A. Appendix

A.1. Causality Preliminaries

In this section, we review some of the basic definitions in causality that may help understand this work.

Structural Causal Models (SCMs) (Pearl, 2009) provide a rigorous definition of cause-effect relations between different random variables. Exogenous variables (noise) are the only source of stochasticity in an SCM, with the endogenous variables (observables) deterministically fixed via functions over the exogenous and other endogenous variables.

Definition A.1. (Structural Causal Models). A Structural Causal Model is a 4-tuple $(X, U, f, P_u))$ where, (i) X is a finite set of endogenous variables, usually the observable random variables in the system; (ii) U is a finite set of exogenous variables, usually treated as unobserved or noise variables; (iii) f is a set of functions $[f_1, f_2, ...f_n]$, where n refers to the cardinality of the set X. These functions define causal mechanisms, such that $\forall x_i \in X, x_i = f_i(Par, u_i)$. The set Par is a subset of $X - \{x_i\}$ and $u_i \in U$. We do not consider feedback causal models here; (iv) P_u defines a probability distribution over U. It is not necessary for every node in an SCM to have a unique/shared noise. Deterministic causal systems have been considered in literature (Daniusis et al., 2010).

An SCM $M(X, U, f, P_u)$ can be trivially represented by a directed graphical model $G = (V, E)$, where the vertices V represent the endogenous variables X (each vertex v_i corresponds to an observable x_i). We will use random variables and vertices interchangeably henceforth. The edges E denote the causal mechanisms f. Concretely, if $x_i = f_i(Par, u_i)$ then $\forall x_j \in Par$, there exists a directed edge from the vertex v_j corresponding to x_j to the vertex v_i corresponding to x_i. The vertex v_j is called the *parent* vertex while the vertex v_i is referred to as the *child* vertex. Such a graph is called a **causal Bayesian network**. The distribution of every vertex in a causal Bayesian network depends only upon its parent vertices (local Markov property) (Kiiveri et al., 1984).

A *path* is defined as a sequence of unique vertices $v_o, v_1, v_2, ..., v_n$ with edges between each consecutive vertex v_i and v_{i+1}. A collider is defined with respect to a path as a vertex v_i which has a $\rightarrow v_i \leftarrow$ structure. (The direction of the arrows imply the direction of the edges along the path.) *d-separation* is a well-studied property of graphical models (Pearl, 2009; Geiger et al., 1990) that is often used to decipher conditional independences between random variables that admit a probability distribution faithful to the graphical model.

Proposition 4. (Pearl, 2009) Two random variables a and b are said to be conditionally independent given a set of random variables Z if they are *d-separated* in the corresponding graphical model G.

Definition A.2. (d-separation). Two vertices v_a and v_b are said to be *d-separated* if all paths connecting the two vertices are "blocked" by a set of random variables Z.

A path is said to be "blocked" if either (i) there exists a *collider* that is not in $Anc(Z)$, or, (ii) there exists a *non-collider* $v \in Z$ along the path. $Anc(Z)$ is the set of all vertices which exhibit a *directed path* to any vertex $v \in Z$. A *directed path* from vertex v_i to v_j is a *path* such that there is no incoming edge to v_i and no outgoing edge from v_j.

The $do(.)$ operator (Definition 4.1)(Pearl, 2009; 2012) is used to identify causal effects from a given SCM or causal Bayesian network. Although similar in appearance to the conditional expectation $\mathbb{E}(y|x = 1)$, $\mathbb{E}(y|do(x) = 1)$ refers to the expectation of the random variable y taken over its interventional distribution $P(y|do(x) = 1)$.

Definition A.3. (Average Causal Effect). The Average Causal Effect (ACE) of a binary random variable x on another random variable y is commonly defined as $\mathbb{E}(y|do(x = 1)) - \mathbb{E}(y|do(x = 0))$.

Formally, a causal Bayesian network $G = (V, E)$ induces a joint distribution over its vertices $P_V = \prod_{v_i \in V} P(v_i|parents(v_i))$. Performing interventions on random variables X_i are analogous to surgically removing incoming edges to their corresponding vertices V_{X_i} in the network G. This is because the value of the random variables X_i now depend on the nature of the intervention caused by the "external doer" and not the inherent causal structure of the system. The interventional joint distribution over the vertices of G would be $P_{(V|do(V_{X_i}))} = \prod_{v_i \in V - V_{X_i}} P(v_i|parents(v_i)))$. Notice that in $P_{(V|do(V_{X_i}))}$, the factorization of the interventional joint distribution ignores the intervened random variables X_i. In an SCM $M(X, U, f, P_u)$, performing a $do(x = x')$ operation is the same as an intervened SCM $M^i(X, U, f^i, P_u)$, where the causal mechanism f_x for variable x, is replaced by the constant function x'. f^i is obtained from the set f by replacing all the instances of random variable x in the arguments of the causal functions by x'.

A.2. More on Prior Work

Existing methods for attribution can broadly be categorized into gradient-based methods and local regression-based methods.

As stated in Sections 1 and 2 (main paper), in the former approach, gradients of a function are not ideal indicators of an input feature's influence on the output. Partial derivatives of a continuous function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ are also functions $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$ over the same domain $\mathbb{R}^n$ (the subscript i

denotes the partial derivative with respect to the i^{th} input feature). The attribution value of the i^{th} feature which is derived from g_i would in turn be biased by the values of other input features. For instance, consider a simple function $f : \mathbb{R}^2 \rightarrow \mathbb{R}$, $f(x_1, x_2) = x_1 x_2$. The respective partial derivatives are $g_1 = x_2$ and $g_2 = x_1$. Consider a points $a = [5, 1000]$. $g_1(a) = 1000$ and $g_2(a) = 5$. This implies that for output $f(a) = 5000$, x_1 had a stronger influence than x_2. But in reality x_2 has a stronger contribution towards $f(a)$ than x_1. Gradients are thus viable candidates for the question "How much would perturbing a particular input affect the output?", but not for determining which input influenced a particular output neuron.

Besides, perturbations and gradients can be viewed as capturing the Individual Causal Effect (ICE) of input neuron x_i with values α on output y.

$$ICE^y_{do(x_i=\alpha)} = \mathbb{E}[y|do(x_i = \alpha), x_{j\neq i} = data] - baseline \tag{5}$$

In Equation 5, $x_{j\neq i} = data$ denotes conditioning the input neurons other than x_i to the input training instance values. The Expectation operator for y is over the unobservable noise which is equal to the learned neural function $f(.)$ itself, i.e., $ICE^y_{do(x_i=\alpha)} = f(x_1, x_2, ..., \alpha.., x_n) - baseline$, where the baseline is $f(x_1, x_2, ..., \alpha - \epsilon, .., x_n)$ for some $\epsilon \in \mathbb{R}$. Evidently, inter-feature interactions can conceal the real importance of input feature x_i in this computation, when only the ICE is analyzed.

The latter approach of "interpretable" regression is highly prone to artifacts as regression primarily maps correlations rather than causation. Regression of an output variable y (the neural network output) on a set of input features is akin to calculating $\mathbb{E}[y|x_1, x_2, ..., x_k]$, given k input features. However, true causal effects of x_i on y are discerned via $\mathbb{E}[y|do(x_i)]$, as in (Pearl, 2009). The only way regressing on a particular input feature would give $\mathbb{E}[y|do(x_i)]$ is if all the backdoor variables are controlled and a weighted average according to the distribution of these backdoor variables is taken (Pearl, 2009). Thus, causal statements made from regressing on all input variables (say, the weights of a linear approximator to a deep network) would be far from the true picture.

A.3. Proofs

A.3.1. Proof of Proposition 1

Proof. In a feedforward neural network, each layer neurons can be written as functions of neurons in its previous layer, i.e. $\forall i \in l : \forall l_{i_j} \in l_i : l_{i_j} = f_{i_j}(l_{i-1})$. The input layer l_1 can be assumed to be functions of independent noise variables U such that $l_{1_i} = f_{1_i}(u_i)\ \forall l_{1_i} \in l_1$ and $u_i \in U$. This structure in the random variables, neurons in the network, can be equivalently expressed by a SCM $M([l_1, l_2, ..., l_n], U, [f_1, f_2, ...f_n], P_u)$. □

A.3.2. Proof of Corollary 1.1

Proof. All notations are consistent with their definitions in Proposition 1. Starting with each neuron l_{n_i} in the output layer l_n, the corresponding causal function $f_{n_i}(l_{n-1})$ can be substituted as $f_{n_i}(f_{n-1_1}(l_{n-2}), f_{n-1_2}(l_{n-2}), f_{n-1_3}(l_{n-2}), ...f_{n-1_{|l_{n-1}|}}(l_{n-2}))$. This can also be written as $l_{n_i} = f'_{n_i}(l_{n-2})$. f_{i_j} refers to the causal function of neuron j in layer i. Similarly, l_{i_j} refers to neuron j in layer i. Proceeding recursively layer by layer, we obtain modified functions such that, $\forall l_{n_i} \in layer\ l_n : l_{n_i} = f'_{n_i}(l_1)$. The causal mechanisms set f' of the reduced SCM M' would be $\{f'_{n_i}|l_{n_i} \in l_n\} \cup \{l_{1_i} = f_{1_i}(u_i)\ |l_{1_i} \in l_1 \text{ and } u_i \in U\}$ □

A.3.3. Proof of Proposition 2

Proof. Let M^c be the causally sufficient SCM for a given SCM M'. Let $G^c = (V, E)$ be the corresponding causal bayesian network. Presence of dependency between input features in neural network N implies the existence of common exogenous parent vertices in the graph G^c. All the paths from one input neuron to another in graph G^c either passes through an exogenous variable or a vertex corresponding to an output neuron. The output neurons are colliders and the intervention on v_i, surgically removes all incoming edges to v_i (refer to Section A.1). As all the paths from v_i to every other input neuron v_j are "blocked", from Definition A.2, the intervened input neuron is d-seperated from all other input neurons. □

A.3.4. Proof of Proposition 3

Proof. Let p_{y^t} be a probability density over the output variables y^t at time t. Now, from Corollary 1.1 and Section 3

$$y^t = f(x^1, x^2, ..., x^{t-1}). \tag{6}$$

$f(.)$ is a recurrent function (the neural network). □

In the reduced SCM M' for the recurrent function $f(.)$, if the values of all other input neurons at different timesteps are controlled (fixed), y^t transforms according to $f(x^{t-k})$. Let's assume y^t depends on x^{t-k} via a one-to-one mapping. Note, if there exists a one-to-one mapping between x^{t-k} and y^t, then the conditional entropy $H(y^t|x^{t-k})$ would be 0, thus maximizing the mutual information between the two random variables. So, we limit the lookback to only those timesteps that register a one-to-one mapping with y^t.

The probability of y^t in an infinitesimal volume dy^t is given by,

$$P(y^t) = p(y^t)dy^t \tag{7}$$

Algorithm 1 Calculate interventional expectation for feedforward networks

Result: $\mathbb{E}(y|do(x_i))$
Input: output neuron y, intervened input neuron x_i, input value constraints $[low^i, high^i]$, number of interventions num, means μ, covariance matrix Cov, neural network function $f()$
Initialize: $Cov[x_i][:] := 0$; $Cov[:][x_i] := 0$; $interventional\_expectation := []$; $\alpha = low^i$
while $\alpha \leq high^i$ **do**
  $\mu[i] := \alpha$
  $interventional\_expectation$.append($f(\mu)$+ $\frac{1}{2}$trace(matmul($\nabla^2 f(\mu), Cov$)))
  $\alpha := \alpha + \frac{high^i - low^i}{num}$
end while

By change of variables

$$P(y^t) = p(y^t(x^{t-k}))|det(\nabla_{x^{t-k}} y^t)|dx^{t-k} \tag{8}$$

Now, dy^t and dx^{t-k} are volumes and hence are positive constants. y^t exists in the training data and hence $P(y^t) > 0$. Similarly, $p(y^t(x^{t-k})) \neq 0$. Thus, if $P(y^t)$ evaluated using Equation 8 is zero, there is a contradiction. Hence, the assumption that y^t depends on x^{t-k} via a one-to-one mapping is incorrect. $\tau_x = \max_k(|det(\nabla_{x^{t-k}} y^t)| > 0)$. would be optimal for a particular input sequence x and output y^t. $\mathbb{E}_x[\max_k(|det(\nabla_{x^{t-k}} y^t)| > 0)]$ is taken as the τ for the entire dataset, to prevent re-computation for every new input sequence.

A.4. Algorithms/Pseudocode

A.4.1. Algorithm for Phase I in Feedforward Networks

Algorithm 1 outputs an array of size num with interventional expectations of an output neuron y given different interventions ($do(.)$) on x_i. The user input parameter num decides how many evenly spaced α values are desired. The accuracy of the learned polynomial functions in Phase II depends on the size of num.

Consider n training points, and k input neurons in a feedforward network. Usually, $n \gg k$ to avoid memorization by the network. Computations are performed on-the-fly via a single pass through the computational graph in frameworks such as Tensorflow (Abadi et al., 2016) and PyTorch(Team, 2017). If one single pass over the computational graph is considered 1 unit of computation, the computational complexity of Phase I (Algorithm 1) would be O($k \times num$). Compare this to the computational complexity of O($n \times num$) for calculating the interventional expectations naively. For every perturbation α of neuron x_i, we would require atleast n forward passes on the network to estimate $\mathbb{E}(y|do(x_i = \alpha))$.

Algorithm 2 Calculate interventional expectation for recurrent networks

Result: $\mathbb{E}(y^t|do(x_i^{\hat{t}}))$
Input: output neuron y^t, intervened input neuron $x_i^{\hat{t}}$ at time $\hat{t}$, input value constraints $[low_i^{\hat{t}}, high_i^{\hat{t}}]$, number of interventions num, training input data $Data$, recurrent function $f()$
Initialize: $\alpha = low_i^{\hat{t}}$; $interventional\_expectation := []$;
while $\alpha \leq high_i^{\hat{t}}$ **do**
  $data\_iterator$:= 0
  $inputdata := Data[:, : t+1, :]$ //past is independent of the present timestep t
  $inputdata[:, t, i] := \alpha$ //setting the value of the intervened variable
  while $data\_iterator < Data.size()$ **do**
    $next\_timestep\_input := f(input\_data)$
    $inputdata$.append($next\_timestep\_input$)
    $data\_iterator$ += 1
  end while
  μ := Mean($inputdata$) //Calculate mean of each input neuron
  Cov := Covariance($inputdata$)
  $tempvar := f(\mu)$
  $hess := \nabla^2 f(\mu)$
  $interventional\_expectation$.append($tempvar$ + $\frac{1}{2}$trace(matmul($hess$,Cov)))
  $\alpha := \alpha + \frac{high^i - low^i}{num}$
end while

A.4.2. Algorithm for Phase I in Recurrent networks

See Algorithm 2. The input training data is arranged in a tensor of size $num\_samples \times num\_time \times num\_features$.

A.4.3. Algorithm for Phase II

See Algorithm 3.

A.5. Scaling to Large Data

In this section we follow the same notations as defined in Section 4 in the main text. Evaluating the interventional expectations using Eqn 4 involves calculating the Hessian. This is a costly operation. For a system with k input features it takes about $O(k)$ backward passes along the computational graph. Several domains involve a large number of input features. Such a large k regime would render Equation 4 inefficient. Note however that

Algorithm 3 Learning causal regressors

Result: $baseline, predictive\_mean, predictive\_variance$
Input: interventional expectation for different interventions $\mathbb{E}(y|do(x_i))$, input value constraints $[low^i, high^i]$
Initialize: $\alpha = low^i$;
order := Bayesian_Model_Selection($\mathbb{E}(y|do(x_i))$)
predictive_mean, predictive_variance := Bayesian_linear_regression($\mathbb{E}(y|do(x_i))$, order)
baseline := Integrate(predictive_mean, $low^i, high^i$)

we never explicitly require the Hessian, just the term $\sum_{i=1}^{k}\sum_{j=1}^{k}\nabla^2 f'_y(\mu)_{ij}Cov(x_i, x_j|do(x_l = \alpha))$. In this section, we propose an efficient methodology to compute the interventional expectations for high-dimensional data.

We begin with computing $Cov(\mathbf{x}, \mathbf{x}|do(x_l = \alpha))$, where $\mathbf{x}$ is the input vector. Consider the eigendecomposition of $Cov(\mathbf{x}, \mathbf{x}|do(x_l = \alpha)) = \sum_{r=1}^{k}\lambda_r e_r e_r^T$, where e_r is the r^{th} eigenvector and λ_r the corresponding eigenvalue. Let $v_r = \lambda^{1/2}e_r$. Performing a Taylor series expansion of f'_y around μ, we get:

$$f'_y(\mu + \epsilon v_r) = f'_y(\mu) + \epsilon\nabla^T f'_y(\mu)v_r + \frac{\epsilon^2}{2}v_r^T\nabla^2 f'_y(\mu)v_r + O(\epsilon^3 v_r^3)$$

$$f'_y(\mu - \epsilon v_r) = f'_y(\mu) - \epsilon\nabla^T f'_y(\mu)v_r + \frac{\epsilon^2}{2}v_r^T\nabla^2 f'_y(\mu)v_r + O(-\epsilon^3 v_r^3)$$

Adding the equations:

$$f'_y(\mu - \epsilon v_r) + f'_y(\mu + \epsilon v_r) - 2f'_y(\mu) = \epsilon^2 v_r^T\nabla^2 f'_y(\mu)v_r + O(\epsilon^4 v_r^4)$$

$$\frac{1}{\epsilon^2}\Big(f'_y(\mu - \epsilon v_r) + f'_y(\mu + \epsilon v_r) - 2f'_y(\mu)\Big) = v_r^T\nabla^2 f'_y(\mu)v_r + O(\epsilon^2 v_r^4)$$

Rather:

$$\lim_{\epsilon\to 0}\frac{1}{\epsilon^2}\Big(f'_y(\mu-\epsilon v_r)+f'_y(\mu+\epsilon v_r)-2f'_y(\mu)\Big) = v_r^T\nabla^2 f'_y(\mu)v_r \tag{9}$$

Equation 9 calculates the second order directional derivative along v_r. Since $Cov(x_i, x_j|do(x_l = \alpha)) = \sum_{r=1}^{k} v_{ri}v_{rj}$ (ri and rj refer to the i^{th} & j^{th} entry of v_r respectively), $\sum_{r=1}^{k} v_r^T\nabla^2 f'_y(\mu)v_r = \sum_{i=1}^{k}\sum_{j=1}^{k}\nabla^2 f'_y(\mu)_{ij}Cov(x_i, x_j|do(x_l = \alpha))$. Thus, the second order term in Eqn 4 can be calculated by three forward passes on the computational graph with inputs $\mu, \mu+\epsilon V, \mu-\epsilon V$, where V is the matrix with v_rs as columns and ϵ is taken to be very small (10^{-6}). Although eigendecomposition is also compute-intensive, the availability of efficient procedures allowed us to get results significantly faster than exact calculations (0.04s for the approximation v/s 3.04s per computation for experiments on MNIST dataset with a deep neural network of 4 hidden layers).

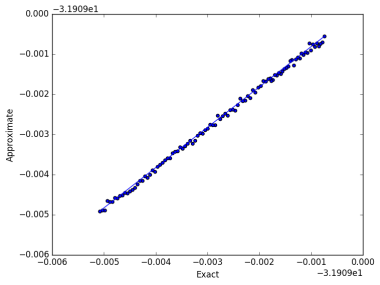

Figure 6. Quality of approximation via second order directional derivatives

Figure 6 shows results for the approximate second order term calculated v/s the exact second order term for different α values (Section A.5). The function f'_y is a neural network trained on MNIST images. Both the methods agree "almost" perfectly with each other as indicated by the $y = x$ line.

In case of feedforward networks, from Corollary 2.1, we know that $Cov(x_i, x_j|do(x_l = \alpha)) = Cov(x_i, x_j)$, i.e., the observational covariances. For recurrent networks, $Cov(x_i, x_j|do(x_l = \alpha))$ can be calculated after explicitly intervening on the system (Section 4.5).

A.6. More on Experiments and Results

A.6.1. Generation of Synthetic Dataset

We used the following procedure for generating the synthetic dataset used for experiments (Section 5.2):

- Sample individual sequences uniformly of length between $[T, T + 5]$. We used $T = 10$. Let x^t refer to the sequence value at length t.
- $\forall i; 2 < i \leq T$ Sample $x^i \sim \mathcal{N}(0, 0.2)$.
- With probability 0.5 either (a) sample $\forall i; 0 \leq i < 3$ $x^i \sim \mathcal{N}(1, 0.2)$ and label such sequences class 1 or (b) sample $\forall i; 0 \leq i < 3$ $x^i \sim \mathcal{N}(-1, 0.2)$ and label such sequences class 0.

A.6.2. Calculation of the interventional expectations in Section 5.4

From the generative model of the VAE we have access to $p(x_{ij}|z, c)$. Each pixel $p(x_{ij}|z, c)$ is modelled as a Bernoulli random variable with parameter θ_{ij}, z being the continous latents $[z_0, z_1, z_2, ..., z_9]$ and c being the class-specific binary variables $[c_0, c_1, c_2, ..., c_9]$. Interventional expectations required for calculated ACEs are calculated via Equation 2.

For continuous latents:

$$\mathbb{E}[x_{ij}|do(z_k = \alpha), do(c_l = 1)] = \mathop{\mathbb{E}}_{z\setminus z_k}[\mathop{\mathbb{E}}_{x_{ij}}[x_{ij}|do(z_k = \alpha), do(c_l = 1), z]].$$

From the generative model prior $p(z, c)$, we know that each z_k is independently distributed according to $\mathcal{N}(0, 1)$, so the intervention does not change the distribution of the other variables. However, the multinouli distribution over the c's forces all the other $c_{i \neq l} = 0$. Thus, the above expression can be simply computed via Monte Carlo integration as follows: $\frac{1}{K} \Sigma_{z \setminus z_k \sim N(0, I_9)} \theta_{ij}$, where K samples are drawn.

For discrete latents: there are two cases depending on the intervention value α.
Case 1:

$$\mathbb{E}[x_{ij}|do(c_k = 1)] = \mathop{\mathbb{E}}_{z}[\mathop{\mathbb{E}}_{x_{ij}}[x_{ij}|do(c_k = 1), z]]$$

As before, the multinoulli distribution over the cs restricts all the other $c_{i \neq k} = 0$. Thus, the above expression can be simply computed via Monte Carlo integration as follows: $\frac{1}{K} \Sigma_{z \sim N(0, I_{10})} \theta_{ij}$, where K samples are drawn.
Case 2:

$$\mathbb{E}[x_{ij}|do(c_k = 0) = \mathop{\mathbb{E}}_{z, c \setminus c_k}[\mathop{\mathbb{E}}_{x_{ij}}[x_{ij}|do(c_k = 0), c, z]]$$

Now, as $c_k = 0$, the distribution over all the other $c_{i \neq k} \sim Mult(1, U\{0, 9\} \setminus k)$. Thus, the above expression can be simply computed via Monte Carlo Integration as follows, $\frac{1}{K} \Sigma_{z \sim N(0, I_{10}) c \setminus c_k \sim Mult(1, U\{0,9\} \setminus k)} \theta_{ij}$, where K samples are drawn.

A.6.3. Additional Results: Visualizing Causal Effect

In continuation to results in Section 5.4, we present additional results here. We fix the class part of the latent and sample a random vector z from $\mathcal{N}(0, 1)$. Then we intervene on one of the dimensions of z and pass the latent through the decoder. We intervene with values in the range -3 to 3. This is repeated for every dimension. When the decoded images are sorted based on the value of intervention, we are able to see the effect of rotation in dimension z_0 and the effect of scaling in dimension z_6. Other dimensions show no effect, as shown in Figure 7.

Figure 8 shows causal attributions of the continuous latents z_k (defined in Section 5.4) for the decoded image for different class-specific latents (c_k). In all the cases z_0 and z_6 capture rotation and scaling of the digit respectively. z_2 like all the other z_ks showed no discernable causal effect. We also show causal attributions of these latents for digits 0, 2 and 3 in Figure 8 to 10.

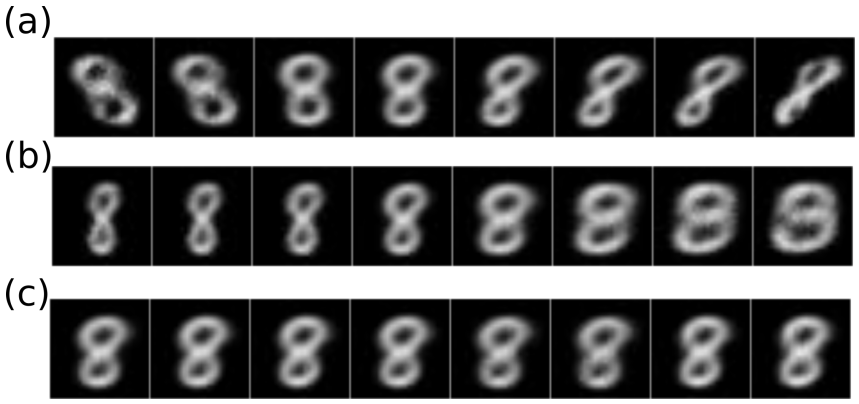

Figure 7. Decoded images generated by a random latent vector, with interventions between -3.0 to 3.0 on (a) z_0, (b) z_6, (c) z_2. The observed trends are consistent with the causal effects observed via causal attributions on the respective z_ks. z_0 captures rotation, z_6 captures scaling, and z_2 captures nothing discernable.

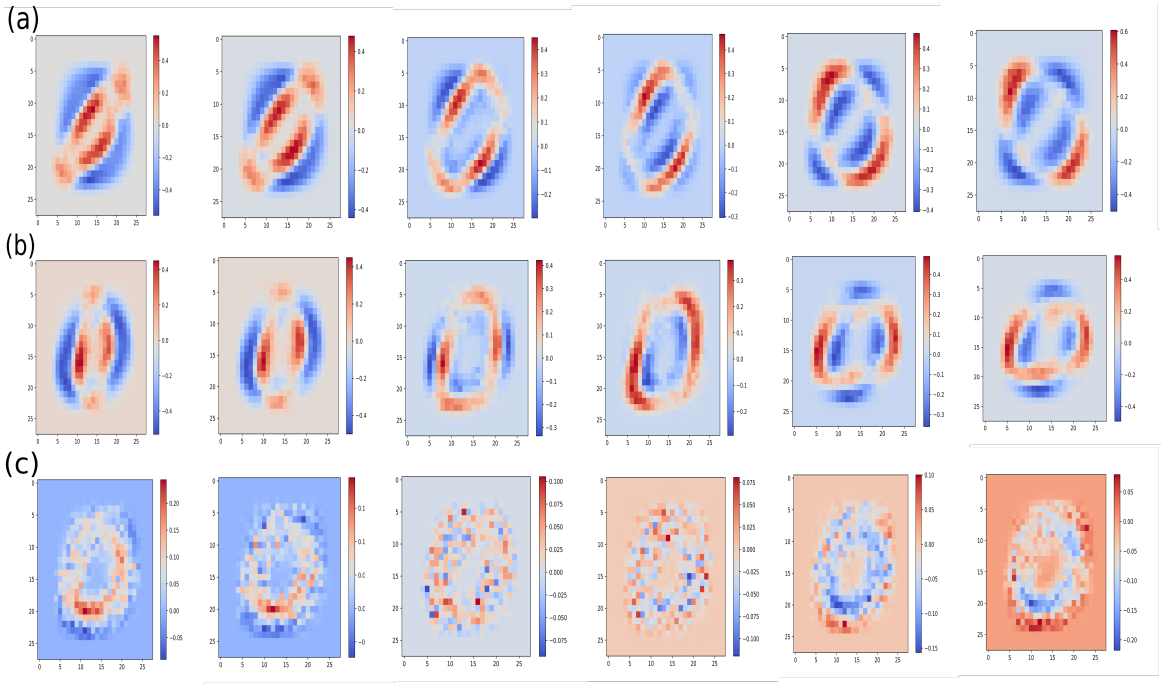

Figure 8. Causal attributions of (a) z_0 & c_0, (b) z_6 & c_0, (c) z_2 & c_0 for the decoded image. Refer Section 5.4 for details. Red indicates a stronger causal effect, and blue indicates a weaker effect. The class-specific latent intervened on here is digit 0.

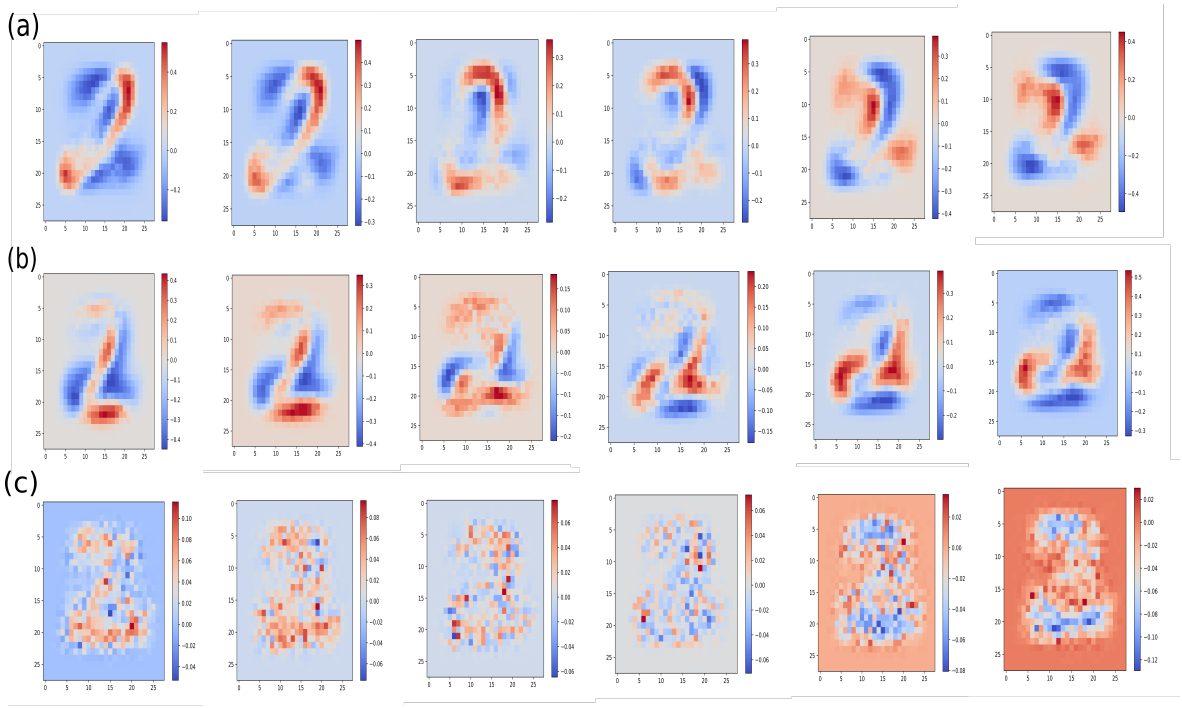

Figure 9. Causal attributions of (a) z_0 & c_3, (b) z_6 & c_3, (c) z_2 & c_3 for the decoded image. Refer Section 5.4 for details. Red indicates a stronger causal effect, and blue indicates a weaker effect. The class-specific latent intervened on here is 2.

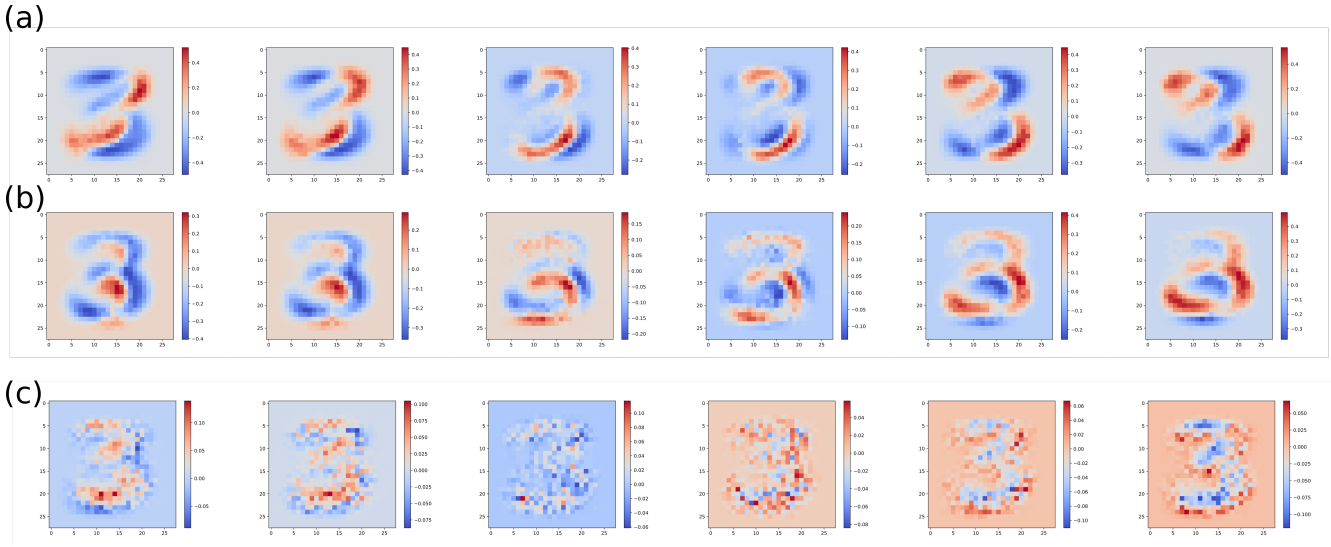

Figure 10. Causal attributions of (a) z_0 & c_3, (b) z_6 & c_3, (c) z_2 & c_3 for the decoded image. Refer Section 5.4 for details. Red indicates a stronger causal effect, and blue indicates a weaker effect. The class-specific latent intervened on here is 3.