ECINN: Efficient Counterfactuals from Invertible Neural Networks

Frederik Hvilshøj
Aarhus University, CS
fhvilshoj@cs.au.dk

Alexandros Iosifidis
Aarhus University, ENG
ai@ece.au.dk

Ira Assent
Aarhus University, CS
ira@cs.au.dk

Abstract

Counterfactual examples identify how inputs can be altered to change the predicted class of a classifier, thus opening up the black-box nature of, e.g.*, deep neural networks. We propose a method, ECINN, that utilizes the generative capacities of invertible neural networks for image classification to generate counterfactual examples efficiently. In contrast to competing methods that sometimes need a thousand evaluations or more of the classifier, ECINN has a closed-form expression and generates a counterfactual in the time of only two evaluations. Arguably, the main challenge of generating counterfactual examples is to alter only input features that affect the predicted outcome,* i.e.*, class-dependent features. Our experiments demonstrate how ECINN alters class-dependent image regions to change the perceptual and predicted class of the counterfactuals. Additionally, we extend ECINN to also produce heatmaps (ECINNh) for easy inspection of,* e.g.*, pairwise class-dependent changes in the generated counterfactual examples. Experimentally, we find that ECINNh outperforms established methods that generate heatmap-based explanations.*

1. Introduction

Deep neural networks are becoming increasingly popular and exhibit unprecedented capabilities within a range of computer vision tasks, some even surpassing human performance [41]. The price for such high performance is a lack of transparency. In high stake domains like health care, autonomous transportation, or automated decision-making involving human lives, opaque models can be an issue, *e.g.*, due to a lack of understanding of the networks.

In recent years, a great effort has been devoted to open up the black-box nature of deep neural networks for computer vision. Among others, heatmaps [3], class-maximizing samples [30], and contrastive examples [7] have been proposed. In this work, we mainly focus on the latter.

Contrastive examples are also known as counterfactual examples, even though models do not possess any causal structure as described in [24]. We adopt the setting from [11] and consider the generic question, "For situation X, why was the outcome Y and not Z?" We provide a counterfactual example to give an explanation of the form "Had X been $\hat{X}$, then the outcome would have been Z."

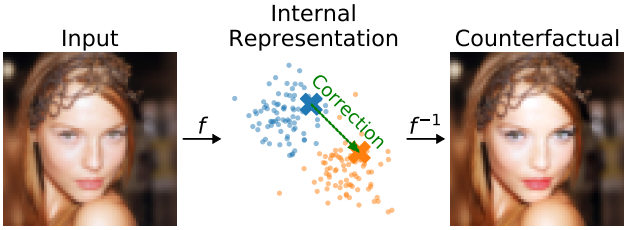


Figure 1. INN f transforms image of woman *without* makeup (left) into an internal representation. The internal representation is corrected with closed-form expression (center). Inverse INN f^{-1} generates counterfactual example *with* makeup (right).

Being able to provide counterfactual examples for complex neural networks has an immense potential to improve human-model-interactions. To name but a few, surveillance systems could be assessed for biases when picking out candidates for screening, and self-driving vehicles could be better diagnosed when misinterpreting their image feeds [11].

Good counterfactual examples are broadly agreed to be realistic, minimal, and actionable [10, 38]. In the image domain, however, minimal changes are hard to measure in a semantically meaningful way. For example, an adversarial attack changing just one pixel can be enough to change predictions [33]. While such an attack is minimal in terms of the number of pixels changed, it is not realistic and thus not desired in the context of counterfactual examples. As such, we argue that the main challenge is to generate *perceptible* and *realistically* looking images where only class-relevant features are changed. For example, lips, eyes, and maybe cheeks would change if makeup was applied to a face but not hair color or the background. Recently, many methods for generating counterfactual examples have been proposed [1, 6, 7, 10, 11, 13, 23, 36, 37, 40]. A common drawback for all the methods is that they need to query the model under consideration many times. To the best of our knowledge, we, in contrast, introduce the first algorithm that pro-

duces a counterfactual example from just one query of the model and one reverse pass.

A one-pass-solution is possible because we utilize Invertible Neural Networks (INNs) [8], which, in contrast to usual discriminative models, preserve all information between input and output layers and, in turn, allow recovering inputs exactly from their outputs. Additionally, INNs are known to have semantically organized latent spaces where translations in specific directions result in semantic changes in the input space [9]. As such, it can be argued that INNs are ideal for combining generative and discriminative capabilities for neural networks [4].

We propose the method Efficient Counterfactuals from INNs (ECINN), which utilizes already trained INN classifiers to transform inputs into an internal representation. Among internal representations, closed-form counterfactual corrections become possible. Counterfactual examples are then generated by transforming the corrected internal representations through the reverse INN. Figure 1 depicts the high-level structure of ECINN. The figure shows how an input image of a woman without makeup (left) is transformed by an INN denoted f into an internal representation (center). The internal representation is then corrected, as indicated by the green arrow, before being reverted by f^{-1} to form a counterfactual example that wears makeup (right).

We demonstrate experimentally how ECINN produces counterfactual examples that change class-dependent features while class-independent features are left largely untouched. We further demonstrate how visualizations of discrepancies between inputs and counterfactuals outperform established heatmap generation methods which produce surprisingly noisy heatmaps on conditional INNs. Backed by a simple experiment, we conjecture that such differences are due to the absence of ReLU activation in INNs.

2. Related Work

In this work, we utilize INNs to generate counterfactual examples. Therefore, we devote some attention to INNs in this section but keep the main focus on explaining neural networks with counterfactual examples.

2.1. Explaining Neural Networks

Counterfactual Examples. In recent years, many methods have been proposed for synthesizing counterfactual examples or identifying counterfactual features on various types of data. To name but a few, [1, 11, 36, 37, 39] operate on image data, [13, 14, 40] consider text, and yet other methods operate on relatively low dimensional data compared to images and text [6, 10, 38].

Methods for generating counterfactual examples can be categorized by the insights needed into the predictive model. Methods from the first category consider the predictive model as opaque and need no insight. Methods from the second category utilize gradients of the predictive model, while methods from the last category use internal data representations of the predictive model. All methods mentioned here have the drawback that they need to query the predictive model multiple times. [39] identifies counterfactual regions in input images but does not generate counterfactual examples. In contrast, after a preprocessing step that needs to be done only once, our method uses a single forward and inverse pass through the model to generate a counterfactual example.

In the first category, methods operating on opaque models typically work by iteratively generating candidate sets of counterfactual examples and then querying the predictive model to test candidates. [10] utilizes a greedy heuristic from simple data statistics to determine what input features to perturb, while [28] uses a genetic algorithm. [37] segments input images into super-pixels and use a greedy algorithm to perturb super-pixels to identify which regions affect the output of the classifier. On text data, [40] finetunes a GPT-2 model [25] to generate similar sentences to the input sentence to generate new candidates. Similarly, [36] uses autoencoders and KD-trees to identify images similar to input images to speed up the search for candidates that change the prediction of the predictive network. In comparison to a forward and inverse pass which ECINN uses, the default maximum queries of the classifier in the official code of [36] is a thousand.

The second category of methods employs gradient optimization techniques to identify inputs that change the decision of the predictive model. We note that such ideas are not new. Previous work, albeit from a different perspective, has developed methods for synthesizing inputs that maximize desired (output) neurons of a given network. For example, [30] uses gradient descent with an L_2-norm prior loss on a random input to maximize output neurons. [21] includes a local pixel variation prior in the loss to obtain more realistically looking features in the generated images. Even though the methods give insights into the inner workings of the classifier, they suffer from generating unrealistic images. More recently, [23] proposed to train a generative model to, given the input image, make new alternative images that would change the prediction of the classifier. In a similar vein, [7] utilizes a pretrained and fixed autoencoder to identify a latent code that generates the desired output through gradient optimization.

The third category of methods contains two different strategies. First, [11] considers convolutional neural networks as a composition of a (convolutional) feature extractor and a classification network and proposes two algorithms to mix fibers of the feature extractor applied to the input and a sample from the counterfactual class. Second, [1] similarly uses a part of the classifying network as a feature extractor to cluster such features. The result is an identifica-

tion of semantic features like stripes, wool, etc. A gradient descent algorithm then learns how to add or remove from an input to obtain a counterfactual example.

The work we present in this paper fits best into the third category. However, our approach is conceptually different. Instead of generating counterfactual examples from an "arbitrary" neural network, we choose a specific family of neural networks, INNs, to generate counterfactual examples efficiently without the use of multiple queries of the model or gradient computations.

Heatmaps. There exist many methods that produce explanations in the form of heatmaps. Some methods work on black-box predictive models [5, 26, 32, 43] while others utilize gradient-like computations on the predictive model [3, 19, 29, 31, 34, 42]. In this work, we compare our method against four methods from the latter category. i) DeepLift [29] which is based on discrepancies between modified gradients of the input and a non-informative reference point, ii) Integrated Gradients [34] (IntGrad) which approximates integrals of gradients from a reference point to the input, iii) GradSHAP, and iv) DeepLiftSHAP which are two related methods for approximating SHAP values [19]. We refer the reader to [19] for a detailed description of the four methods.

2.2. INNs as Generative Classifiers

INNs have gained wide attention as unsupervised generative models which allow generating realistically looking "fake" samples [8, 9, 17]; when used for generative modeling, INNs are typically referred to as Normalizing Flows. Despite hidden in appendices, both [9] and [17] present samples generated from class-conditional INNs. Later, it was explicitly described how to follow the INNs by a Gaussian mixture model (GMM) to obtain a generative classifier [12, 22], which both allows class-conditional sampling and sample classification. However, adding classification abilities comes at a price. As demonstrated in [2], there is a trade-off between classification performance and the quality of the generated fake images. The work introduces an information bottleneck loss, which explicitly trades off the classification and generation performance through a hyperparameter β. [2] further introduces a new invertible model architecture, which we refer to as IB-INN.

Regarding interpretability, [20] shows how conditional INNs can be trustworthy classifiers by visualizing decision spaces, comparing class similarities, and computing posterior heatmaps. In this work, we further show conditional INNs to be trustworthy classifiers by using them for generating counterfactual examples.

3. Efficient Counterfactual Examples

This section constitutes our main contribution. We combine theoretical insights and practical observations from INNs to generate counterfactual examples efficiently.

3.1. Problem Statement

As mentioned, counterfactual examples are samples that indicate why an input instance was predicted to be one class rather than another. Specifically, we modify the definition from [38] which states that counterfactual examples are statements taking the form: "Score p was returned because variables V had values $(v_1, v_2, \dots)$ associated with them. If V instead had values $(v'_1, v'_2, \dots)$, and all other variables had remained constant, score p' would have been returned." In the context of image classification, we define counterfactual examples as visualizations showing how the input image can be altered to change the predicted class.

Desiderata. In line with the desiderata of [10] and [38], we find that three properties are of high importance for counterfactuals to be useful. i) *Only semantically relevant features should be changed*. For example, facial features like lips, cheeks, and eyes might change while background and hair should not when a counterfactual is generated for a face without makeup. ii) *Counterfactuals should look realistic*. Examples of unrealistic counterfactuals could be misplaced eyes on a face, extreme color values, or a "one-pixel-change" like the adversarial examples presented in [33]. iii) *Both tipping-point counterfactuals and convincing counterfactuals should be prioritized*. We refer to counterfactuals on the decision boundary between the input and the target class as tipping-point counterfactuals. Likewise, counterfactuals, where the target class is predicted with high confidence, are referred to as convincing counterfactuals. Tipping-point counterfactuals are essential because they identify a minimal correction to the input. However, they might not always make sense due to visual class differences. For example, when changing the predicted class of a cat to a dog, a tipping-point counterfactual might fail to show how the ears should be pointy instead of hanging because the tipping-point would represent something in between. On the contrary, a convincing counterfactual would successfully show such transformation, but potentially with too pronounced changes. Providing both types of explanations thus give a deeper insight into the decisions of the classifier.

We emphasize that the counterfactual examples discussed in this work are not causal as counterfactual examples described in, *e.g.*, [24]. Although the ambiguity of the name is unfortunate, we stick to the naming convention to be consistent with related work.

3.2. Conditional INNs

We find INNs to be well suited for the counterfactual problem because they are bijective, *i.e.*, every latent vector corresponds to exactly one input. In contrast, typical classification models are inherently surjective, *i.e.*, there exist many inputs which produce each output. Identifying the best input from an output thus becomes simpler for INNs.

It is also known that well-trained INNs have semantically organized latent spaces [9]. We believe that when many latent representations of samples from the same class are averaged, then class-independent information like background and object orientation will cancel out and leave just class-dependent information. ECINN isolates such latent class-dependent information and uses it to correct latent space embeddings to generate counterfactual examples.

A conditional INN f is typically trained by computing latent vectors $z = f(X)$ from input vectors X and using the latent vectors to fit a GMM to class labels Y. However, to use Z rather than X in the GMM, one must use the change-of-variables formula, which states that

$$\log p_X(x|y) = \log p_Z(f(x)|y) + \log |det\,(J)|\,. \qquad (1)$$

That is, the class-conditional log density of an input x in the image space, $p_X(x|y)$, is equal to the class-conditional log density of $f(x)$ in the latent space $p_Z(f(x)|y)$, but with an additional Jacobian term, $J = \frac{\partial f(x)}{\partial x}$. Typically, the class-dependent latent densities are chosen to be Gaussians, $p_Z(z|y) = \mathcal{N}(\mu_y, \mathbb{1})$. By Bayes' rule, we notice that under a uniform prior distribution over labels, $p(y) = 1/K$ for K classes, the log posterior probability becomes

$$\log p_X(y|x) = \log \frac{p_X(x|y)}{\sum_{y'} p_X(x|y')} \propto -||f(x) - \mu_y||^2. \qquad (2)$$

From Equation (2), we see that independent of the Jacobian determinant, latent vector $z = f(x)$ will be predicted to be from the class y with the closest model mean, μ_y. In turn, the latent space of the classifier can be analyzed under L_2-norms instead of less efficient and complex densities $p_X(x|y)$, which depend on the Jacobian determinant. In the following subsection, we present how ECINN utilizes this insight to produce counterfactual examples efficiently.

3.3. ECINN

At a high level, ECINN transforms images into a latent space through an INN f. In the latent space, a closed-form expression is used to correct the latent embedding to change the predicted class of the INN. From the corrected embedding, a counterfactual is generated by the inverse INN f^{-1}.

As a preprocessing step that needs to be done only once, we group the training samples by their classified output, $G_j = \{x|C(x) = j\}$, where $C(x) = \arg\max_y p_X(y|x)$ is the predicted class. Afterwards, we compute mean latent vectors $\bar{\mu}_j = \frac{1}{|G_j|} \sum_{x \in G_j} f(x)$ for each class j and define the vector from $\hat{\mu}_p$ to $\hat{\mu}_q$ as $\Delta_{p,q} = \bar{\mu}_q - \bar{\mu}_p$.

Given a target class q and an input x, a counterfactual example $\hat{x}^{(q)}$ is produced from the predicted class $C(x) = p$ by adding a scaled version of $\Delta_{p,q}$ to the latent space embedding $z = f(x)$ and inverting it through the INN,

$$\hat{x}^{(q)} = f^{-1}(f(x) + \alpha \Delta_{p,q}). \qquad (3)$$

As indicated by Equation (3), generating a single counterfactual example requires just one evaluation of f and f^{-1}.

To follow our third desideratum and provide both tipping-point and convincing counterfactuals, we compute two counterfactuals for each input with different values of α. First, we choose α_0 to produce a tipping-point counterfactual, which potentially reveals minimal semantic changes in the image space to change the predicted class. In the latent space, α_0 will be the value that moves the latent vector exactly onto the decision boundary between the input and target class. Due to Equation (2), α_0 is identified analytically such that $||z + \alpha_0 \Delta_{p,q} - \mu_p|| = ||z + \alpha_0 \Delta_{p,q} - \mu_q||$. The closed-form expression for α_0 is given in the supplementary material along with a proof. Second, we choose α_1 such that the target class q is predicted with high confidence to produce a convincing counterfactual. α_1 is chosen heuristically to be $\alpha_1 = \frac{4}{5} + \frac{\alpha_0}{2}$. Although it is not guaranteed that the counterfactual example generated is predicted to be from the target class, *i.e.*, $C(\hat{x}^{(q)}) = q$, we observed that the relation holds in practice.

In Figure 2, we illustrate the intuition of our method. The figure shows two unit variance normal distributions in the latent space. The blue line indicates the decision boundary between the two normal distributions, and the orange line is the line that passes through z in direction $\Delta_{p,q}$. With green squares, we indicate the two computed means $\bar{\mu}_p$ and $\bar{\mu}_q$, that are used to define $\Delta_{p,q}$ (green arrow). The two points of interest are the blue square on the intersection of the blue and the orange line and the black square to the right. According to the model, the blue square is equally likely to stem from either of the two classes, and the black square is very likely to stem from class q. In the experimental section,

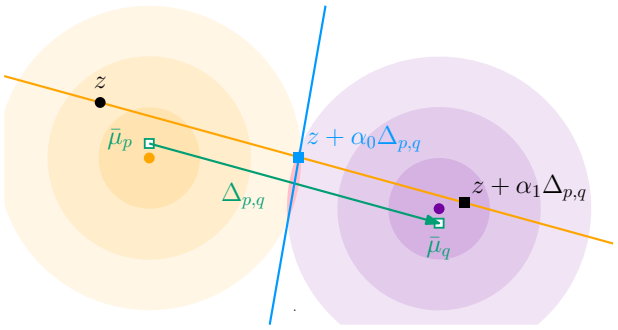


Figure 2. Latent space corrections by ECINN.

we visualize both points as counterfactual examples in the image space by inverting them through f^{-1}.

With ECINN, we have connected the well-suited properties of INNs to the observation that the latent space of the conditional INNs is easy to analyze due to the relation shown in Equation (2).

ECINNh. A common assumption for heatmap generation methods is that removing information from pixels identified as important should cause the predicted class to become less likely under the model [27]. Following this, we introduce ECINNh for producing pairwise heatmaps by highlighting pixel discrepancies between inputs, x and the generated counterfactual examples, $\hat{x}^{(q)}$.

$$h(x, q) = \hat{x}^{(q)} - x. \tag{4}$$

Equation (4) yields an estimate of how important each pixel of the input is for expressing class p of the compared to a target class q. A large absolute value means that the associated pixel needs to change a lot for the predicted class to change, *i.e.*, it intuitively contains a high amount of class-dependent information. In turn, such pixel can be interpreted as having a large impact on the predicted class. On the contrary, values close to zero indicate that associated pixels can remain unchanged while the predicted class change. Such pixels have no class-dependent information to remove and will probably not change the predicted class if altered.

The two heatmap generation methods DeConvNet [42] and GuidedBackProp [31] are examples of how ReLU-activations have a large effect on gradient-based heatmaps. Both methods demonstrate how applying ReLUs to gradient computations removes noise in the generated heatmaps. Because INNs need to be invertible, components like ReLU activations are not directly applicable. Therefore, we expect established heatmap generation methods introduced for networks with ReLU activations to be underperforming on conditional INNs. In contrast, as heatmaps generated by ECINNh were explicitly designed for INNs and are not based on gradients, we expect them to be less noisy and of a higher quality.

In conclusion, we introduce ECINN which allows computing counterfactuals efficiently by utilizing properties of INNs. ECINN complies with our first two desiderata by using INNs to generate counterfactuals from latent space directions, which represent class-dependents changes while leaving out most class-independent information. Furthermore, by providing both tipping-point and convincing counterfactuals, we follow the third desideratum. Finally, we present the ECINNh extension, which generates heatmaps that allow easy inspection of class-dependent changes in the generated explanations.

4. Experiments

In this section, we evaluate how our counterfactual examples perform. Our experiments show how ECINN produces meaningful counterfactual examples across three different image datasets, changes class-dependent features while maintaining class-independent features, and outperforms established heatmapping methods.

Experimental Details. We evaluate ECINN on a synthetic FakeMNIST dataset, on the MNIST dataset [18], and the CelebA-HQ dataset [15]. On all three datasets, classification errors of the IB-INN models are comparable to those of a standard classification network (see Table 1 in the supplementary material). For all our experiments, we have trained IB-INN models "as-is."[1] We note that the β-value of the IB-INN loss influences the performance of our method. In the presented experiments, we found that values close to one strike a good balance between classification accuracy and generative performance. In the supplementary material, we provide an overview of all models used, their hyperparameters, and their performances. We also include additional samples of all plots. Results presented in this section are all with samples from the test set and were found to be consistent across samples.

We provide code in an iPython Notebook, `code.ipynb`, which can be uploaded to Google Colab and run with one run command. Upon submission, we plan to release our code. Finally, we suggest reading this section on a screen to enable zooming on the figures.

4.1. FakeMNIST

The goal of the first experiment is to verify ECINN in a controlled setting. We construct an image dataset where less than two percent of the pixels are class-*dependent*. The remaining pixels are *independent* of the class label. As argued, a proper counterfactual example for a well-trained model should alter only the class-dependent pixels. Additionally, if the class-dependent pixels are not present, such an instance should be equally likely to be from any class.

[1] We adopted models and training code from https://github.com/VLL-HD/IB-INN.

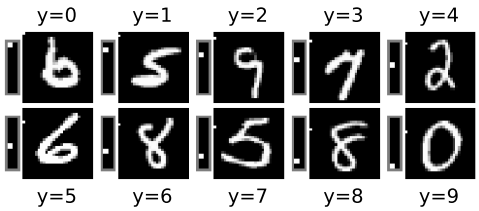


Figure 3. Random samples from the FakeMNIST dataset. For improved readability, smaller rectangles to the left of images magnify the top left 10×2 pixels, indicating the class.

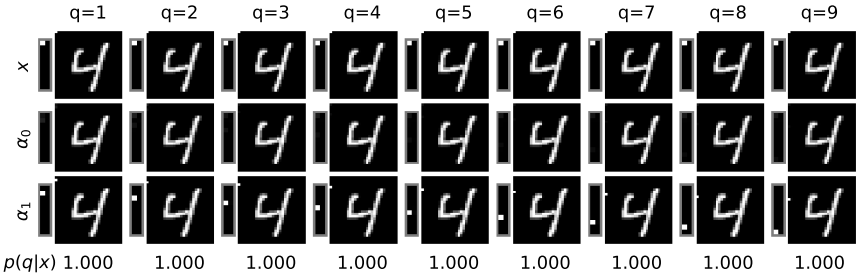


Figure 4. Counterfactual examples generated for the FakeMNIST dataset. Columns represent targets q and rows are input, α_0 counterfactuals, and α_1 counterfactuals, respectively.

We generate a dataset by computing new uniformly random labels for all MNIST samples and color the top left 10×1 pixels accordingly. For example, if an image gets assigned label "5," we color the sixth pixel in the left column white. As such, the labels are independent of the depicted digits and only depend on the top left pixels. Figure 3 shows a sample from each of the ten classes. The top-left pixels vary with the labels (y), and the depicted digits do not.

In Figure 4, we have drawn a random sample from the class $y = 0$ (first row) and display a counterfactual example for α_0 (second row), which is equally likely to stem from the class $y = 0$ and the target class q. Additionally, the figure includes a counterfactual example for α_1, which represents a high confidence ($p(q|x)$ close to 1) of the classifier (third row). Each column corresponds to a different target class q as indicated by the labels above each column.

Figure 4 shows that the dot in the top left corner of the input does change position while the class-independent digit remains unchanged as expected. Specifically, the third row from left to right reveals how the dot in the top left corner travels downwards to end in the tenth pixel. Notably, the second row has almost no dot, which aligns well with the interpretation about equally likely class probabilities above.

4.2. MNIST

Next, we apply ECINN to the MNIST dataset. We seek to investigate two properties. First, we verify our second desideratum, *i.e.*, that ECINN produces realistic counterfactual examples. Second, we investigate how well class-independent features like font-weight, tilt, and size are maintained by ECINN, *i.e.*, our first desideratum.

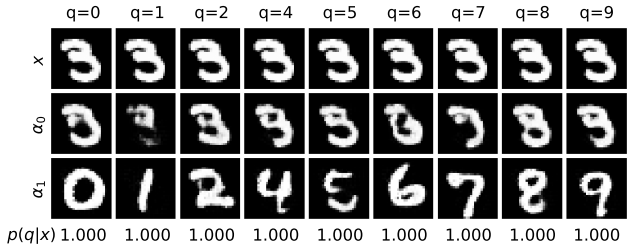


Figure 5. Counterfactual examples for a sample of a three with different target classes, q.

Realistic Counterfactuals. In Figure 5, we depict counterfactual examples in the same fasion as Figure 4. The figure shows how an image of a three is properly transformed into any of the remaining nine classes. Note that in the second row, the counterfactual examples are in many cases such that even a human might mistake the image for both the input and target class. By contrast, the third row contains samples where the three has successfully transformed into the target class. This experiment demonstrates that ECINN complies with our second desideratum by generating realistic counterfactuals.

Class-Independent Properties. In Figure 6 and 7, we demonstrate how class-independent properties like font-weight, tilt, and size are preserved during counterfactual generation. First, Figure 6 includes nine different inputs (first row), each from a different class, that are all translated

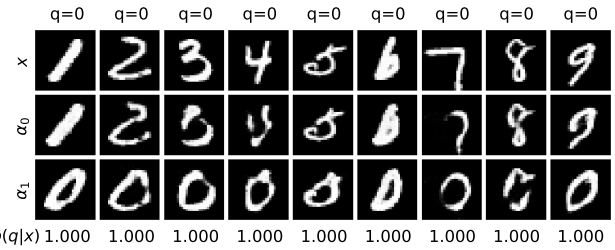


Figure 6. Counterfactual examples with $q = 0$.

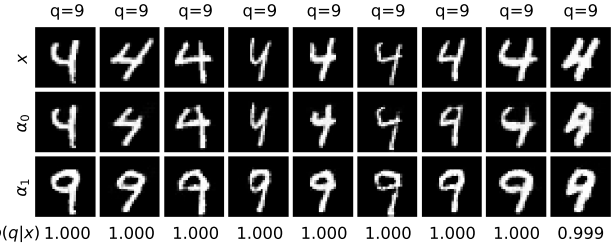


Figure 7. Diverse counterfactuals for same input and target class.

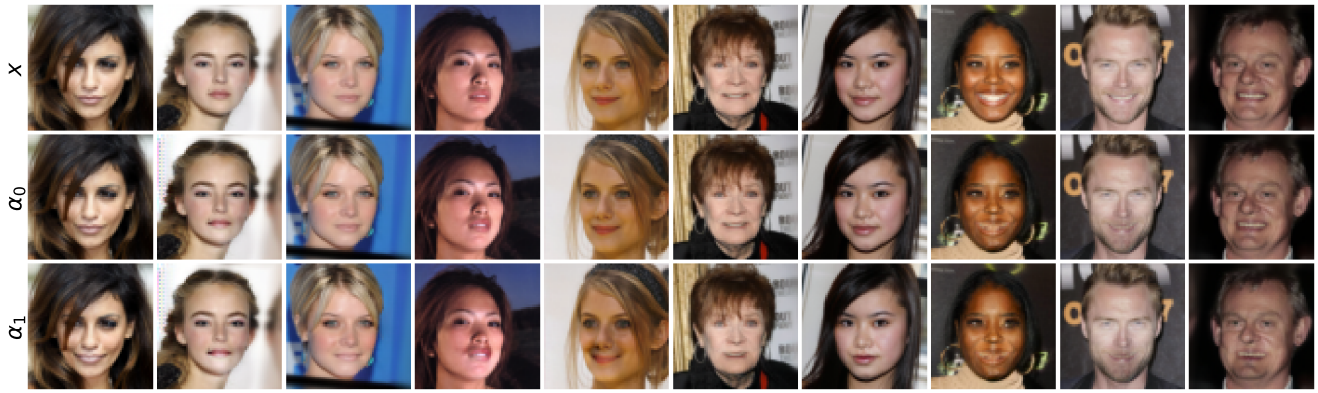


Figure 8. Counterfactual examples for frowning and smiling faces. First row is the input. Second and third row are generated with α_0 and α_1, respectively. First five columns have target q = smile and last five columns q = frown. $p(q|x) > 1 - 10^{-4}$ for all samples.

to the target class, $q = 0$. We observe that the nine outcomes (row three) are perceptually different while resembling the target class. Each counterfactual example maintains class-independent properties from the input while resembling the target class. For example, the narrow and tilted one (first column) becomes a narrow and tilted zero. Similarly, explaining the pointy five yields a pointy zero. The observations suggest that ECINN maintains properties that are not directly dependent on the label.

In Figure 7, we further investigate how class-independent properties of the input images are maintained. We sample nine different images from the class $y = 4$ and compute their counterfactual examples for the target class $q = 9$. We observe how bold inputs yield bold counterfactuals; likewise, slim inputs yield slim counterfactuals. Similar observations can be made for, *e.g.*, tilt, size, and shapes.

In conclusion, we observe that for the MNIST dataset, ECINN produces counterfactual examples which comply with our desiderata by realistically changing both the predicted and the perceived class while maintaining class-independent features such as font-weight, tilt, and size.

4.3. CelebA-HQ.

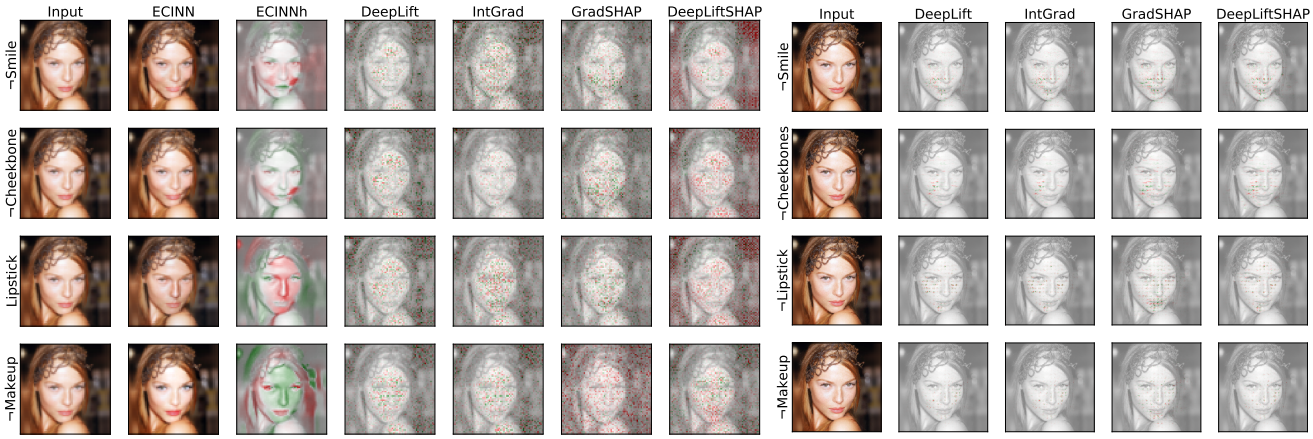
To evaluate ECINN on a more diverse and complex dataset, we extend our experiments to the CelebA-HQ dataset. We train IB-INNs to predict various labels, where each label occurs in at least 45% of the dataset.

Counterfactual Examples. In Figure 8, we show counterfactual examples as for MNIST, but on the smile versus frown label; similar plots for other labels can be found in the supplementary material. The first five columns depict how ECINN turns frowning people into smiling ones, while the last five columns make smiling people frown. First, we observe that class irrelevant features such as hair, skin color, and backgrounds remain perceptually unchanged as desired. Second, we notice that some of the counterfactual examples in the last row look unrealistic. In particular, it seems to be hard for the method to open and close mouths. In some cases, we also observe small artifacts like the ones in the left-most pixels of the second column. Based on our MNIST experiments, which did not suffer from computational limitations, we believe that scaling from roughly 40 million parameters that our models use to around 200 million parameters (as is common with previous work [17]) can remove the artifacts and generate higher quality counterfactual examples. Furthermore, the low-resolution version of CelebA-HQ that we use due to limited resources is arguably harder to synthesize than higher resolutions.

ECINNh. From the above experiments, we observe that ECINN can identify the image locations connected to the class of interest. Identifying such locations has been the main focus of various heatmap generation methods focusing on the explainability of deep learning models. To demonstrate this capability, we compare ECINNh to Integrated Gradients [34], DeepLift [29], DeepLiftSHAP [19], and GradSHAP [19].[2] We train four models to predict whether persons are smiling, have high cheekbones, wear lipstick, or wear heavy makeup. Since the labels are binary, we can directly compare the methods because the target class q is defined by the predicted class p, *i.e.*, $q = 1 - p$.

Figure 9a depicts heatmaps made by ECINNh (third column) and by the four mentioned methods (last four columns). The figure further includes the input image and the counterfactual examples generated by ECINN (first two columns, respectively). The four different rows resemble the four IB-INNs trained on the different labels. The text to the left indicates what the classifier has predicted the input to be (the symbol ¬ means "not"). Comparing the heatmaps across models and methods, we observe that

[2] All methods implemented through the PyTorch Captum framework, https://github.com/pytorch/captum.



(a) Predictive models: IB-INN [2]

(b) Predictive models: InceptionV3 [35]

Figure 9. Comparison of heatmaps on INNs. Rows represent identical model architectures trained on different labels. Columns represent methods. Text on the left indicates predicted label.

ECINNh consistently produces more coherent and meaningful heatmaps. For example, in the first row, the heatmap shows how ECINN puts a large emphasis on the lips, cheekbones, and eyes, which are all associated with smiling. In contrast, the four other heatmaps are more scattered over the whole input and only slightly denser in the facial regions.

An additional property of ECINNh is that it gives a rich insight into the behavior of the model. Inspecting the third row in Figure 9a reveals that when predicting the lipstick label, the model puts emphasis on the whole face and not just the lips to classify the sample. As such, ECINNh reveals how the model has effectively learned to detect makeup as a whole rather than lipstick. Further comparing our heatmap of the lipstick model to that of the makeup model confirms this insight. Up to a sign, the two heatmaps are almost identical. In turn, the two models emphasize very similar features to predict lipstick and makeup, respectively. A similar observation can be made between smiles and high cheekbones. As the lipstick and makeup labels are almost certain to correlate, the behavior we observe is expected. The important thing to notice is that our method, in contrast to the others, identifies this behavior. For INNs, we conclude that heatmaps computed with ECINNh are of higher quality than competing methods.

Model Architectures. Even though it is well known that Integrated Gradients can be noisy, we find the results in Figure 9a to be unusually noisy. We hypothesize that the reason is the absence of ReLU activations of INN layers.[3] As described above, there are no ReLU activations in INNs to filter out noise during backpropagation. This might explain why especially Integrated Gradients work well on, *e.g.*, the Inception-V3 network [35], but not on INNs. To investigate our hypothesis, we train Inception-V3 networks to classify the same samples as the IB-INNs, but in a 224×224 resolution due to the architecture of Inception-V3. The resulting heatmaps are depicted in Figure 9b. Comparing the four methods to Figure 9a, they are less noisy and even to some extent coincide with the areas of the heatmaps of ECINNh. We here compare heatmaps across different datasets and architectures and cannot conclude the definitive cause. However, this observation supports our hypothesis about noisy explanations under the absence of ReLU activations.

In summary, our experiments demonstrate that ECINN changes class-dependent features such as the shape of digits or the expression of a smile while leaving leaves class-independent features like tilt, font-weight, and background largely untouched. The experiments further highlight how heatmaps generated by ECINNh gives more faithful and coherent explanations than the methods we compare against.

5. Conclusion

We introduce ECINN as an efficient method for computing counterfactual examples, requiring only a forward and an inverse pass. ECINN transforms input images into a latent space where counterfactual corrections of latent vectors have a closed-form expression. Through an inverse INN, counterfactual examples are generated from the corrected latent vectors. In compliance with our desiderata, ECINN generates counterfactual explanations that i) change only class-dependent features, ii) are realistic, and iii) are diverse. These properties of ECINN are further employed in the proposed ECINNh to produce high-quality heatmaps for conditional INNs, while heatmaps of established methods are noisy.

[3] INNs do have ReLUs but only in auxiliary networks of coupling layers [9], where they do not help filter noise.

References

[1] Arjun Akula, Shuai Wang, and Song-Chun Zhu. CoCoX: Generating Conceptual and Counterfactual Explanations via Fault-Lines. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. 1, 2

[2] Lynton Ardizzone, Radek Mackowiak, Carsten Rother, and Ullrich Köthe. Training normalizing flows with the information bottleneck for competitive generative classification. *NeurIPS*, 2020. 3, 8, 11

[3] 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*, 2015. 1, 3

[4] Jens Behrmann, Will Grathwohl, Ricky T.Q. Chen, David Duvenaud, and Jörn Henrik Jacobsen. Invertible residual networks. In *ICML*, 2019. 2

[5] Chun Hao Chang, Elliot Creager, Anna Goldenberg, and David Duvenaud. Explaining image classifiers by counterfactual generation. In *ICLR*, 2019. 3

[6] Furui Cheng, Yao Ming, and Huamin Qu. DECE: decision explorer with counterfactual explanations for machine learning models. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1438–1447, 2021. 1, 2

[7] Amit Dhurandhar, Pin Yu Chen, Ronny Luss, Chun Chen Tu, Paishun Ting, Karthikeyan Shanmugam, and Payel Das. Explanations based on the Missing: Towards Contrastive Explanations with Pertinent Negatives. In *NeurIPS*, 2018. 1, 2

[8] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-Linear Independent Components Estimation. In *ICLR (Workshop)*, 2015. 2, 3

[9] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *ICLR*, 2019. 2, 3, 4, 8

[10] Oscar Gomez, Steffen Holter, Jun Yuan, and Enrico Bertini. ViCE: Visual Counterfactual Explanations for Machine Learning Models. *International Conference on Intelligent User Interfaces, Proceedings IUI*, pages 531–535, 2020. 1, 2, 3

[11] Yash Goyal, Ziyan Wu, Jan Ernst, Dhruv Batra, Devi Parikh, and Stefan Lee. Counterfactual Visual Explanations. In *ICML*, 2019. 1, 2

[12] Pavel Izmailov, Polina Kirichenko, Marc Finzi, and Andrew Gordon Wilson. Semi-supervised learning with normalizing flows. In *ICML*, 2020. 3

[13] Alon Jacovi, Swabha Swayamdipta, Shauli Ravfogel, Yanai Elazar, Yejin Choi, and Yoav Goldberg. Contrastive Explanations for Model Interpretability. *arXiv preprint arXiv:2103.01378*, 2021. 1, 2

[14] Sin-Han Kang, Honggyu Jung, Dong-Ok Won, and Seong-Whan Lee. Counterfactual explanation based on gradual construction for deep networks. *arXiv preprint arXiv:2008.01897*, 2020. 2

[15] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *ICLR*, 2018. 5

[16] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014. 11

[17] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. In *NeurIPS*, 2018. 3, 7

[18] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. 5

[19] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NIPS*, 2017. 3, 7

[20] Radek Mackowiak, Lynton Ardizzone, Ullrich Köthe, and Carsten Rother. Generative classifiers as a basis for trustworthy computer vision. *arXiv preprint arXiv:2007.15036*, 2020. 3

[21] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*, 2015. 2

[22] Tan M. Nguyen, Animesh Garg, Richard G. Baraniuk, and Anima Anandkumar. InfoCNF: Efficient conditional continuous normalizing flow using adaptive solvers. *arXiv preprint arXiv:1912.03978*, 2019. 3

[23] Jingjing Pan, Yash Goyal, and Stefan Lee. Question-conditioned counterfactual image generation for VQA. *arXiv preprint arXiv:1911.06352*, 2019. 1, 2

[24] Judea Pearl. Causes of effects and effects of causes. *Sociological Methods & Research*, 44(1):149–164, 2015. 1, 3

[25] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 2

[26] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should i trust you?" Explaining the predictions of any classifier. In *KDD*, 2016. 3

[27] 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*, 2017. 5

[28] Shubham Sharma, Jette Henderson, and Joydeep Ghosh. CERTIFAI: Counterfactual Explanations for Robustness, Transparency, Interpretability, and Fairness of Artificial Intelligence models. *CoRR*, 2019. 2

[29] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *ICML*, 2017. 3, 7

[30] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR*, 2014. 1, 2

[31] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR*, 2015. 3, 5

[32] Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, 41(3):647–665, 2014. 3

[33] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019. 1, 3

[34] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *ICML*, 2017. 3, 7

[35] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 8, 11

[36] Arnaud Van Looveren and Janis Klaise. Interpretable counterfactual explanations guided by prototypes. *arXiv preprint arXiv:1907.02584*, 2019. 1, 2

[37] Tom Vermeire and David Martens. Explainable image classification with evidence counterfactual. *arXiv preprint arXiv:2004.07511*, 2020. 1, 2

[38] Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv. JL & Tech.*, 31:841, 2017. 1, 2, 3

[39] Pei Wang and Nuno Vasconcelos. SCOUT: Self-aware Discriminant Counterfactual Explanations. In *CVPR*, 2020. 2

[40] Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel S. Weld. Polyjuice: Automated, General-purpose Counterfactual Generation. *arXiv preprint arXiv:2101.00288*, 2021. 1, 2

[41] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V. Le. Self-training with noisy student improves imagenet classification. In *CVPR*, 2020. 1

[42] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014. 3, 5

[43] Luisa M. Zintgraf, Taco S. Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. In *ICLR*, 2017. 3

A. Analytical α-value, α_0

Define $y(\alpha) = z + \alpha\Delta_{p,q}$ to be the line intersecting z with direction $\Delta_{p,q}$. We wish to identify the intersection between $y(\alpha)$ and the hyperplane that constitutes the decision boundary between the two normal distributions $\mathcal{N}(\mu_p, \mathbb{1})$ and $\mathcal{N}(\mu_q, \mathbb{1})$. Due to the simplicity of the covariance matrices of the normal distributions, we can define $w = \mu_q - \mu_p$ and $b = -\left(\frac{\mu_p+\mu_q}{2}\right)^\intercal w$ to form the decision boundary

$$w^\intercal x + b = 0. \tag{5}$$

Equation (5) corresponds to the blue line in Figure 2.

To find the α-value which corresponds to the intersection, set $x = z + \alpha\Delta_{p,q}$ and solve for α in Equation (5):

$$w^\intercal(z + \alpha\Delta_{p,q}) + b = 0 \tag{6}$$

$$\Rightarrow \quad \alpha w^\intercal \Delta_{p,q} = -(w^\intercal z + b) \tag{7}$$

$$\Rightarrow \quad \alpha = -\frac{w^\intercal z + b}{w^\intercal \Delta_{p,q}}. \tag{8}$$

B. Experimental Details

In Table 1, we provide an overview of hyperparameters and performances of the networks used in this work.

IB-INN. We have trained IB-INN models "as-is"[4] and adjusted only the β-value of the loss function. On FakeMNIST and MNIST, the IB-INN models were trained for 60 epochs with stochastic gradient descent and a milestone scheduler stepping from learning rate 0.07 to 0.007 after 50 epochs. On CelebA-HQ, the IB-INN models were trained for 800 epochs with the Adam optimizer [16] and a milestone scheduler stepping with a factor $\frac{1}{10}$ after every 200 epochs.

Inception-V3. In the last subsection of the main paper, we compare the heatmaps of conditional INNs to heatmaps of the Inception-V3 [35].

We used the inception network "as-is"[5] with the exception that we turned off the auxiliary classifier and changed the output layer to have only two output neurons. We trained the models with default parameters of the Adam optimizer and learning rate 0.001 for 9 epochs. After 9 epochs, the models started overfitting. We did not optimize the learning rate or other hyperparameters.

C. IB-INN Model and Loss

The model architecture and loss function used in this work were proposed by [2]. The loss was derived from an information bottleneck formulation with a hyperparameter, β, that allows trading off generative and classification capabilities. The loss function is based on mutual information I:

$$\mathcal{L}_{IB} = I(X, Z) - \beta I(Z, Y). \tag{9}$$

Mutual information quantifies the amount of information which is shared between variables.[6] As such, by minimizing $\mathcal{L}_{IB}$, the mutual information between the input and the latent vector is minimized while the mutual information between the latent vector and class label is maximized. In practice, the first term, $I(X, Z)$, can be thought of as a generative loss, which results in a good performance on generating images. The second term, $I(Z, Y)$, is closely related to the categorical cross-entropy loss, thus promoting high accuracy. Throughout our experiments, we use models trained with the IB-INN loss, $\mathcal{L}_{IB}$.

For simplicity, we do not conduct experiments across multiple values of β. Overall, we find that values close to one strike a good balance between counterfactual examples and model accuracy in our experiments. We do, however, include Figure 10 which demonstrates the conflicting effect of β on the quality of counterfactuals and the accuracy of the model.

Model	IB-INN β	IB-INN BPD	IB-INN Err.	Inception-V3 Err.
FakeMNIST	1.4*	1.77	0%	-
MNIST	1.4*	1.89	0.85%	-
CelebA-HQ				
Smile	1	3.32	7.42%	6.62%
High cheekbones	1	3.09	14.38%	13.74%
Lipstick	1	3.06	4.87%	5.70%
Heavy makeup	1	3.08	12.68%	10.84%

Table 1. Hyperparameters, negative log-likelihood measured in bits per dimension (BPD), and error rates for the models used in this work. *1.4 was rounded from 1.4265.

D. Additional Samples

We include pdfs with extra samples of all figures from the experiments. For each figure, there is a corresponding pdf in the related work zip-file. For example, Figure 3 has a corresponding pdf in the supplementary material named `figure3.pdf` with additional samples.

[4] IB-INN code: https://github.com/VLL-HD/IB-INN

[5] Inception-V3 code: https://pytorch.org/vision/stable/models.html#inception-v3

[6] For an invertible mapping f and $Z = f(X)$, $\mathcal{L}_{IB}$ is, in fact, ill-defined, and the authors [2] add noise to X to overcome the issue.

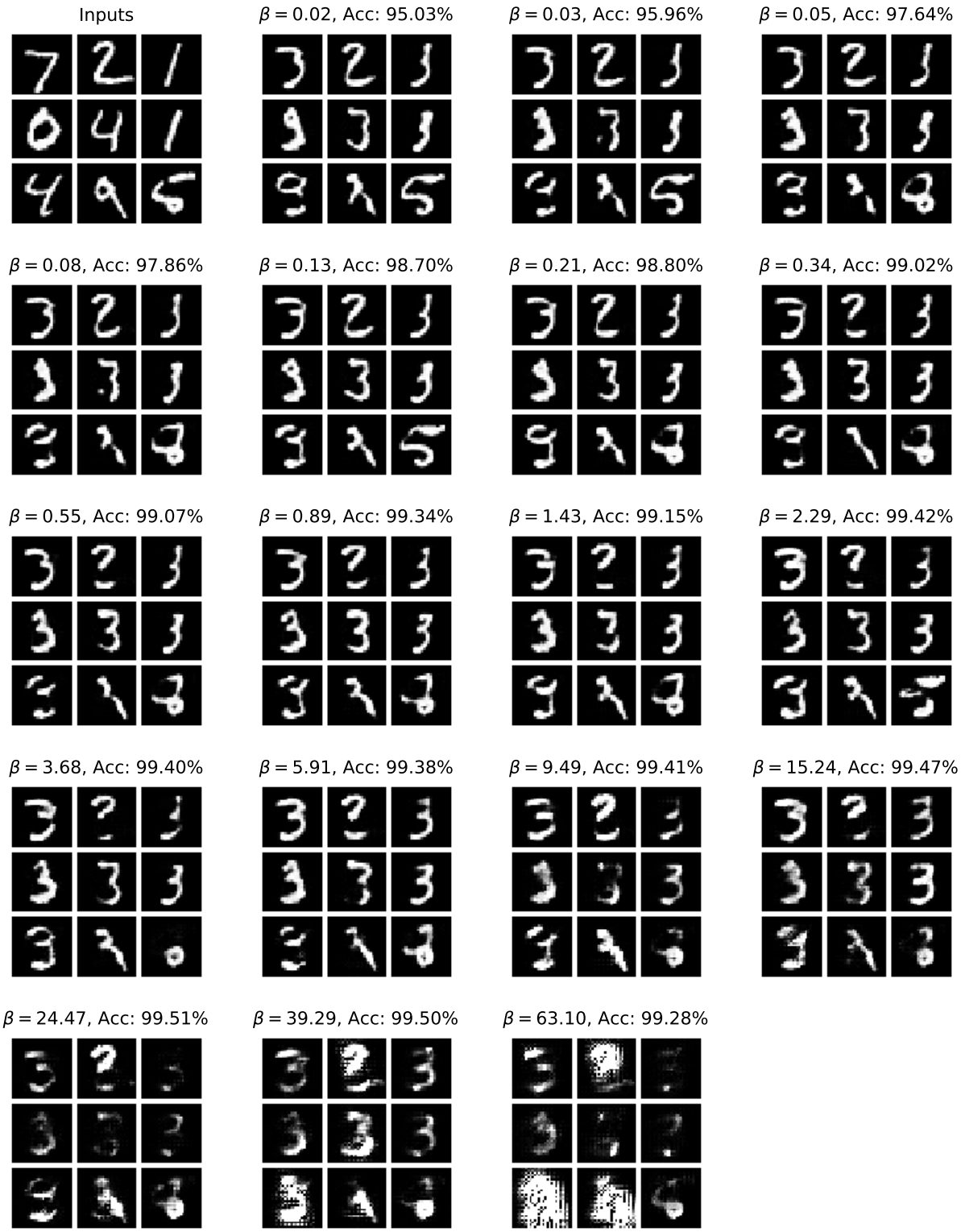


Figure 10. Counterfactual examples for MNIST models trained with different values of β. The top left square represents the input images that are all changed with target $q = 3$. Above plots are β-values in ascending order and corresponding test set accuracies.