Generating counterfactuals using variational autoencoders for model interpretability

Alexandre Santangelo* Natasha Tagasovska†

Abstract

Capable of accomplishing a variety of tasks with an ever increasing accuracy, machine learning (ML) models have been growing in popularity and entering many domains of science as well as our everyday life. Nevertheless, the way in which they accomplish these tasks remain mostly a mystery. This lack of interpretability leads to errors, bias and incomprehension of our models. In order to shed some light into their behaviour, we propose the use of *counterfactuals*: altered data samples which may help interpreting the ML model decision process. In this thesis we introduce a method to generate counterfactuals using Conditional variational autoencoders. This method allows us to modify the original data samples from a certain class, by "mixing" them with another class from the dataset (for example, decoding a picture of a cat as if it was a dog) thus, producing a counterfactual. The resulting counterfactual is then classified. Using the classification result, the discrepancy between the original sample and the counterfactual, we can better interpret our model classification method. Our method for generating counterfactuals has been successfully tested on two models and image datasets. We hope further research will generalize its use to other models and datasets.

Keywords: counterfactuals, variational autoencoder, interpretability

1 Introduction

Machine learning models are currently achieving a high level of accuracy for a variety of tasks. While those results are encouraging, ML models confidence rates and the interpretability of their solutions are still very open research areas that hold many questions. Answers to those could help us shed some light on how ML models make decisions. Understanding these hidden mechanisms will be the next big step in ML, helping us in achieve better results overall.

This research is being conducted at EPFL (École Polytechnique Fédérale de Lausanne) under the supervision of Natasha Tagasovska working at the Swiss Data Center. This is the result of my bachelor thesis after a work of three months.

*a.santangeloibanez@gmail.com

†natasa.tagasovska@epfl.ch

Motivation Counterfactuals are useful in the interpretation of ML models. They are, in essence, a slightly modified version of an original sample taken from the studied dataset that will be classified differently by the model. The difference between the counterfactual and the original sample helps us understand how our model makes decisions.
Our objective is to develop a methodology for generating counterfactuals for any model and dataset. The method we propose in our paper consists in building a CVAE (Conditional variational autoencoder) with a very similar architecture to that of the model (in our case: classifier) being interpreted. We then use such CVAE to generate counterfactuals by choosing an original sample from a certain class distribution (i.e a picture of a cat from the dataset) and decoding the sample under a different distribution (i.e. decoding the picture of a cat as if it was a dog). The generated counterfactual conserves characteristics of the original class while adding features of a different class (We could find a cat with a elongated snout that could make it look like a dog and is classified as a dog by the model). Not all generated counterfactuals are of the same quality, that is why we developed a metric which indicates us just how good the counterfactual is. We are hoping the method will easily generalize to all sorts of models and datasets. This could be one of the ways to shed a light on the dark world of ML models hidden techniques.

The rest of the thesis is organized as follows: Section 2 will introduce the concept of counterfactuals explanations, one of the key concepts of this thesis.
Section 3 introduces our generative model, the Variational autoencoder, with a brief explanation of the model architecture. Subsection 3.1 briefly dives in the math behind our model and subsection 3.2 introduces Conditional variational autoencoders, a kind of VAE that better fits our need.
Section 4 introduces our method for computing aforementioned counterfactuals, taking a look at two methods: ENC-DEC and COND-MIX in sections 4.1 and 4.2 respectively. Section 4.3 explains our metric in detail as it is key to understanding the purpose of this thesis.
Section 5 presents the results obtained with both techniques and discusses their advantages. Finally, section 6 discusses what should be done in the near future to further improve our method.

(A) Original sample, 4

(B) Counterfactual of sample

(C) Sample from 9's distribution

2 Counterfactuals

Counterfactual explanations describe events of the form "If X hadn't occurred then Y wouldn't have occurred" [1] . Humans typically rely on this causal way of thinking when trying to understand things, it is not an unfamiliar explanation process for us. Applying this to Machine learning translates to slightly perturbing the inputs of the model and observing the classification change. We can then observe the effect of each input or group of inputs to the classification result, these observations give us insights that help us interpret the inside mechanisms of the model.

2.1 Input modification

Modifying the input is not a simple task, especially because the input features might not always be interpretable as straightforward characteristics (as when the inputs correspond to the age of a person, employment or annual salary) but, instead N-dimensional inputs (tensors, vectors) that represent an image through pixel values. Modifying a pixel might change the model prediction, but it will most certainly not be interpretable.
This is why the modification of an input in an informed manner that can later serve interpretation purposes is crucial. This is the objective of this thesis. In the later sections we will show how our counterfactual generation method corresponds to this task.

3 Variational autoencoders

Neural networks are one of the most promising ML models today, hence, it shouldn't come as a surprise that the most popular generative models (GAN, VAE,...) rely on neural networks.
Before introducing variational autoencoders, we should understand autoencoders.
Autoencoders are ML models composed by an encoding net Φ and a decoding net Ψ. Their goal is to reconstruct the high dimensional data: $X \in \mathrm{R}^d$ which they are given as input after encoding it, using Φ, in a latent space of lower dimension: $z \in \mathrm{R}^m, m << d$ and then decoding the encoded sample, z using Ψ to output $X' \in \mathrm{R}^d$. During training, the autoencoder is given an input $x \in \mathrm{R}^d$ and outputs its reconstruction: $x' \in \mathrm{R}^d$. Some loss function (depending on the kind of input) is then used over the input and output $loss(x, x')$ in order to minimize $\delta = x' - x$.

In order to generate data ($X \in \mathrm{R}^d$), a process that is not so unfamiliar to us humans, as we do it everyday when imagining or remembering things, variational autoencoders rely on a latent space $z \in \mathrm{R}^m, m << d$ produced by an encoding net Φ and a decoding net Ψ.
Variational autoencoders diverge from regular autoencoders because of how Φ encodes data to z. In VAEs, each feature in the latent space of is normally distributed, making their latent space more continuous, allowing for smoother transitions between classes. This characteristic helps us better sample points z_i to generate new data $x_i = \Psi(z_i)$ which follow

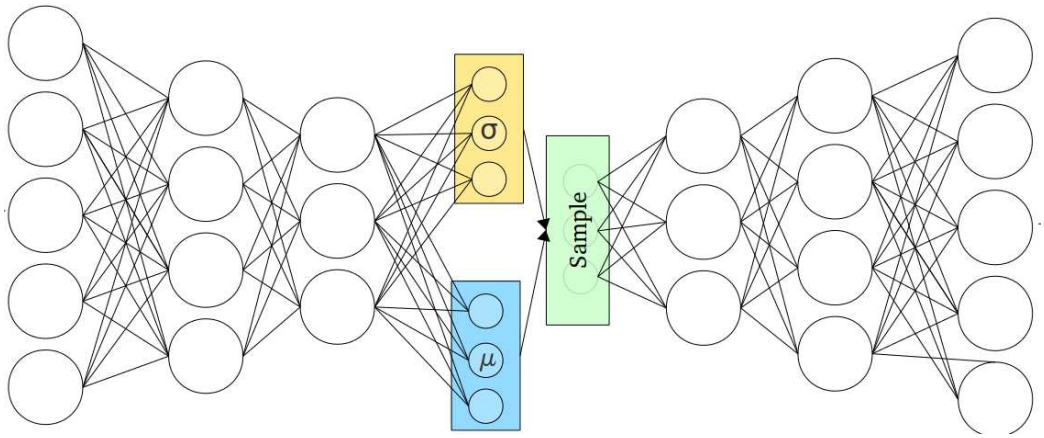


FIGURE 2: Variational autoencoder architecture

the original data X distribution.
This process is similar to human imagination, our brain encodes what we see (the data) as concepts (the latent space features) inside our heads, and then decodes them easily. Long neck animal, yellow fur, dark spots, Africa and that's it you're seeing a giraffe ! With details that I did not give to you, only with broad concepts and your brain. Much like your brain just did, we would like our model to only keep the important *meaningful and interpretable* features of X in z and reproduce the original data correctly. Alas, ML models don't always store in z meaningful information that we could *interpret as concepts*, but rather encode what they consider important. Nevertheless, some of the encoded features are somewhat *human friendly concepts* which we can interpret.

3.1 Objective function

In order to generate data from X we should have access to $P(X)$: the underlying distribution of our data. To learn $P(X)$ we need $P(Z)$ and $P(X|Z)$. $P(Z)$ will be inferred from $P(Z|X)$, which makes sense as we want our latent space to be likely under the assumption of our data. But we also need to infer $P(Z|X)$ which we do using variational inference by modeling the true $P(z|X)$ distribution under a simpler (Gaussian) distribution $Q(z|X)$. We then use the Kullback–Leibler divergence as a metric to know how different P and Q are. By inferring $P(Z|X)$ from $Q(Z|X)$ we formulate and rearrange the KL divergence to arrive at the objective function :

$$\log P(X) - D_{KL}[Q(z|X)\|P(z|X)] = E[\log P(X|z)] - D_{KL}[Q(z|X)\|P(z)]$$

3.1.1 Interpretation

The VAE objective function has a nice interpretation. We are trying to find $P(X)$ by finding the lower bound of $\log P(X)$ under some error $D_{KL}[Q(z|X)\|P(z|X)]$ from the inference

by using the maximum likelihood estimation of $\log P(X|z)$ and minimising the difference between $Q(z|X)$ and $P(z)$.

Since Gaussian distributions are entirely defined by their first and second moments and can be sampled from using a *reparametrization trick* which adds random noise ϵ to our system, making z differentiable, we chose to model Q as a Gaussian distribution by setting $P(z) = N(0, 1)$. We can write the KL divergence term as :

$$D_{KL}[N(\mu(X), \Sigma(X))\|N(0, 1)] = \frac{1}{2}\sum_{k}\Big(\exp(\Sigma(X)) + \mu^2(X) - 1 - \Sigma(X)\Big)$$

.

3.2 Conditional VAE

Being able to condition our generative model allows us to have more control over the data generation process. Our goal is to, as if we were using our brain, generate data based on conditionals (which are a translation of human concepts to machine language).
This is why we use CVAEs (Conditional variational autoencoders). They operate exactly as VAEs but rely on an extra input : the conditional c, which is concatenated with the original input to enter the encoding net and concatenated with the latent vector z to enter the decoding net.

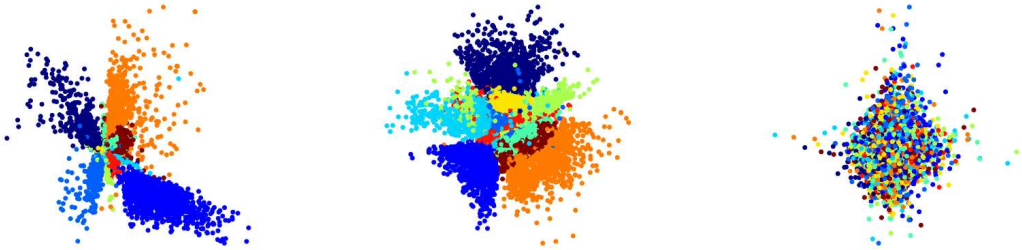
3.2.1 Conditional

ML models operate on very high level concepts which are constituted by the interaction of millions of parameters and aren't understandable to us humans, which leads to the lack of interpretability at stake.
To facilitate the translation from high level computer concepts to human friendly and interpretable ideas we rely on counterfactuals. A simple example of a counterfactual was our way of communicating to our CVAE model working on MNIST. We expressed numbers as a one hot encoded vector : a 9 became [0000000001]

3.2.2 Objective function

The addition of this conditional to our model facilitates its task as it now has more information. The encoding net $P(z|X)$ becomes $P(z|X, c)$ and the decoding net $P(X|z)$ becomes $P(X|z, c)$.
The objective function becomes :

$$\log P(X|c) - D_{KL}[Q(z|X, c)\|P(z|X, c)] = E[\log P(X|z, c)] - D_{KL}[Q(z|X, c)\|P(z|c)]$$



(A) Autoencoder (B) Variational autoencoder (C) Conditional VAE

FIGURE 3: Plot of different models latent spaces colored by class

4 Method for generating counterfactuals

In order to generate counterfactuals using variational autoencoders we considered two techniques that used the properties of VAEs, i.e, the continuous latent space and the conditional controlled generation of data.
As we are generating counterfactuals for model interpretability, the variational and conditional aspect of our CVAE must be used. The normalisation of the latent space features allows for a certain independence between the features, it also brings the classical properties of a normal distribution, i.e, the latent vectors lying around the vector $\vec{0}$, are decoded into the most representative samples of the conditional class.
The conditional aspect of the CVAE allows us to decode latent vectors and encode data with certain conditionals that might not always correspond to the label, even create new conditionals by mixing already established ones. These approaches will explored in the following sections.

4.1 ENC-DEC method

This method, inspired from [3], relies both on the encoding and decoding part of the CVAE as follows :

x_i : original data sample

c_i : conditional corresponding to the label of the original data sample

z_i : The latent vector $\Phi(x_i, c_i')$

x_i' : the generated counterfactual, $\Psi(x_i', c_i')$.

c_i' : conditional different from c_i.

We first encode x_i the original data sample with a conditional $c_i' \neq c_i$.

$$z_i = ENC(x_i, c_i') = Phi(x_i, c_i')$$

We then decode z_i with c'_i, obtaining our counterfactual x'_i.

$$x'_i = DEC(z_i, c'_i) = \Psi(x'_i, c'_i)$$

4.1.1 Interpretation

The counterfactual x'_i resembles x_i up to some point depending on the latent space dimension size, which is directly correlated with how similar x_i and x'_i are. This makes sense, as with a bigger latent vector the encoder Φ can encode more information in z_i and the decoder Ψ obtains more information.
The encoding process receives as input a concatenation of : $x \in \mathrm{R}^d$, a high dimensional data sample, and $c'_i \in \mathrm{R}^m, m << d$: a typically low dimensional conditional. The encoder picks up more information from x_i , which it encodes in the latent features of z_i, but c'_i gives it information about how to encode this sample, about what distribution to mimic.
The conditional is key, as it specifies where features from x_i (eye color, face shape, hair type... or sometimes features that are not human friendly concepts) should be encoded in the latent vector.
The decoder, Ψ, guided by the conditional c'_i, (which specifies the distribution of the latent space), receives as input a concatenation of z_i and c'_i to generate the counterfactual.
The decoder is typically more influenced by the conditional than the encoder is. That is why a good latent vector z_i, with an adequate dimension making it capable of keeping a good amount of information about x_i is key. If the dimension is too low, x'_i won't have any similarities with x_i and won't be useful; but if the dimension is too high, the model will simply replicate the input x_i by saving all its information in z_i during the training, making it no longer a generative model.

4.2 COND-MIX method

This methods principally relies on the *conditional* aspect of the autoencoder. In this case the counterfactual does not depend on the latent space as much as with the ENC-DEC method.

x_i : original data sample

c_i : conditional corresponding to label of original data sample

z_i : latent space representation $\Phi(x_i, c_i)$

c'_i : conditional different from c_i.

c^*_i : conditional resulting from the mix of c'_i and c_i. $(c' \star c_i)$

x'_i : the generated counterfactual $\Psi(z_i, c^*_i)$.

We first need to produce our conditional c^*_i by mixing c'_i and c together $(c'_i \star c)$ in such a way that the model will interpret it as both concepts, sampling from both conditional

distributions. This can be done by simply adding the counterfactuals (as we did for MNIST, where $c_i = [0000100000]$, $c'_i = [0000000001] \implies c^*_i = c'_i \star c_i = [0000100001]$).

$$c^*_i = c'_i \star c_i$$

We then produce z_i by encoding x_i,the original data sample, with c^*_i :

$$z_i = ENC(x_i, c^*_i) = \Phi(x_i, c_i)$$

And produce x'_i by decoding z_i with c^*_i :

$$x'_i = DEC(z_i, c^*_i) = \Psi(z_i, c^*_i)$$

4.2.1 Interpretation

The encoding part is a bit more complicated in this method as the encoder Φ distributes x_i's features according to two distributions (c_i and c'_i) at once, as it has never learned to build z_i from c^*_i during its training. Here the counterfactual x'_i will conserve features from x_i because of c_i and pick up features from c'_i. The traits encoded in z_i help smooth the transition from conditional c_i to c'_i by making the label distribution c_i have more influence than c'_i distribution.
The decoder Ψ is going generate x'_i by sampling from the two distributions equally. The counterfactual will conserve more features from x_i than it will pick from c'_i as, even if Ψ distributed x_i characteristics along two distributions c_i and c'_i equally, x_i features are more easily distributed along c_i than c'_i, giving x'_i more features from x_i.

4.3 Metric

Counterfactuals need to meet certain criteria in order to be used for model interpretability [2]:

1. The difference : $\delta = x'_i - x_i$ between x_i and x'_i should be minimal.
2. x'_i should lie close to the dataset distribution.
3. The difference : $\delta = x'_i - x_i$ between x_i and x'_i should be sparse.

4.3.1 Minimal difference

Minimising $\delta = x'_i - x_i$ is essential for the counterfactual to be interpretable.
As we need to find the minimum set of differences that cause the model to classify x'_i differently than x_i. This set of differences will give us insights on what the model considers key to difference one class from the other.
In order to measure this difference we rely on the binary cross entropy between x_i and x'_i : $BCE(x_i, x'_i)$ which performs correctly when normalized:

$$BCE(x'_i, x_i) - \mu(BCE(x_i, Repr(c'_i)))$$

Where the term $\mu(BCE(x_i, Repr(c'_i)))$ represents the average binary cross entropy between x_i's features distribution and the feature distribution of a data sample representing the conditional c'_i. Without this regulation, the value depends too much on the conditional distribution and the original data sample, introducing a bias.

4.3.2 Belonging to distribution

The generated counterfactual x'_i shouldn't be too different from the rest of the data. That is, we want our counterfactual to be as close as possible to both c_i and c'_i distributions. We would like to find known features of the dataset in our difference $\delta = x'_i - x_i$. The difference could then be interpreted in function of which conditional or known feature has changed the most (i.e, the difference could be a change in eye color, face aging, etc... Or, again, something not interpretable).
In order to measure the belonging of x'_i to each distribution, we simply compute the KL divergence between the encoding of x'_i with Φ and a normal distribution.

$$KLD(\Phi(x'_i, c'_i), N(0,1)) + KLD(\Phi(x'_i, c'), N(0,1))$$

We compute two KLDs, one with respect to each distribution, and minimize both. These terms make use of the normally distributed features, if the KLD is high, it indicates that the data point lies far from the mean of the distribution.
We also rely on two additional terms : the reconstruction loss between the counterfactual and reconstruction of the counterfactual with both distributions.
$RCL(x, c) = BCE(x, DEC(ENC(x, c), c))$

$$RCL(x'_i, c_i) + RCL(x'_i, c'_i)$$

These two terms are helpful, as the reconstruction loss has been minimized in training (loss function of the autoencoder), if the reconstruction loss is high for some data point it indicates that the model hasn't been trained on similar data, i.e, the data point does not belong to the distribution.

4.3.3 Sparse difference

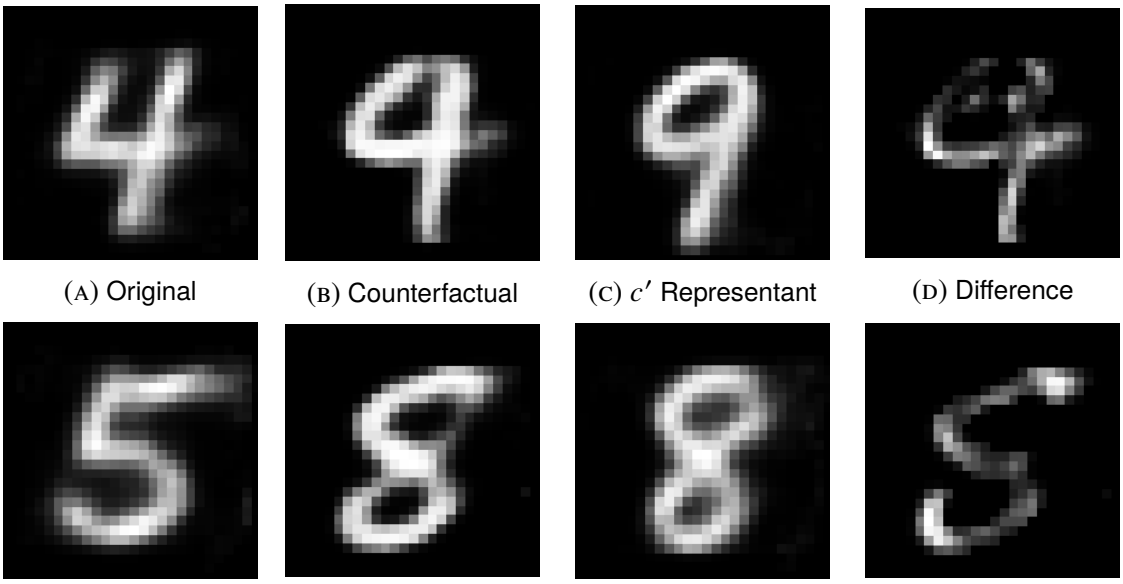
The difference should be sparse, meaning there should only be spread out local changes, i.e, we don't want to blur the sample, but rather change some specific features that will flip the decision of our classifier. Our method meets this metric as it mixes probabilistic distributions, preventing random spread out changes by only modifying the features of the data that follow the distribution.

5 Results

We tried our method on two datasets: MNIST and Colored MNIST.
Using a classifier model for each dataset and then building a CVAE by copying the classifier's architecture, we generated counterfactuals for each dataset and model.

5.1 MNIST

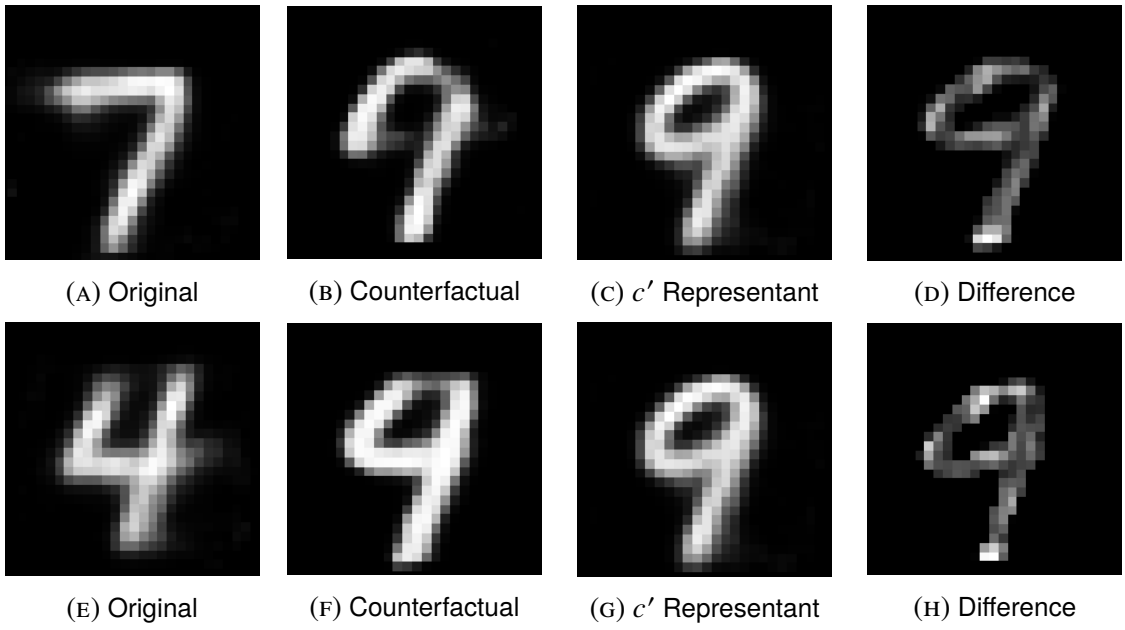
MNIST is a dataset of 40.000 28x28 images of handwritten digits.
We constructed a Conditional variational autoencoder for MNIST using a neural network for the encoding and decoding layers. We first observed the latent space and compared it to the latent space of an autoencoder and a variational autoencoder. The CVAE's latent space was much more continuous than that of the autoencoder (Figure 3). Our CVAE model's latent space can be seen as the superposition of 10 (As MNIST has ten classes) normal distributions, the model generates data by sampling from a distribution provided by the conditional. The set of conditionals are one-hot encoded vectors representing the ten classes (3 = [0001000000]). The conditional $c_i \in \mathrm{R}^{10}$ is concatenated with $x_i \in \mathrm{R}^{784=28x28}$ forming our input.
In order to compute counterfactuals for one data sample x_i, we first pick a list of conditionals $C = c_0, c_1, ..., c_n$ (digits from 0 to 9 per se) and then apply our method (ENC-DEC or COND-MIX) on x_i for each conditional c_i. This process is repeated multiples times in order to produce a list of counterfactuals which is then sorted according to our metric.



(A) Original (B) Counterfactual (C) c' Representant (D) Difference

(E) Original (F) Counterfactual (G) c' Representant (H) Difference

FIGURE 4: Results from COND-MIX on MNIST



(A) Original (B) Counterfactual (C) c' Representant (D) Difference

(E) Original (F) Counterfactual (G) c' Representant (H) Difference

FIGURE 5: Results from ENC-DEC on MNIST

5.2 Colored MNIST

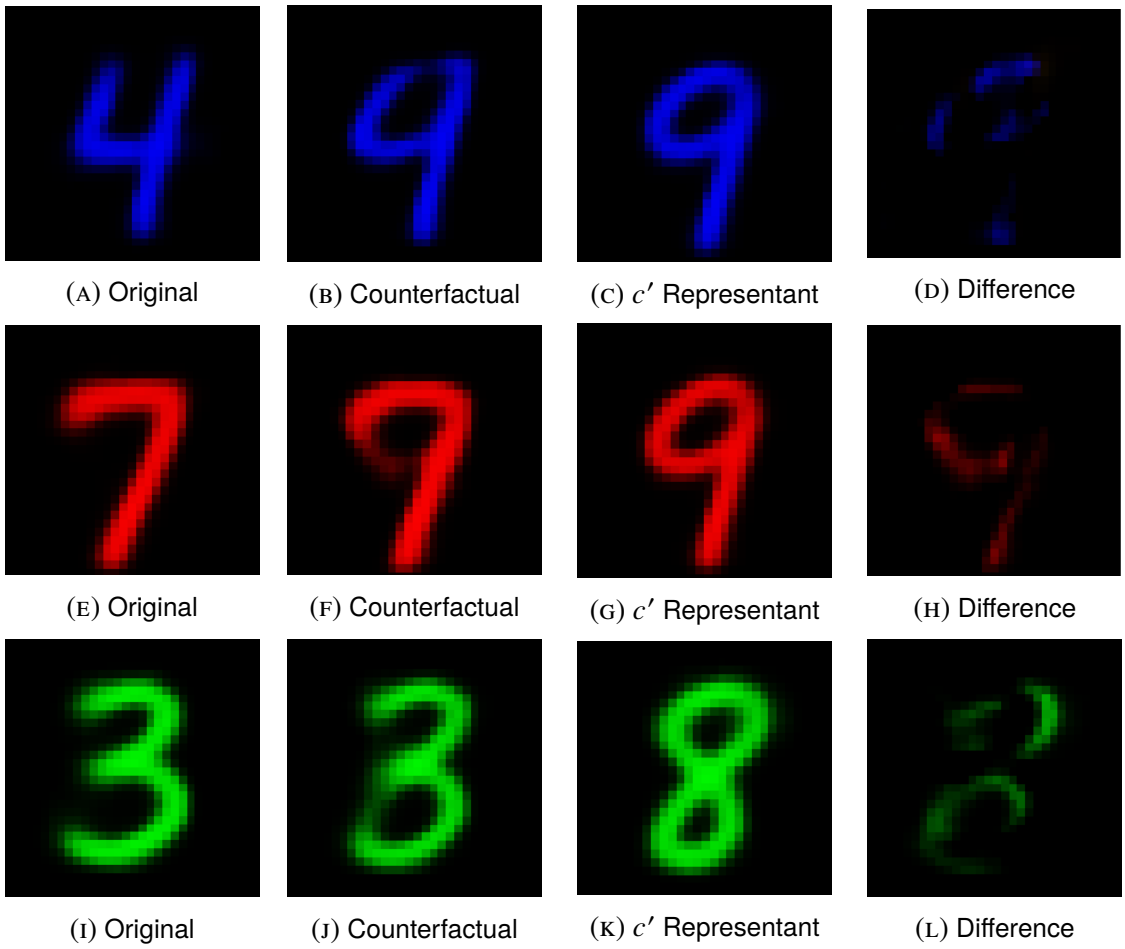
Colored-MNIST is a toy dataset built from MNIST, our dataset was built by colouring 1/3 of the numbers red, 1/3 green and 1/3 blue. Our idea was to study a dataset more complicated than MNIST, having 30 classes instead of 10. We used the same method as in MNIST to build our conditionals (one hot encoding of the number) and concatenated it with a one hot encoded vector of the colour. A red 5 would be encoded as [0000010000100] . As our input is a 28x28x3 tensor, we use a CNN that outputs a 256 dim vector which is then concatenated with our conditional. It is then fed to a neural network which encodes it as z. The encoding is then fed to a NN which output is fed to a CNN to reconstruct the original image. After training the CNN classifier and our CVAE on Colored-MNIST, we started generating counterfactuals using our two methods.

5.2.1 COND-MIX

We were able to obtain satisfying results on Colored-MNIST. The difference between the images is sparse and successfully identifies the most important aspects that differentiate one class from the other.

5.2.2 ENC-DEC

This method didn't yield satisfying results on colored MNIST even though the results on MNIST were good, we think it is because our MNIST model has a NN architecture and concatenates the conditional with a 784 dim vector. But for Colored-MNIST the input is



(A) Original (B) Counterfactual (C) c' Representant (D) Difference

(E) Original (F) Counterfactual (G) c' Representant (H) Difference

(I) Original (J) Counterfactual (K) c' Representant (L) Difference

FIGURE 6: Results from COND-MIX on Colored-MNIST

a 28x28x3 tensor which is fed to a CNN. The CNN then outputs a 256 dimension vector which is concatenated with our conditional.

6 Discussion

The present results obtained by using our method are very promising. The method has shown a great potential to modify images along conditionals. In our context these modifications were aimed at interpretable purposes, generating counterfactuals, but the method could just as easily be used to generate adversarial examples.

Nevertheless, the method is still at an early stage and all its potential still needs to be tested. Our next step on this project, have we had more time, would have been to test our method on biased datasets to see if it could generate counterfactuals that would point towards that bias (We briefly tested this on Colored MNIST and the outcome seemed positive, but we did not have definitive results). This method, counterfactuals in general, could be key to unlocking a whole new area of machine learning, in which we would learn from our models. That

is, training a classifier on a complex task and, using interpretable techniques, interpret the classifier way of thinking to learn more about this complex task.
All in all, this paper made me realize just how deep machine learning goes, that whole research areas are yet to be explored and that we still have much to learn about ML and artificial intelligence in general.
Machine learning seems to be a field where the depth of one's knowledge doesn't get him closer to the end, but further away.

References

[1] *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable.* 2021-01-18.

[2] *Interpretable Counterfactual Explanations Guided by Prototypes* arXiv:1907.02584v2 [cs.LG]

[3] *Explaining Classifiers with Causal Concept Effect (CaCE)* arXiv:1907.07165v2 [cs.LG] 28 Feb 2020

[4] *Auto-Encoding Variational Bayes*. arXiv:1312.6114v10 [stat.ML]

[5] *Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV)*