but hide any details of the items they depict. This will definitely be a problem later in the process, because the classes we are trying to identify here are very confounded by themselves, and with the loss of so much detail their attributions are not going to be good.

Figure 5.12 shows some of the previous problems. Since the VAE can only mimic the shape of objects, the Semantic Morphing process can only try to change that shape, with very noisy, blurry, non-clear results. The performance in terms of error rate is also much worse, a 0.19 (19 errors out of 100 samples) and the results are not that natural. The choice of hyperparameters is the same as with NotMNIST because their tweaking did not accomplish any meaningful improvement.

Finally, we include some examples of attribution, although they are definitely wrong due to the limitations we just discussed. Figure 5.13 shows a good example (among many wrong ones) where the mask manages to highlight the definite features that change the prediction: the back of the shoe, converting the sneaker into an ankle boot, and the bottom of the sneaker, that changes into a heel. The mask, though, is too noisy, probably because of the very few detail in the VAE decodings.

We also show two clearly wrong attributions, in figure 5.14. In both cases,

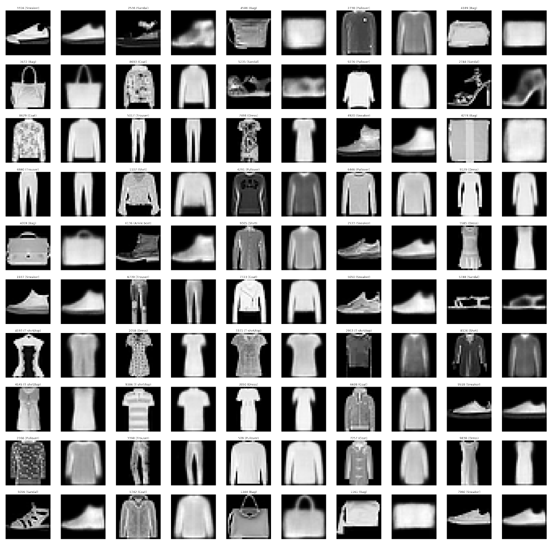


Figure 5.11: VAE reconstructions for FashionMNIST dataset

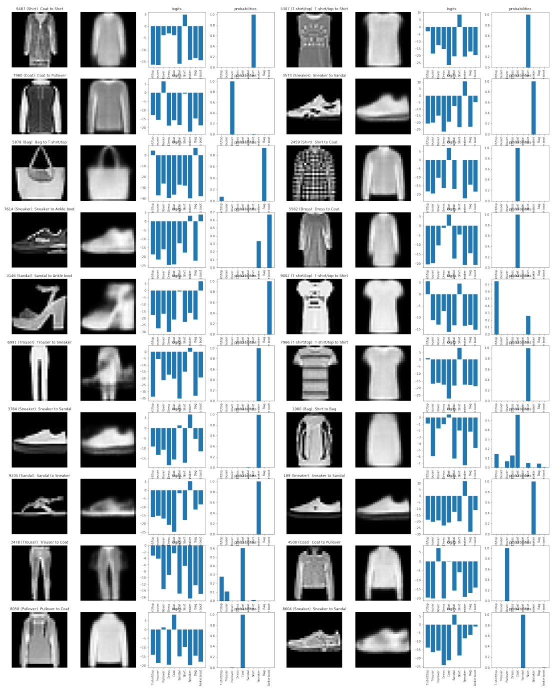


Figure 5.12: Semantic morphing pairs for FashionMNIST dataset

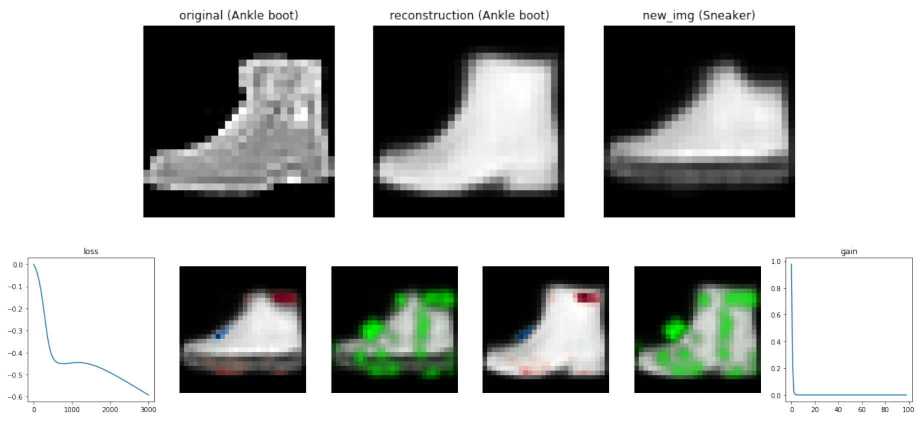


Figure 5.13: FashionMNIST - Path-mask attribution examples

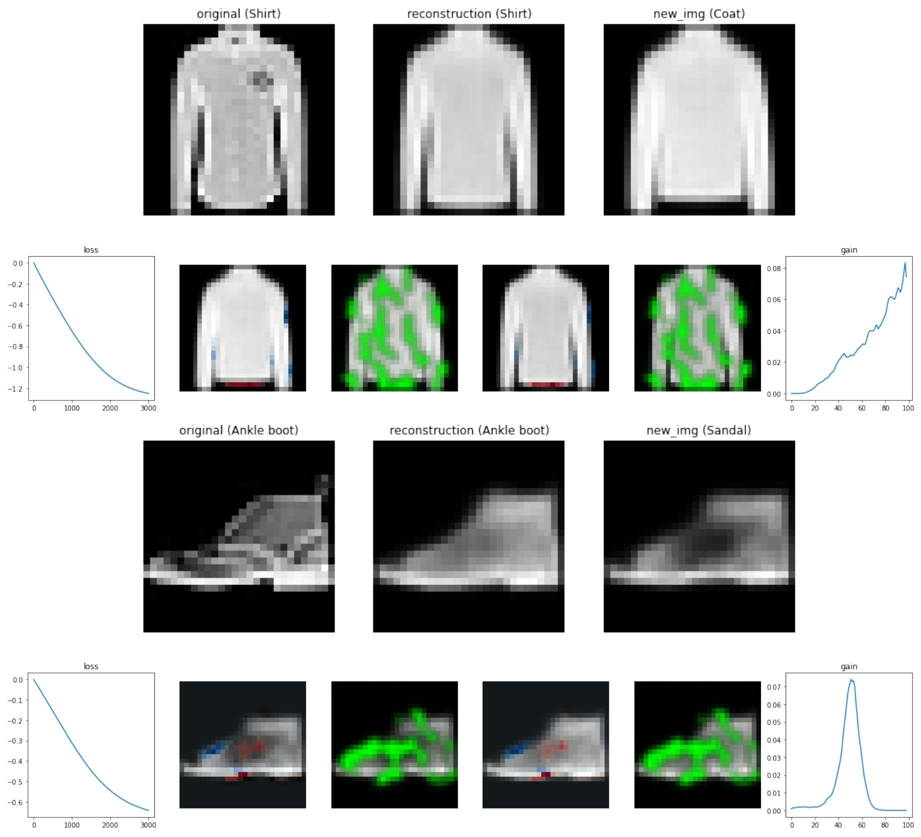


Figure 5.14: FashionMNIST - Example of wrong reconstruction

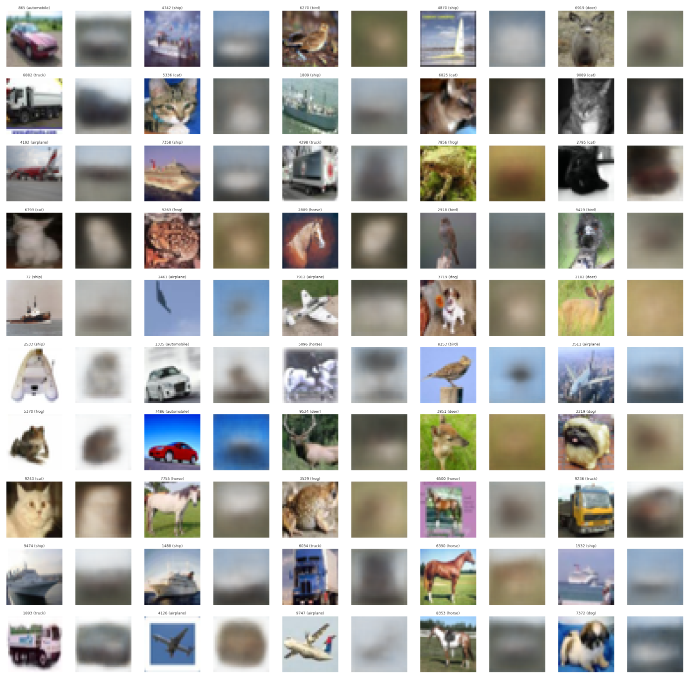


Figure 5.15: CIFAR10 - VAE pairs

the VAE gives bad reconstructions and the mask-path method cannot do anything, resulting in an almost random mask.

CIFAR10

We only use CIFAR10 to prove for the last time the shortcomings of the Convolutional Variational Autoencoder approach. Figure 5.15 shows the VAE pairs experiments. There we can see that the VAE only manages to create blurry spots of colour that, although they actually represent the original image, are so devoid of shape that they are useless for our purposes.

This ends this chapter of results. Despite its meager ending, we should not forget the good results in the original datasets. What we try to explain in this chapter is which are the specific points in our approach that create the problem. In the following chapter we will wrap up our research, state the specific points we identify as weaknesses and propose alternatives and new lines of research to further extend our method to these harder datasets.

Chapter 6

Conclusions

We begin this final chapter with a short overview on the proposed method strengths and weaknesses in section 6.1, in an attempt to identify possible research paths to explore in future projects, which we will detail in section 6.2, to finally end with our conclusions in section 6.3.

6.1 Model strengths and weaknesses

In this project we propose an attribution approach based on detecting important features in the dichotomy between two classes for a classifier. We show that our approach can give us an actual explanation of the reasons behind a specific prediction, and can be particularly useful in debugging a model when we detect some mistakes. By identifying the features that motivate certain prediction errors, we can work on extending datasets with samples of that problem that might teach the network to correctly predict them, as there would be more evidence to work with. Therefore, this focus on class-to-class attribution can be beneficial to model interpretability.

Additionally, the use of mask methods instead of pure gradients creates a smoother, less noisy attribution map, which also tends to be more informative than the usual saliency maps of gradient methods filled with noisy patterns. Even if the mask method does not provide a clear answer to this question, the transformation path, analyzed by human supervisors, might highlight which features need to change to obtain the desired label, so even that part of our approach can lead to interesting conclusions.

Nevertheless, the approach is still insufficient. We are aware that better results on embedding reconstruction and semantic morphing are required to tackle harder problems. When any of these parts is lacking for a particular input or dataset, the whole approach cannot be used. This is the reason why most of our future research will try to improve on this.

An additional problem we encountered was how to visualize the meaning of the transformation mask. Since the mask only tells us what pixels in the image need to change in the same way as the original transformation path, simple subtraction of pixel intensity in those regions might not be sufficient for more subtle elements like texture. In this direction, we also propose additional research that tries to overcome this problem.

6.2 Further work

As stated before, the main problem with our approach is the quality of the embedding reconstructions and the semantic morphing process. This raises the question of whether interpretability as a whole will always be limited by the capacity of our generative models. For this particular problem we will attempt to use more sophisticated approaches, like Generative Adversarial Networks (GANs), proposed by Goodfellow in [26], that have achieved state of the art performance in realistic image generation, among many other applications. For our purposes, we need a generative model capable of creating realistic, detail-preserving networks, that also encode the original images in continuous embeddings. It is also important to be able to characterize the manifold distribution at the embedding level, since that is what helps us in defining the likelihood of our generated images when working on Semantic Morphing.

A particular example of GAN that might prove very useful for our approach is CycleGAN ([27]), a type of GAN specialized in image translation. As an example, if the input image belongs to the type *horse* and we ask for type *zebra*, the network will edit the image in such a way that the new label will be *zebra* while preserving enough of the original image so that we can still recover it from the transformed one. This ensures that the information that defined our original image is not lost in the class transformation. Figure 6.1 shows an example of this technique. This approach is particularly interesting for the Semantic Morphing part, since we can obtain more realistic images that belong to the class of interest. Therefore, we think this might be a worthwhile experiment for future work.

Finally, in terms of the applicability of mask attribution, we think that applying the same technique to the convolution activations directly, instead of to the actual pixels, might help in discerning discriminating, informative features. The idea of the mask applying changes to the images is equivalent at the activations level, but there the information is more precise than with

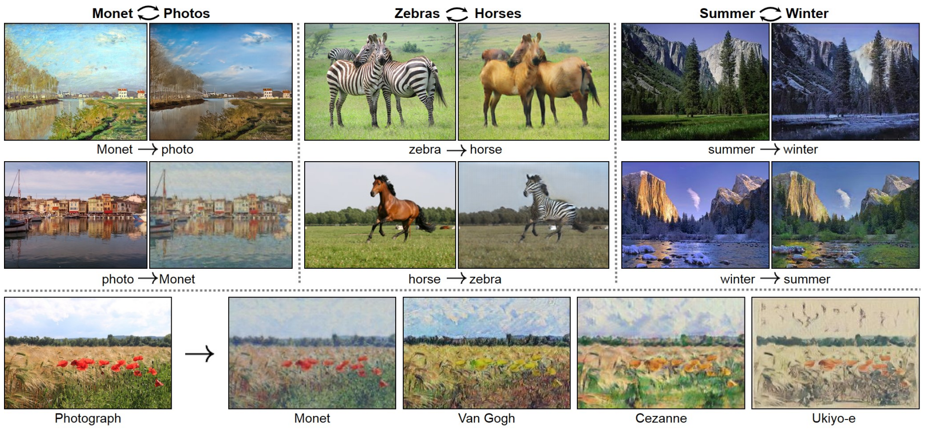


Figure 6.1: CycleGAN examples

raw pixel intensities. Additionally, we could employ dimensionality reduction techniques like the embedding approach in section 2.2.5, that could also remove much of the noise contained in the layer. By denoising the activations with this and applying the mask to highlight the important embedded features, we should improve even further the attribution results. However, this also requires an additional visualization strategy to understand what each of the features detected at the activations level means. This could be achieved either with feature visualization or with pixel-attribution techniques.

6.3 Conclusions

This project has helped us in identifying a new path to tackle the problem of model interpretability, specifically for Convolutional Neural Networks applied to image processing. The change of focus to study attribution between two classes allows for more interpretable explanations. We have also found that the point of maximum entropy between these two classes, obtained along the transformation path that comes as a result of the Semantic Morphing process, does carry the most relevant information in the explanation of our input. This is the most important conclusion of this work and the base on which we develop our own method.

The optimization approach we have opted for tries to consider attribution as a problem of objective and loss definition, focusing on *what* to accomplish instead of *how*. All the models defined in the first part of chapter 2 tried to specify a particular attribution strategy, with hopes that its results will be informative in the end. We consider that specifying the characteristics of the attribution beforehand and building models to accomplish those can be a more productive strategy. The first results of our method seem to suggest that there is a profitable path to explore here. Moreover, the last two papers discussed in the aforementioned chapter focus on this strategy too.

The problem is still open for new approaches and more precise explanations, but it is still hard to work on. The lack of a precise definition of explainability, of valid metrics for method comparison and the increasing complexity of Deep Learning models make this a demanding area. Nevertheless, its applications are doubtlessly very useful and even necessary, both for sensitive fields and for new legislative initiatives that start demanding the ”right of explanation”.

Our hope for this project is to motivate new approaches to the attribution problem, and ultimately, to the explainability problem, to finally lift the infamous ”black-box” label that Deep Learning is enduring. Our opinion is that these models can be as interpretable as any other given the right tools, and the progress of the explainability field will definitely help in the application of Deep Learning in many other areas that have not perceived its benefits yet.

Appendix A

Convolutional Neural Networks

In this appendix we will briefly describe what Convolutional Neural Networks (CNN) are and how and why they work.

Firstly proposed by LeCun in [19], CNNs were meant to solve a problem of excessive complexity in neural networks that work with image inputs. If we think about a gray-scale image input, like MNIST, we get a $W \times H$ pixel matrix that corresponds to $W \cdot H$ input features. If we used a linear layer (a layer of perceptrons), the number of parameters (without biases) for each output activation is $W \cdot H$, one for each input feature. Normally, however, neural networks would try to create several output activations per layer, so we require $d_l \cdot W \cdot H$ parameters for that single layer, where d_l is the number of output features for that layer. This number by itself is big enough, but if we also take into account that the power of neural networks is in their depth, this quantity is even more troublesome.

For the case of MNIST and just a single output unit, the layer would require $28 \cdot 28 = 784$ parameters. For CIFAR10, a RGB colour image, since there are three colour channels, the number of parameters of its 32×32 images is $32 \cdot 32 \cdot 2 = 3,072$. And, with bigger (and more common) images such as the ones in ImageNet, which are normally sampled to 256×256, $256 \cdot 256 \cdot 3 = 196,608$. There is clearly a scalability problem of this approach in terms of image size.

The problems that arise with a high number of trainable parameters are twofold. For starters, a bigger number of parameters implies bigger memory requirements and training time (more gradients to compute). However, the worse consequence of this parameter explosion is that the network VC complexity (its capability to shatter a certain number of points) is thereby increased too. This can easily lead to overfitting, which eventually affects the overall predictive capabilities of the network.

Regularization techniques could be applied to mitigate this complexity issue, but the resource problems would still be left. The idea in CNNs is to solve both problems in one single architectural assumption. The only parameter in a Convolutional layer is a **kernel matrix** of dimensions $C_i \times W_k \times H_k$, where C_i, W_k, H_k are the number of input channels and the width and height of the kernel matrix, respectively. Note that the kernel matrix is usually square and its size is very small (normally 3×3 or 5×5).

This kernel matrix is applied to the input image in an operation called **convolution** (hence, the name of Convolutional Neural Networks), which can be reformulated in a simple scalar product. The convolution operation multiplies, for each pixel, each of its neighboring pixel values and its own value by the kernel matrix own values, therefore computing a scalar product between the neighbors of a pixel and the kernel. The result is a single value corresponding to that pixel; joining all pixel computations we can recreate another image-like matrix, that is usually referred to as the *filtered image*.

Finally, this convolution kernel matrix creates a single filtered image, but CNNs usually include several kernel matrices stacked together, to output a multi-channel filtered image, where each channel is the result of a convolution operation with a specific kernel matrix. Therefore, kernel matrices are usually stacked together in a single $C_o \times C_i \times W_k \times H_k$, where C_o is the number of output channels resulting from the Convolutional layer.

Note that this operation bears resemblance to the classic perceptron or linear layer. Each output activation is computed as a scalar product between input and parameters. In the case of CNNs, however, the scalar product only takes into account the neighborhood of a pixel (therefore, all the remaining pixels would have weight 0). Additionally, all pixels share the same kernel matrix (weights) with which their corresponding activations are computed.

Two assumptions are considered in this single architectural change. Firstly, the only relevant input for the activation of a pixel is the neighborhood of that pixel. This means that the values of a pixel far away from the one we are considering are not important in describing the image features located in that pixel. Secondly, the kernel matrix is position-invariant. The filtered image pixel activations are computed independently of the pixel position. Therefore, we cannot learn features specific for each area in the image canvas. Instead, we learn position-invariant features, thereby regularizing the network while adapting to an intrinsic characteristic of images.

This approach, together with the advances in GPU-training, has revolutionized the field of Computer Vision and extended to other areas sensitive to this idea of neighborhood-aware activations, like Speech Recognition. Most of the popularity of current Deep Learning technologies is derived from the breakthrough in Gradient-based training with the Backpropragation algorithm and the breakthrough in image-input networks with CNNs.

Convolutional Neural Networks, however, do not exclusively use Convolution layers in their overall architecture. Several advances in many aspects of the network have also become essential for CNNs.

1. ReLU activation layers. The classic non-linearities used in Neural Networks were the sigmoid and tanh functions, but these operations normally resulted in gradient saturation and slower or insufficient training. The ReLU activation layer eliminated this problem, with significant leaps in performance. Alternatives to the ReLU activation function, like Leaky ReLU or eLU are also considered by the community.
2. Pooling operations. The result of Convolution layers is a filtered image with multiple channels and width and height very close to the original image input. Therefore, the network does not reduce the intermediate activation dimensions practically. This can be mitigated with striding convolutions, but another approach is to compute a pooling operation (normally MaxPool), an operation that summarizes a square of pixels into a single pixel, thereby reducing the size of the filtered image. These techniques are still very popular, but all-convolutional networks have been gaining interest overtime, as proposed by Springenberg in [28].
3. Dropout. Dropout is a regularization technique that randomly deactivates some activation neurons during training time. The result of this simple operation are networks with better generalization, counterintuitively. They can be understood as an ensemble approach, as argued by Goodfellow in [29].
4. BatchNormalization. This layer learns how to scale its input to the one requested by the gradient optimization method during training time. Despite its simple definition, its effect on overall performance and training speed has popularized them for many applications, not just CNNs.

Figure A.1 shows the architecture of LeNet-5, one of the first Convolutional Neural Networks proposed by LeCun in [19]. There we can see the application of stacked Convolutional-Subsampling layers, with three fully-connected layers that end the classifier. Figure A.2 shows the architecture for AlexNet, introduced by Krizhevsky ([30]) in 2012, the neural network that outperformed all prior competitors in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) by reducing the top-5 error from 26% to 15.3%. The network works with higher resolution images and consists of a deeper iteration of the previous CNN; this model showed that deeper architectures could effectively lead to better results.

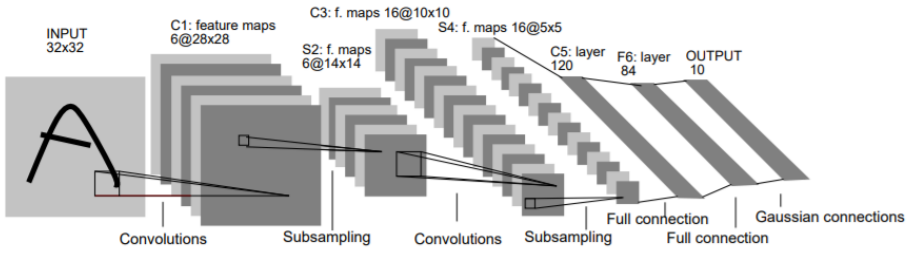


Figure A.1: LeNet-5 architecture, from [19]

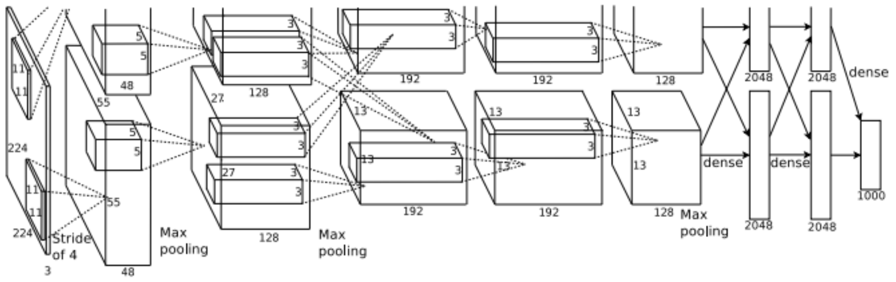


Figure A.2: AlexNet architecture, from [30]

Many other architectures have been proposed during recent years, like GoogleNet (Inception), VGGNet or the Resnet, and the predictive capabilities of these models have been increasing overtime with unprecedented leaps in performance. It is in this context of significant achievements that the focus has been shifting towards interpretability. Can we understand the mechanisms uncovered by these learning machines? This is what this project tries to answer.

There are multiple aspects to comment on all these techniques, but we cannot discuss them further due to size constraints. The interested reader can continue learning about these layers in [29].

Appendix B

Variational Autoencoders

In this second appendix we will overview the techniques known as Autoencoders; specifically, we will talk about what is known as Variational Autoencoders (VAE), proposed by Kingma in [25].

An autoencoder is a model that takes an input x and produces an output with the same dimensions as x, $\tilde{x}$, so that $\tilde{x}$ is a reconstruction of x. In the case of a neural network, we will train this model by minimizing a reconstruction loss (i.e., the L_2 norm of $\tilde{x} - x$).

This model by itself is not useful at all (we can just use the identity function, which would be the perfect autoencoder). The main characteristic that defines an autoencoder is that at some point in the middle of its computations, we must pass through a *bottleneck*. In the case of a sequential neural network autoencoder, this means that some intermediate layer (which we will call from now on the *embedding layer*) creates as output a low-dimensional representation of the input.

For example, if we have a MNIST image as input, a 28×28 grayscale image, and we pass it through an autoencoder of dimension 100, this means that the embedding layer (an intermediate, hidden layer) produces as output a 100-dimensional vector. This embedding layer divides the network in two. All previous layers, including the embedding layer, are called the **encoder**: a subnetwork responsible of producing what can be called the **embedding code**. The latter half of the network, called the **decoder**, takes that code and has to produce a reconstruction of the original input. Since the decoder only has that lower dimensional representation, that code needs to contain the most relevant information, so that the decoder can effectively reconstruct the input.

Therefore, what we do with an autoencoder is actually **dimensionality**

reduction. The hope is that the low-dimensional embedding contains only relevant information that describes effectively the input and can be processed by the decoder to reconstruct it. Note that if we use a sequential architecture and a differentiable reconstruction loss, we can use the usual optimization procedures to train these networks.

Variational Autoencoders propose a more specific idea. Proposed in [25] as a method for Probabilistic Graphical Models that try to describe latent random variables, this technique can be applied to Neural Networks, generating what is know as the Variational Autoencoder. The VAE assumes that the inputs it receives come from a 0-centered (multivariate) isotropic Gaussian distribution ($N(\mu, \Sigma = \sigma^2 I_d)$). Additionally, the encoder does not generate a single point belonging to that distribution. Instead, it generates two vectors of the same dimensionality of the embedding: the mean and standard deviation of another isotropic Gaussian whose elements are reconstructions of the input that generates those two codes.

At training time, any input generates the mean and std of its respective distribution, a point is sampled from that distribution and is used as the embedding code that the encoder takes as input to generate the reconstruction. At test time, we normally take the mean of that distribution as the actual code for the input.

This approach creates well-behaved embedding codes that can be used to sample new images similar to any input of your choice. In our project, we use the VAE to encode the data manifold of our dataset so that we can move in this manifold via the embedding code.

In order to train a VAE, we need to enforce the distribution constraint mentioned earlier. We use a KL-divergence loss, added to the reconstruction loss, to enforce that the embedding codes come from an isotropic Gaussian distribution with standard deviation 1.

Again, we will not go further in the explanation of this method, but the interested reader can refer back to [29] for more details.

Appendix C

Model architectures

In this short appendix we will include the definition of each model used along the project. We will use a list-of-tuples Python syntax, detailing layer by layer the definition for each model in the Pytorch framework. Additionally, we include the Flatten and Reshape layers, that respectively flatten or reshape (to the specified shape) all dimensions of the input except for the batch dimension.

MNIST

CNN

Listing C.1: MNIST CNN

```
nn.Conv2d(1, 16, kernel_size=5, stride=3, padding=2, bias=False),
nn.BatchNorm2d(16),
nn.ReLU(),
nn.Conv2d(16, 32, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(32),
nn.ReLU(),
nn.Conv2d(32, 64, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(64),
nn.ReLU(),
Flatten(),
nn.Linear(64 * 6 * 6, 128, bias=False),
nn.BatchNorm1d(128),
nn.ReLU(),
nn.Dropout(),
nn.Linear(128, 10)
```

VAE

Listing C.2: MNIST VAE

```
nn.Conv2d(1, 16, kernel_size=5, stride=3, padding=2, bias=False),
nn.BatchNorm2d(16),
nn.ReLU(),
nn.Conv2d(16, 32, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(32),
nn.ReLU(),
nn.Conv2d(32, 64, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(64),
nn.ReLU(),
Flatten()),
nn.Linear(64 * 6 * 6, 128, bias=False),
nn.BatchNorm1d(128),
nn.ReLU(),
nn.Dropout(),

nn.Linear(128, 100), # encoder mu
nn.Linear(128, 100), # encoder std

nn.Linear(100, 64 * 6 * 6),
nn.BatchNorm1d(64 * 6 * 6),
nn.ReLU(),
nn.Dropout(),
Reshape(-1, 64, 6, 6),
nn.ConvTranspose2d(64, 32, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(32),
nn.ReLU(),
nn.ConvTranspose2d(32, 16, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(16),
nn.ReLU(),
nn.ConvTranspose2d(16, 1, kernel_size=5, stride=3, padding=2, bias=False),
nn.BatchNorm2d(1),
nn.Sigmoid()
```

NotMNIST

The CNN and VAE for NotMNIST are the same than the ones for MNIST.

FashionMNIST

CNN

Listing C.3: FashionMNIST CNN

```
nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1, bias=False),
nn.BatchNorm2d(32),
nn.ReLU(),
nn.Conv2d(32, 64, kernel_size=3, stride=3, padding=1, bias=False),
nn.BatchNorm2d(64),
nn.ReLU(),
nn.Conv2d(64, 128, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.Conv2d(128, 128, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(128),
nn.ReLU(),
Flatten(),
nn.Linear(128 * 6 * 6, 128, bias=False),
nn.BatchNorm1d(128),
nn.ReLU(),
nn.Dropout(),
nn.Linear(128, 10)
```

VAE

Listing C.4: FashionMNIST VAE

```
nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1, bias=False),
nn.BatchNorm2d(32),
nn.ReLU(),
nn.Conv2d(32, 64, kernel_size=3, stride=3, padding=1, bias=False),
nn.BatchNorm2d(64),
nn.ReLU(),
nn.Conv2d(64, 128, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.Conv2d(128, 128, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.Conv2d(128, 256, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(256),
nn.ReLU(),
Flatten(),
nn.Linear(256 * 4 * 4, 128, bias=False),
nn.BatchNorm1d(128),
nn.ReLU(),
nn.Dropout(),

nn.Linear(128, 100), # encoder mu
nn.Linear(128, 100), # encoder std

nn.Linear(100, 256 * 4 * 4),
nn.BatchNorm1d(256 * 4 * 4),
nn.ReLU(),
nn.Dropout(),
Reshape(-1, 256, 4, 4),
nn.ConvTranspose2d(256, 128, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.ConvTranspose2d(128, 128, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.ConvTranspose2d(128, 64, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(64),
nn.ReLU(),
nn.ConvTranspose2d(64, 32, kernel_size=3, stride=3, padding=1, bias=False),
nn.BatchNorm2d(32),
nn.ReLU(),
nn.ConvTranspose2d(32, 1, kernel_size=3, stride=1, padding=1, bias=False),
nn.BatchNorm2d(1),
nn.Sigmoid()
```

CIFAR10

CNN

Listing C.5: CIFAR10 CNN

```
nn.Conv2d(3, 64, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(64),
nn.ReLU(),
nn.Conv2d(64, 64, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(64),
nn.ReLU(),
nn.MaxPool2d(2),
nn.Conv2d(64, 128, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.Conv2d(128, 128, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.MaxPool2d(2),
nn.Conv2d(128, 256, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(256),
nn.ReLU(),
Flatten(),
nn.Linear(256 * 3 * 3, 256, bias=False),
nn.BatchNorm1d(256),
nn.ReLU(),
nn.Dropout(),
nn.Linear(256, 128, bias=False),
nn.BatchNorm1d(128),
nn.ReLU(),
nn.Dropout(),
nn.Linear(128, 10)
```

VAE

Listing C.6: CIFAR10 VAE

```
nn.Conv2d(3, 64, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(64),
nn.ReLU(),
nn.Conv2d(64, 128, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.MaxPool2d(2),
nn.Conv2d(128, 256, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(256),
nn.ReLU(),
nn.Conv2d(256, 128, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.Conv2d(128, 128, kernel_size=5, stride=3, padding=2, bias=False),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.Conv2d(128, 256, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(256),
nn.ReLU(),
Flatten(),
nn.Linear(256 * 2 * 2, 128, bias=False),
nn.BatchNorm1d(128),
nn.ReLU(),
nn.Dropout(),
nn.Linear(128, 100), # encoder mu
nn.Linear(128, 100), # encoder std

nn.Linear(100, 256 * 2 * 2),
nn.BatchNorm1d(256 * 2 * 2),
nn.ReLU(),
nn.Dropout(),
Reshape(-1, 256, 2, 2),
nn.ConvTranspose2d(256, 128, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.ConvTranspose2d(128, 128, kernel_size=5, stride=3, padding=2, bias=False),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.ConvTranspose2d(128, 256, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(256),
nn.ReLU(),
nn.ConvTranspose2d(256, 128, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(128),
nn.ReLU(),
nn.Upsample(scale_factor=2),
nn.ConvTranspose2d(128, 64, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(64),
nn.ReLU(),
nn.ConvTranspose2d(64, 3, kernel_size=3, stride=1, bias=False),
nn.BatchNorm2d(3),
nn.Sigmoid()
```

Bibliography

[1] Aylin Caliskan, Joanna J. Bryson, and Arvind Narayanan. Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334):183–186, 2017.

[2] Zachary C. Lipton. The Mythos of Model Interpretability. 2016. arXiv:1606.03490 [cs.LG].

[3] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. 2013. arXiv:1312.6199 [cs.CV].

[4] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature Visualization. Available at `https://distill.pub/2017/feature-visualization/`.

[5] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T. Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un)reliability of saliency methods. 2017. arXiv:1711.00867 [stat.ML].

[6] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. 2017. arXiv:1703.01365 [cs.LG].

[7] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The Building Blocks of Interpretability. Available at `https://distill.pub/2018/building-blocks/`.

[8] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal 1341.3*, 2009.

[9] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June:5188–5196, 2015.

[10] Mike Tyka. Class visualization with bilateral filters. Available at `https://mtyka.github.io/deepdream/2016/02/05/bilateral-class-vis.html`.

[11] Anh Nguyen, Jason Yosinski, and Jeff Clune. Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks. 2016. arXiv:1602.03616 [cs.NE].

[12] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8689 LNCS(PART 1):818–833, 2014.

[13] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. SmoothGrad: removing noise by adding noise. 2017. arXiv:1706.03825 [cs.LG].

[14] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Gradients of Counterfactuals. 2016. arXiv:1611.02639 [cs.LG].

[15] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 10(7), 2015.

[16] Jianming Zhang, Sarah Adel Bargal, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-Down Neural Attention by Excitation Backprop. *International Journal of Computer Vision*, 2017.

[17] Zhongang Qi, Saeed Khorram, and Fuxin Li. Embedding Deep Networks into Visual Explanations. 2017. arXiv:1709.05360 [cs.CV].

[18] Ruth C. Fong and Andrea Vedaldi. Interpretable Explanations of Black Boxes by Meaningful Perturbation. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-October:3449–3457, 2017.

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998.

[20] Yaroslav Bulatov. notMNIST dataset. Available at `http://yaroslavvb.blogspot.com.es/2011/09/notmnist-dataset.html`.

[21] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. 2017. arXiv:1708.07747 [cs.LG].

[22] Alex Krizhevsky and G Hinton. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 2010.

[23] Wojciech Samek, Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, and Klaus Robert Müller. Evaluating the visualization of what a deep neural network has learned. *IEEE Transactions on Neural Networks and Learning Systems*, 28(11):2660–2673, 2017.

[24] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for Deep Neural Networks. *ICLR Submission*, 2018. arXiv:1711.06104 [cs.LG].

[25] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. 2013. arXiv:1312.6114 [stat.ML].

[26] Ij Goodfellow, J Pouget-Abadie, and Mehdi Mirza. Generative Adversarial Networks. 2014. arXiv:1406.2661 [stat.ML].

[27] Jun-yan Zhu, Taesung Park, and Alexei A Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. 2017. arXiv:1703.10593 [cs.CV].

[28] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for Simplicity: The All Convolutional Net. 2014. arXiv:1412.6806 [cs.LG].

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. *Cambridge: MIT press*, 1, 2015.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, 2012.