GETTING A CLUE: A METHOD FOR EXPLAINING UNCERTAINTY ESTIMATES

Javier Antorán
University of Cambridge
ja666@cam.ac.uk

Umang Bhatt
University of Cambridge
usb20@cam.ac.uk

Tameem Adel
University of Cambridge
University of Liverpool
tah47@cam.ac.uk

Adrian Weller
University of Cambridge
The Alan Turing Institute
aw665@cam.ac.uk

José Miguel Hernández-Lobato
University of Cambridge
The Alan Turing Institute
jmh233@cam.ac.uk

ABSTRACT

Both uncertainty estimation and interpretability are important factors for trustworthy machine learning systems. However, there is little work at the intersection of these two areas. We address this gap by proposing a novel method for interpreting uncertainty estimates from differentiable probabilistic models, like Bayesian Neural Networks (BNNs). Our method, Counterfactual Latent Uncertainty Explanations (CLUE), indicates how to change an input, while keeping it on the data manifold, such that a BNN becomes more confident about the input’s prediction. We validate CLUE through 1) a novel framework for evaluating counterfactual explanations of uncertainty, 2) a series of ablation experiments, and 3) a user study. Our experiments show that CLUE outperforms baselines and enables practitioners to better understand which input patterns are responsible for predictive uncertainty.

1 INTRODUCTION

There is growing interest in probabilistic machine learning models, which aim to provide reliable estimates of uncertainty about their predictions (MacKay, 1992). These estimates are helpful in high-stakes applications such as predicting loan defaults or recidivism, or in work towards autonomous vehicles. Well-calibrated uncertainty can be as important as making accurate predictions, leading to increased robustness of automated decision-making systems and helping prevent systems from behaving erratically for out-of-distribution (OOD) test points. In practice, predictive uncertainty conveys skepticism about a model’s output. However, its utility need not stop there: we posit predictive uncertainty could be rendered more useful and actionable if it were expressed in terms of model inputs, answering the question: "Which input patterns lead my prediction to be uncertain?"

Understanding which input features are responsible for predictive uncertainty can help practitioners learn in which regions the training data is sparse. For example, when training a loan default predictor, a data scientist (i.e., practitioner) can identify sub-groups (by age, gender, race, etc.) under-represented in the training data. Collecting more data from these groups, and thus further constraining their model’s parameters, could lead to accurate predictions for a broader range of clients. In a clinical scenario, a doctor (i.e., domain expert) can use an automated decision-making system to assess whether a patient should receive a treatment. In the case of high uncertainty, the system would suggest that the doctor should not rely on its output. If uncertainty were explained in terms of which features the model finds anomalous, the doctor could appropriately direct their attention.

While explaining predictions from deep models has become a burgeoning field (Montavon et al., 2018; Bhatt et al., 2020b), there has been relatively little research on explaining what leads to neural networks’ predictive uncertainty. In this work, we introduce Counterfactual Latent Uncertainty Explanations (CLUE), to our knowledge, the first approach to shed light on the subset of input space features that are responsible for uncertainty in probabilistic models. Specifically, we focus on explaining Bayesian Neural Networks (BNNs). We refer to the explanations given by our method as
CLUEs. CLUEs try to answer the question: “What is the smallest change that could be made to an input, while keeping it in distribution, so that our model becomes certain in its decision for said input?” CLUEs can be generated for tabular and image data on both classification and regression tasks.

An application of CLUE is to improve transparency in the real-world deployment of a probabilistic model, such as a BNN, by complementing existing approaches to model interpretability (Ribeiro et al., 2016; Sundararajan et al., 2017; Chang et al., 2019). When the BNN is confident in its prediction, practitioners can generate an explanation via earlier feature importance techniques. When the BNN is uncertain, its prediction may well be wrong. This potentially wrong prediction could be the result of factors not related to the actual patterns present in the input data, e.g. parameter initialization, randomness in mini-batch construction, etc. An explanation of an uncertain prediction will be disproportionately affected by these factors. Indeed, recent work on feature attribution touches on the unreliability of saliency maps when test points are OOD (Adebayo et al., 2020). Therefore, when the BNN is uncertain, it makes sense to provide an explanation of why the BNN is uncertain (i.e., CLUE) instead of an explanation of the BNN’s prediction. This is illustrated in Figure 1.

Our code is at: [github.com/cambridge-mlg/CLUE](https://github.com/cambridge-mlg/CLUE)

- We introduce CLUE, an approach that finds counterfactual explanations of uncertainty in input space, by searching in the latent space of a deep generative model (DGM). We put forth an algorithm for generating CLUEs and show how CLUEs are best displayed.
- We propose a computationally grounded approach for evaluating counterfactual explanations of uncertainty. It leverages a separate conditional DGM as a synthetic data generator, allowing us to quantify how well explanations reflect the true generative process of the data.
- We evaluate CLUE quantitatively through comparison to baseline approaches under the above framework and through ablative analysis. We also perform a user study, showing that CLUEs allow practitioners to predict on which new inputs a BNN will be uncertain.

## 2 Preliminaries

### 2.1 Uncertainty in BNNs

Given a dataset \( \mathcal{D} = \{x^{(n)}, y^{(n)}\}_{n=1}^N \), a prior on our model’s weights \( p(w) \), and a likelihood function \( p(D|w) = \prod_{n=1}^N p(y^{(n)}|x^{(n)}, w) \), the posterior distribution over the predictor’s parameters \( p(w|D) \propto p(D|w)p(w) \) encodes our uncertainty about what value \( w \) should take. Through marginalization, this parameter uncertainty is translated into predictive uncertainty, yielding reliable error bounds and preventing overfitting:

\[
p(y^*|x^*, D) = \int p(y^*|x^*, w)p(w|D) \, dw. \tag{1}
\]

For BNNs, both the posterior over parameters and predictive distribution (1) are intractable. Fortunately, there is a rich literature concerning approximations to these objects (MacKay [1992]; Hernández-Lobato & Adams [2015]; Gal [2016]). In this work, we use scale-adapted Stochastic Gradient Hamiltonian Monte Carlo (SG-HMC) (Springenberg et al., 2016). For regression, we use...
heteroscedastic Gaussian likelihood functions, quantifying uncertainty using their standard deviation, \( \sigma(y|x) \). For classification, we take the entropy \( H(y|x) \) of categorical distributions as uncertainty. Details are given in Appendix \( B \). Hereafter, we use \( \mathcal{H} \) to refer to any uncertainty metric, be it \( \sigma \) or \( \mathcal{H} \).

Predictive uncertainty can be separated into two components, as shown in Figure 2. Each conveys different information to practitioners (Depeweg, 2019). Irreducible or \textit{aleatoric uncertainty} is caused by inherent noise in the generative process of the data, usually manifesting as class overlap. Model or \textit{epistemic uncertainty} represents our lack of knowledge about \( w \). Stemming from a model being under-specified by the data, epistemic uncertainty arises when we query points off the training manifold. Capturing model uncertainty is the main advantage of BNNs over regular NNs. It enables the former to be used for uncertainty aware tasks, such as OOD detection (Daxberger & Hernández-Lobato, 2019), continual learning (Nguyen et al., 2018), active learning (Depeweg et al., 2018), and Bayesian optimization (Springenberg et al., 2016).

2.2 Uncertainty Sensitivity Analysis

To the best of our knowledge, the only existing method for interpreting uncertainty estimates is Uncertainty Sensitivity Analysis (Depeweg et al., 2017). This method quantifies the global importance of an input dimension to a chosen metric of uncertainty \( \mathcal{H} \) using a sum of linear approximations centered at each test point:

\[
I_i = \frac{1}{|D_{\text{test}}|} \sum_{n=1}^{|D_{\text{test}}|} \left| \frac{\partial \mathcal{H}(y_n|x_n)}{\partial x_{n,i}} \right|.
\]  

As discussed by Rudin (2019), linear explanations of non-linear models, such as BNNs, can be misleading. Even generalized linear models, which are often considered to be “inherently interpretable,” like logistic regression, produce non-linear uncertainty estimates in input space. This can be seen in Figure 2. Furthermore, high-dimensional input spaces limit the actionability of these explanations, as \( \nabla_x \mathcal{H} \) will likely not point in the direction of the data manifold. In Figure 3 and Appendix D, we show how this can result in sensitivity analysis generating meaningless explanations.

Our method, CLUE, leverages the latent space of a DGM to avoid working with high-dimensional input spaces and to ensure explanations are in-distribution. CLUE does not rely on crude linear approximations. The counterfactual nature of CLUE guarantees explanations have tangible meaning.

Figure 3: Left: Taking a step in the direction of maximum sensitivity leads to a seemingly noisy input configuration for which \( \mathcal{H} \) is small. Right: Minimizing CLUE’s uncertainty-based objective in terms of a DGM’s latent variable \( z \) produces a plausible digit with a corrected lower portion.
2.3 Counterfactual Explanations

The term “counterfactual” captures notions of what would have happened if something had been different. Two meanings have been used by ML subcommunities. 1) Those in causal inference make causal assumptions about interdependencies among variables and use these assumptions to incorporate consequential adjustments when particular variables are set to new values (Kusner et al., 2017; Pearl, 2019). 2) In contrast, the interpretability community recently used “counterfactual explanations” to explore how input variables must be modified to change a model’s output without making explicit causal assumptions (Wachter et al., 2018). As such, counterfactual explanations can be seen as a case of contrastive explanations (Dhurandhar et al., 2018; Byrnc, 2019). In this work, we use “counterfactual” in a sense similar to 2): we seek to make small changes to an input in order to reduce the uncertainty assigned to it by our model, without explicit causal assumptions.

Multiple counterfactual explanations can exist for any given input, as the functions we are interested in explaining are often non-injective (Russell, 2019). We are concerned with counterfactual input configurations that are close to the original input \( x_0 \) according to some pairwise distance metric \( d(\cdot, \cdot) \).

Given a desired outcome \( c \) different from the original one \( y_0 \) produced by predictor \( p_f \), counterfactual explanations \( x_c \) are usually generated by solving an optimization problem that resembles:

\[
x_c = \arg \max_{x} \left( p_f(y=c|x) - d(x, x_0) \right) \quad \text{s.t.} \quad y_0 \neq c.
\]

Naively optimizing \( 3 \) in high-dimensional input spaces may result in the creation of adversarial inputs which are not actionable (Goodfellow et al., 2015). Telling a person that they would have been approved for a loan had their age been \(-10\) is of very little use. To right this, recent works define linear constraints on explanations (Ustun et al., 2019; Sharma et al., 2020). An alternative more amenable to high dimensional data is to leverage DGMs (which we dub auxiliary DGMs) to ensure explanations are in-distribution (Dhurandhar et al., 2018; Joshi et al., 2018; Chang et al., 2019; Booth et al., 2020; Tripp et al., 2020). CLUE avoids the above issues by searching for counterfactuals in the lower-dimensional latent space of an auxiliary DGM. This choice is well suited for uncertainty, as the DGM constrains CLUE’s search space to the data manifold. When faced with an OOD input, CLUE returns the nearest in-distribution analog, as shown in Figure 3.

3 Proposed Method

Without loss of generality, we use \( \mathcal{H} \) to refer to any differentiable estimate of uncertainty (\( \sigma \) or \( H \)). We introduce an auxiliary latent variable DGM: \( p_\theta(x) = \int p_\theta(x|z)p(z) \, dz \). In the rest of this paper, we will use the decoder from a variational autoencoder (VAE). Its encoder is denoted as \( q_\theta(z|x) \). We write these models’ predictive means as \( E_{p_\theta(x|z|x)}(x) = \mu_\theta(x|z) \) and \( E_{q_\theta(z|x)}(z) = \mu_\phi(z|x) \) respectively.

CLUE aims to find points in latent space which generate inputs similar to an original observation \( x_0 \) but are assigned low uncertainty. This is achieved by minimizing \( 4 \). CLUEs are then decoded as \( 5 \).

\[
\mathcal{L}(z) = \mathcal{H}(y|\mu_\theta(x|z)) + d(\mu_\theta(x|z), x_0), \quad \text{(4)}
\]

\[
x_{\text{CLUE}} = \mu_\theta(x|z_{\text{CLUE}}) \quad \text{where} \quad z_{\text{CLUE}} = \arg \min_{z} \mathcal{L}(z). \quad \text{(5)}
\]

The pairwise distance metric takes the form \( d(x, x_0) = \lambda_x d_x(x, x_0) + \lambda_y d_y(f(x), f(x_0)) \) such that we can enforce similarity between uncertain points and CLUEs in both input and prediction space. The hyperparameters \( (\lambda_x, \lambda_y) \) control the trade-off between producing low uncertainty CLUEs and CLUEs which are close to the original inputs. In this work, we take \( d_x(x, x_0) = \|x - x_0\|_1 \) to encourage sparse explanations. For regression, \( d_y(f(x), f(x_0)) \) is mean squared error. For classification, we use cross-entropy. Note that the best choice for \( d(\cdot, \cdot) \) will be task-specific.

The CLUE algorithm and a diagram of our procedure are provided in Algorithm 1 and Figure 4 respectively. The hyperparameter \( \lambda_x \) is selected by cross validation for each dataset such that both terms in \( 4 \) are of similar magnitude. We set \( \lambda_y \) to 0 for our main experiments, but explore different values in Appendix H.1. We minimize \( 4 \) with Adam by differentiating through both our BNN and VAE decoder. To facilitate optimization, the initial value of \( z \) is chosen to be \( z_0 = \mu_\phi(z|x_0) \). Optimization runs for a minimum of three iterations and a maximum of 35 iterations, with a learning rate of 0.1. If the decrease in \( \mathcal{L}(z) \) is smaller than \( \mathcal{L}(z_0)/100 \) for three consecutive iterations, we apply early stopping. CLUE can be applied to batches of inputs simultaneously, allowing us to leverage GPU-accelerated matrix computation. Our implementation is detailed in full in Appendix B.
Algorithm 1: CLUE

Inputs: original datapoint $x_0$, distance function $d(\cdot, \cdot)$, Uncertainty estimator $\mathcal{H}$, DGM decoder $\mu_\theta(\cdot)$, DGM encoder $\mu_\phi(\cdot)$

1. Set initial value of $z = \mu_\phi(z|x_0)$;
2. while loss $\mathcal{L}$ is not converged do
   1. Decode: $x = \mu_\theta(x|z)$;
   2. Use predictor to obtain $\mathcal{H}(y|x)$;
   3. $\mathcal{L} = \mathcal{H}(y|x) + d(x, x_0)$;
   4. Update $z$ with $\nabla_z \mathcal{L}$;
3. end
4. Decode explanation: $x_{\text{CLUE}} = \mu_\theta(x|z)$;

Output: Uncertainty counterfactual $x_{\text{CLUE}}$

As noted by Wachter et al. (2018), individual counterfactuals may not shed light on all important features. Fortunately, we can exploit the non-convexity of CLUE’s objective to address this. We initialize CLUE with $z_0 = \mu_\phi(z|x_0) + \epsilon$, where $\epsilon = N(z; 0, \sigma_0 \mathbf{I})$, and perform Algorithm 1 multiple times to obtain different CLUEs. We find $\sigma_0 = 0.15$ to give a good trade-off between optimization speed and CLUE diversity. Appendix C shows examples of different CLUEs obtained for the same inputs.

We want to ensure noise from auxiliary DGM reconstruction does not affect CLUE visualization. For tabular data, we use the change in percentile of each input feature with respect to the training distribution as a measure of importance. We only highlight continuous variables for which CLUEs are separated by 15 percentile points or more from their original inputs. All changes to discrete variables are highlighted. For images, we report changes in pixel values by applying a sign-preserving quadratic function to the difference between CLUEs and original samples: $\Delta \text{CLUE} = |\Delta x| - \Delta x$ with $\Delta x = x_{\text{CLUE}} - x_0$. This is showcased in Figure 5 and in Appendix G. It is common for approaches to generating saliency maps to employ constraints that encourage the contiguity of highlighted pixels (Chang et al. 2019, Dabkowski & Gal 2017). We do not employ such constraints, but we note they might prove useful when applying CLUE to natural images.

4 A FRAMEWORK FOR EVALUATING COUNTERFACTUAL EXPLANATIONS OF UNCERTAINTY COMPUTATIONALLY

Evaluating explanations quantitatively (without resorting to expensive user studies) is a difficult but important task (Doshi-Velez & Kim 2017, Weller 2019). We put forth a computational framework to evaluate counterfactual explanations of uncertainty. In the spirit of Bhatt et al. (2020b), we desire counterfactuals that are 1) informative: they should highlight features which affect our BNN’s uncertainty, and 2) relevant: counterfactuals should lie close to the original inputs and represent plausible parameter settings, lying close to the data manifold. Recall, from Figure 5 that inputs for which our BNN is certain can be constructed by applying adversarial perturbations to uncertain ones. Alas, evaluating these criteria requires access to the generative process of the data.

To evaluate the above requirements, we introduce an additional DGM that will act as a “ground truth” data generating process (g.t. DGM). Specifically, we use a variational autoencoder with arbitrary conditioning (Vanov et al. 2019) (g.t. VAEAC). It jointly models inputs and targets $p_{\theta}(x, y)$. Measuring counterfactuals’ log density under this model $\log p_{\theta}(x_c)$ allows us to evaluate if they are in-distribution. The g.t. VAEAC also allows us to query the conditional distribution over targets given inputs, $p_{\theta}(y|x)$. From this distribution, we can compute an input’s true uncertainty $\mathcal{H}_{gt}$, as given by the generative process of the data. This allows us to evaluate if counterfactuals address the true
1. Train g.t. VAEAC on real data
2. Train BNN, VAE on artificial data
3. Generate counterfactuals for artificial data
4. Evaluate counterfactuals with g.t. VAEAC

Figure 6: Pipeline for computational evaluation of counterfactual explanations of uncertainty. The VAEAC which we treat as a data generating process is colored in green. Colored in orange is the auxiliary DGM used by the approach being evaluated. For approaches that do not use an auxiliary DGM, like Uncertainty Sensitivity Analysis, the orange element will not be present.

We validate CLUE on LSAT academic performance regression (Wightman et al., 1998), UCI Wine quality regression, UCI Credit classification (Dua & Graff 2017), a 7 feature variant of COMPAS recidivism classification (Angwin et al.), and MNIST image classification (LeCun & Cortes 2010). For each, we select roughly the 20% most uncertain test points as those for which we reject our BNNs’ decisions. We only generate CLUEs for “rejected” points. Rejection thresholds, architectures, and hyperparameters are in Appendix B. Experiments with non-Bayesian NNs are in Appendix H.1.

As a baseline, we introduce a localized version of Uncertainty Sensitivity Analysis (Depeweg et al. 2017). It produces counterfactuals by taking a single step, of size \( \eta \), in the direction of the gradient of an input’s uncertainty estimates \( \mathbf{x}_c = \mathbf{x}_0 - \eta \nabla_x \mathcal{H}(y|x_0) \). Averaging \( |\mathbf{x}_0 - \mathbf{x}_c| \) across a test set, we recover (2). As a second baseline, we adapt FIDO (Chang et al. 2019), a counterfactual feature importance method, to minimize uncertainty. We dub this approach U-FIDO. This method places a feature importance method, to minimize uncertainty. We dub this approach U-FIDO. This method places a

\[
\mathcal{L}(\rho) = \mathbb{E}_{p_{\rho}(b)}[\mathcal{H}(y|x_c(b)) + \lambda_1 \|b\|_1], \quad (6)
\]

\[
\mathbf{x}_c(b) = b \odot \mathbf{x}_0 + (1 - b) \odot \mathbb{E}_{p(x_B|x_C \setminus B)}[x_B]. \quad (7)
\]

Counterfactuals are generated by (7), where \( \odot \) is the Hadamard product. We compare CLUE to feature importance methods (Ribeiro et al. 2016; Lundberg & Lee 2017) in Appendix F.
5.1 Computational Evaluation

We compare CLUE, Localized Sensitivity, and U-FIDO using the evaluation framework put forth in Section 4. We would like counterfactuals to explain away as much uncertainty as possible while staying as close to the original inputs as possible. We manage this informativeness (large \(\Delta H_{gt}\)) to relevance (small \(\|\Delta x\|_1\)) trade-off with the hyperparameters \(\eta, \lambda_e,\) and \(\lambda_b\) for Local Sensitivity, CLUE, and U-FIDO, respectively. We perform a logarithmic grid search over hyperparameters and plot Pareto-like curves. Our two metrics of interest take minimum values of 0 but their maximum is dataset and method dependent. For Sensitivity, \(\|\Delta x\|_1\) grows linearly with \(\eta\). For CLUE and U-FIDO, these metrics saturate for large and small values of \(\lambda_e\) (or \(\lambda_b\)). As a result, the values obtained by these methods do not overlap. As shown in Figure 7 CLUE is able to explain away more uncertainty (\(\Delta H_{gt}\)) than U-FIDO, and U-FIDO always obtains smaller values of \(\|\Delta x\|_1\) than CLUE.

To construct a single performance metric, we scale all measurements by the maximum values obtained between U-FIDO or CLUE, e.g. \((\sqrt{2 \cdot \max(\Delta H_{gt} \text{U-FIDO}, \Delta H_{gt} \text{CLUE})})^{-1},\) linearly mapping them to \([0, 1/\sqrt{2}].\) We then negate \(\Delta H_{gt},\) making its optimum value 0. We consider each method’s best performing hyperparameter configuration, as determined by its curve’s point nearest the origin, or knee-point. The euclidean distance from each method’s knee-point to the origin acts as a metric of relative performance. The best value is 0 and the worst is 1. Knee-point distances, computed across three runs, are shown for both uncertainty types in Table I.

We perform an analogous experiment where relevance is quantified as proximity to the data manifold: \(\Delta \log p_{gt} = \min(0, \log p_{gt}(x_e) - \log p_{gt}(x_0)).\) Here, \(\log p_{gt}(x_0)\) refers to the log-likelihood of the artificial data for which counterfactuals are generated. Results are in Table 2, where CLUE performs best in 8/10 tasks. Generating counterfactuals with a VAE ensures that CLUEs are relevant. In Appendix H.2 we perform an analogous informativeness vs relevance experiment on real data. We obtain similar results to our computational evaluation framework, validating its reliability.


\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\text{Method} & \text{LSAT (4)} & \text{COMPAS (7)} & \text{Wine (11)} & \text{Credit (23)} & \text{MNIST (784)} \\
\hline
\text{Sensitivity} & 0.70 & 0.67 & 0.01 & 0.03 & 0.63 & 0.50 & 0.66 & 0.68 \\
\text{CLUE} & 0.52 & 0.64 & 0.01 & 0.14 & 0.52 & 0.29 & 0.26 & 0.27 \\
\text{U-FIDO} & 0.36 & 0.51 & 0.71 & 0.31 & 0.22 & 0.02 & 0.45 & 0.63 & 0.38 & 0.50 \\
\hline
\end{array}
\]

Figure 7: MNIST knee-points.

5.2 User Study

Human-based evaluation is a key step in validating the utility of tools for ML explainability (Hoffman et al., 2018). We want to assess the extent to which CLUEs help machine learning practitioners identify sources of uncertainty in ML models compared to using simple linear approximations (Local Sensitivity) or human intuition. To do this, we propose a forward-simulation task (Doshi-Velez & Kim, 2017), focusing on an appropriate local test to evaluate CLUEs. We show practitioners one
Participant A selects a test point at random from the test set.

2) Participant A pairs the selected point with an uncertain context point.

3) Generate certain context point based on method being evaluated.

4) Participants identify the certainty of the test point given the two context points.

Figure 8: Experimental workflow for our tabular data user study.

Figure 9: Example question shown to main survey participants for the COMPAS dataset: Given the uncertain example on the left and the certain example in the middle, will the model be certain on the test example on the right? The red text highlights the features that differ between context points.

datapoint below our “rejection” threshold and one datapoint above. The former is labeled as “certain” and the latter as “uncertain”; we refer to these as context points. The certain context point serves as a local counterfactual explanation for the uncertain context point. Using both context points for reference, practitioners are asked to predict whether a new test point will be above or below our threshold (i.e., will our BNN’s uncertainty be high or low for the new point). Our survey compares the utility of the certain context points generated by CLUE relative to those from baselines.

In our survey, we compare four different methods, varying how we select certain context points. We either 1) select a certain point at random from the test set as a control, generate a counterfactual certain point with 2) Local Sensitivity or with 3) CLUE, or 4) display a human selected certain point (Human CLUE). To generate a Human CLUE, we ask participants (who will not take the main survey) to pair uncertain context points with similar certain points. We select the points used in our main survey with a pilot procedure similar to Grgic-Hlaca et al. (2018). This procedure, shown in Figure 8, prevents us from injecting biases into point selection and ensures context points are relevant to test points. In our procedure, a participant is shown a pool of randomly selected certain and uncertain points. We ask this participant to select points from this pool: these will be test points. We then ask the participant to map each selected test point to a similar uncertain point without replacement. In this way, we obtain uncertain context points that are relevant to test points.

We use the LSAT and COMPAS datasets in our user study. Ten different participants take each variant of the main survey: our participants are ML graduate students, who serve as proxies for practitioners in industry. The main survey consists of 18 questions, 9 per dataset. An example question is shown in Figure 9. The average participant accuracy by variant is: CLUE (82.22%), Human CLUE (62.22%), Random (61.67%), and Local Sensitivity (52.78%). We measure the statistical significance of CLUE’s superiority with unpaired Wilcoxon signed-rank tests (Demšar, 2006) of CLUE vs each baseline. We obtain the following p-values: Human-CLUE ($2.34\times10^{-5}$), Random ($1.47\times10^{-5}$), and Sensitivity ($2.60\times10^{-9}$)\footnote{Our between-group experimental design could potentially result in dependent observations, which violates the tests’ assumptions. However, the p-values obtained are extremely low, providing confidence in rejecting the null hypothesis of equal performance among approaches.}.

Additional analysis is included in Appendix H.3.
We study CLUE’s reliance on its auxiliary DGM. Further ablative analysis is found in Appendix H. We address this issue with Counterfactual Latent Uncertainty Explanations (CLUE), a method that creates a need for being able to interpret why deep learning methods are uncertain. In turn, this informs stakeholders when it does not know the correct prediction (Bhatt et al., 2021). We run a smaller scale study, wherein we show participants larger sets of context points. Results are in Appendix J.2

5.3 Analysis of CLUE’s Auxiliary Deep Generative Model

We compare Algorithm 1’s encoder-based initialization $z_0 = \mu_\phi(z|x_0)$ with $z_0 = 0$. As shown in Figure 10, for high dimensional datasets, like MNIST, initializing $z$ with the encoder’s mean leads to CLUEs that require smaller changes in input space to explain away similar amounts of uncertainty (i.e., more relevant). In Appendix H.1, similar behavior is observed for Credit, our second highest dimensional dataset. On other datasets, both approaches yield indistinguishable results. CLUEs could plausibly be generated with differentiable DGMs that lack an encoding mechanism, such as GANs. However, an appropriate initialization strategy should be employed.

Capacity of CLUE’s DGM: Figure 10 shows how auto-encoding uncertain MNIST samples with low-capacity VAEs significantly reduces these points’ predictive entropy. CLUEs generated with these VAEs highlight features that the VAEs are unable to reproduce but are not reflective of our BNN’s uncertainty. This results in large values of $||\Delta x||_1$: although the counterfactual examples are indeed more certain than the original samples, they contain unnecessary changes. As our auxiliary DGMs’ capacity increases, the amount of uncertainty preserved when auto-encoding inputs increases as well. $||\Delta x||_1$ decreases while the predictive entropy of our CLUEs stays the same. More expressive DGMs allow for generating sparser, more relevant, CLUEs. Fortunately, even in scenarios where our predictor’s training dataset is limited, we can train powerful DGMs by leveraging unlabeled data.

6 Conclusion

With the widespread adoption of data-driven decision making has come a need for the development of ML tools that can be trusted by their users. This has spawned a subfield of ML dedicated to interpreting deep learning systems’ predictions. A transparent deep learning method should also inform stakeholders when it does not know the correct prediction (Bhatt et al., 2021). In turn, this creates a need for being able to interpret why deep learning methods are uncertain.

We address this issue with Counterfactual Latent Uncertainty Explanations (CLUE), a method that reveals which input features can be changed to reduce the uncertainty of a probabilistic model, like a BNN. We then turn to assessing the utility, to stakeholders, of counterfactual explanations of predictive uncertainty. We put forth a framework for computational evaluation of these types of explanations. Quantitatively, CLUE outperforms simple baselines. Finally, we perform a user study. It finds that users are better able to predict their models’ behavior after being exposed to CLUEs.
ACKNOWLEDGEMENTS

JA acknowledges support from Microsoft Research through its PhD Scholarship Program. UB acknowledges support from DeepMind and the Leverhulme Trust via the Leverhulme Centre for the Future of Intelligence (CFI) and from the Mozilla Foundation. AW acknowledges support from a Turing AI Fellowship under grant EP/V025379/1, The Alan Turing Institute under EPSRC grant EP/N510129/1 & TU/B/000074, and the Leverhulme Trust via CFI.

REFERENCES


Bin Dai and David Wipf. Diagnosing and enhancing VAE models. In *International Conference on Learning Representations*, 2019. URL [https://openreview.net/forum?id=Ble0X3C9tQ](https://openreview.net/forum?id=Ble0X3C9tQ).


APPENDIX

This appendix is formatted as follows.

1. We discuss the datasets used in Appendix A.
2. Implementation details for our experiments are provided in Appendix B.
3. We provide examples of the multiplicity of CLUEs in Appendix C.
4. We discuss the application of uncertainty sensitivity analysis in high dimensional spaces in Appendix D.
5. We visualize CLUE’s optimization in the latent space in Appendix E.
6. We compare CLUE to existing feature importance techniques in Appendix F.
7. We provide additional examples of CLUEs and U-FIDO counterfactuals in Appendix G.
8. We provide additional experimental results in Appendix H.
9. We note additional details of our computational evaluation framework for counterfactual explanations of uncertainty in Appendix I.
10. We include more details on the setup of our user studies in Appendix J.
We employ 5 datasets in our experiments, 4 tabular and one composed of images. All of them are publicly available. Their details are given in Table 3.

<table>
<thead>
<tr>
<th>Name</th>
<th>Targets</th>
<th>Input Type</th>
<th>N. Inputs</th>
<th>N. Train</th>
<th>N. Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSAT</td>
<td>Continuous</td>
<td>Continuous &amp; Categorical</td>
<td>4</td>
<td>17432</td>
<td>4358</td>
</tr>
<tr>
<td>COMPAS</td>
<td>Binary</td>
<td>Continuous &amp; Categorical</td>
<td>7*</td>
<td>5554</td>
<td>618</td>
</tr>
<tr>
<td>Wine (red)</td>
<td>Continuous</td>
<td>Continuous</td>
<td>11</td>
<td>1438</td>
<td>160</td>
</tr>
<tr>
<td>Credit</td>
<td>Binary</td>
<td>Continuous &amp; Categorical</td>
<td>24</td>
<td>27000</td>
<td>3000</td>
</tr>
<tr>
<td>MNIST</td>
<td>Categorical</td>
<td>Image (grayscale)</td>
<td>28×28</td>
<td>60000</td>
<td>10000</td>
</tr>
</tbody>
</table>

We use the LSAT loading script from Cole & Williamson (2019)’s github page. The raw data can be downloaded from (https://raw.githubusercontent.com/throwaway20190523/MonotonicFairness/master/data/law_school_cf_test.csv) and (https://raw.githubusercontent.com/throwaway20190523/MonotonicFairness/master/data/law_school_cf_train.csv).

For the COMPAS criminal recidivism prediction dataset we use a modified version of Zafar et al. (2017)’s loading and pre-processing script. It can be found at (https://github.com/mbilalzafar/fair-classification/blob/master/disparate_mistreatment/propublica_compas_data_demo/load_compas_data.py).

We add an additional feature: “days served” which we compute as the difference, measured in days, between the “c_jail_in” and “c_jail_out” variables. The raw data is found at (https://github.com/propublica/compas-analysis/blob/master/compas-scores-two-years.csv).

The red wine quality prediction dataset can be obtained from and is described in detail at (https://archive.ics.uci.edu/ml/datasets/wine+quality).

The default of credit card clients dataset, which we refer to as “Credit” in this work, can be obtained from and is described in detail at (https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients). Note that this dataset is different from the also commonly used German credit dataset.

The MNIST handwritten digit image dataset can be obtained from (http://yann.lecun.com/exdb/mnist/).
B IMPLEMENTATION DETAILS

B.1 INFEERENCE IN BNNs

We choose a Monte Carlo (MC) based inference approach for our BNNs due to these not being limited to localized approximations of the posterior. Specifically, we make use of scale adapted SG-HMC (Springenberg et al., 2016), an approach to stochastic gradient Hamiltonian Monte Carlo with automatic hyperparameter discovery. This technique estimates the mass matrix and the noise introduced by stochasticity in the gradients using exponentially decaying moving average filters during the chain’s burn-in phase. We use a fixed step size of $\epsilon = 0.01$ and batch sizes of 512. We set a diagonal 0 mean Gaussian prior $p(w) = \mathcal{N}(w; 0, \sigma_w^2 \cdot I)$ over each layer of weights. We place a per-layer conjugate Gamma hyperprior over $\sigma_w^2$ with parameters $\alpha = \beta = 10$. We periodically update $\sigma_w^2$ for each layer using Gibbs sampling.

On MNIST, we burn in our chain for 25 epochs, using the first 15 to estimate SG-HMC parameters. We re-sample momentum parameters every 10 steps and perform a Gibbs sweep over the prior variances every 45 steps. We save parameter settings every 2 epochs until a total of 300 sets of weights are saved. This makes for a total of 625 epochs.

For tabular datasets, we perform a burn-in of 400 epochs, using the first 120 to estimate SG-HMC parameters. We save weight configurations every 20 epochs until a total of 100 sets if weights are saved. This makes for a total of 2500 epochs. Momentum is re-sampled every 10 epochs and the prior over weights is re-sampled every 50 epochs. We use a batch size of 512 for all datasets.

B.2 COMPUTING UNCERTAINTY ESTIMATES

In this work, we consider NNs which parametrize two types of distributions over target variables: the categorical for classification problems and the Gaussian for regression. For classification, our likelihood function is $p(y|x, w) = \text{Cat}(y; f(x, w))$. Given a posterior distribution over weights $p(w|D)$, we use marginalization (1) to translate uncertainty in $w$ into uncertainty in predictions. Unfortunately, this operation is intractable for BNNs. We resort to approximating the predictive posterior with $M$ MC samples:

$$
    p(y^*|x^*, D) = E_{p(w|D)}[p(y^*|x^*, w)] = \frac{1}{M} \sum_{m=0}^{M} f(x^*, w); \quad w \sim p(w|D).
$$

The resulting predictive distribution is categorical. We quantify its uncertainty using entropy:

$$
    H(y^*|x^*, D) = \sum_{k=1}^{K} p(y^*=c_k|x^*, D) \log p(y^*=c_k|x^*, D).
$$

This quantity contains aleatoric and epistemic components ($H_a, H_e$). The former is estimated as:

$$
    H_a = E_{p(w|D)}[H(y^*|x^*, w)] = \frac{1}{M} \sum_{m} H(y^*|x^*, w); \quad w \sim p(w|D).
$$

The epistemic component can be obtained as the difference between the total and aleatoric entropies. This quantity is also known as the mutual information between $y^*$ and $w$:

$$
    H_e = I(y^*, w|x^*, D) = H(y^*|x^*, D) - E_{p(w|D)}[H(y^*|x^*, w)].
$$

For regression, we employ heteroscedastic likelihood functions. Their mean and variance are parametrized by our NN: $p(y^*|x^*, w) = \mathcal{N}(y; f_{\mu}(x^*, w), f_{\sigma^2}(x^*, w))$. Marginalizing over $w$ with MC induces a Gaussian mixture distribution over outputs. Its mean is obtained as:

$$
    \mu_a \approx \frac{1}{M} \sum_{m=0}^{M} f_{\mu}(x^*, w); \quad w \sim p(w|D).
$$
There is no closed-form expression for the entropy of this distribution. Instead, we use the variance of the GMM as an uncertainty metric. It also decomposes into aleatoric and epistemic components ($\sigma_a^2, \sigma_e^2$):

$$
\sigma^2(y^*|x^*, D) = \mathbb{E}_{p(w|D)}[\sigma^2(y^*|x^*, w)] + \sigma_p^2(y^*|x, w).
$$

These are also estimated with MC:

$$
\sigma^2(y^*|x^*, D) \approx \frac{1}{M} \sum_m \mu(y^*|x^*, w)^2 - \left( \frac{1}{M} \sum_m \mu(y^*|x^*, w) \right)^2 + \frac{1}{M} \sum_m \sigma^2(y^*|x^*, w); \ w \sim p(w|D).
$$

Here, $\sigma_e^2$ reflects model uncertainty - our lack of knowledge about $w$ - while $\sigma_a^2$ tells us about the irreducible uncertainty or noise in our training data.

In Figure 11, we show the fit obtained with a BNN with scale adapted SG-HMC on the toy moons dataset. We would like to highlight 2 key differences with respect to the logistic regression example shown in Figure 2. Neural networks are very flexible models. They are capable of perfectly fitting non-linear manifolds, such as moons. In consequence, when these models present aleatoric uncertainty it is most often due to the inputs not containing enough information to predict the targets. As little such noise exists in our particular instantiation of moons, our estimates of aleatoric entropy are close to 0. Despite their flexibility, selecting a NN involves adopting some inductive biases (Wilson & Izmailov, 2020). Additionally, unlike logistic regression, the weight space posterior of a BNN is very difficult to characterize. Both of these things are reflected in the BNN predictive posterior’s epistemic uncertainty only growing in the vertical axis, instead of in all directions.

![Figure 11: Left: Training points and BNN predictive distribution obtained on the moons dataset with SG-HMC. Center: Aleatoric entropy $H_a$ expressed by the model matches regions of class overlap. Right: Epistemic entropy $H_e$ grows as we move away from the data.](image)

**B.3 Architectures and Other Network Hyperparameters**

For all datasets, our BNNs are fully connected networks with residual connections. Auxiliary VAEs and VAEACs used for tabular data use fully connected encoders and decoders with residual connections and batch normalization at every layer. For MNIST, we employ 6 convolutional bottleneck residual blocks (He et al., 2016) for both encoders and decoders. We use the same architecture for the VAEACs used as ground truth generative models in the computationally grounded evaluation framework put forth in Section 4. Note that the ground truth VAEAC models have slightly larger input spaces due to them modeling inputs and targets jointly. All architectural hyperparameters are provided in Table 4.

In order to improve the artificial sample quality of our “ground truth” VAEACs, we leverage a two-stage VAE configuration (Dai & Wipf, 2019). For all datasets, the lower level VAEs use the standard tabular data VAE architecture described above, with 2 hidden layers. We use 300 hidden units for MNIST and 150 for other datasets. Additional details on our use of two-stage VAEs are provided in Appendix I.

We train all generative models with the RAdam optimizer (Liu et al., 2020) with a learning rate of $1e^{-4}$ for tabular data and $3e^{-4}$ for MNIST. We found RAdam to yield marginally better results than Adam.
Table 4: Network architecture hyperparameters used in all experiments. Depth refers to number of hidden layers or residual blocks. Latent dimension values marked with a star (*) refer to the second level VAEs for “ground truth” VAEACs.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>BNN Depth</th>
<th>BNN Width</th>
<th>VAE / VAEAC Depth</th>
<th>VAE Width</th>
<th>VAEAC Width</th>
<th>VAE / VAEAC Latent Dim</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSAT</td>
<td>2</td>
<td>200</td>
<td>3</td>
<td>300</td>
<td>350</td>
<td>4 (*4)</td>
</tr>
<tr>
<td>COMPAS</td>
<td>2</td>
<td>200</td>
<td>3</td>
<td>300</td>
<td>350</td>
<td>4 (*4)</td>
</tr>
<tr>
<td>Wine</td>
<td>2</td>
<td>200</td>
<td>3</td>
<td>300</td>
<td>350</td>
<td>6 (*6)</td>
</tr>
<tr>
<td>Credit</td>
<td>2</td>
<td>200</td>
<td>3</td>
<td>300</td>
<td>350</td>
<td>8 (*8)</td>
</tr>
<tr>
<td>MNIST</td>
<td>2</td>
<td>1200</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>20 (*8)</td>
</tr>
</tbody>
</table>

We convert categorical inputs to our BNNs into one-hot vectors. When building DGMs, we model continuous inputs with diagonal, unit variance (heteroscedastic) Gaussian distributions. This choice makes these models weigh all input dimensions equally, a desirable trait for explanation generation. We place categorical distributions over discrete inputs, expressing them as one-hot vectors. For the LSAT, COMPAS, and Credit datasets, where there are both continuous and discrete features, data likelihood values are obtained as the product of Gaussian likelihoods and categorical likelihoods. During the CLUE optimization procedure, we approximate gradients through one-hot vectors with the softmax function’s gradients. This is known as the softmax straight-through estimator (Bengio, 2013). It is biased but works well in practice. For MNIST, we model pixels as the probabilities of a product of Bernoulli distributions. We feed these probabilities directly into our BNNs and DGMs. We normalize all continuously distributed features such that they have 0 mean and unit variance. This facilitates model training and also ensures that all features are weighed equally under CLUE’s pairwise distance metric in (4). For MNIST, this normalization is applied to whole images instead of individual pixels. Categorical variables are not normalized. Changing a categorical variable implies changing two bits in the corresponding one-hot vector. This creates the same $\ell_1$ regularization penalty as shifting a continuously distributed variable two standard deviations.

B.4 CLUE Hyperparameters

As mentioned in Section 5, in our experiments we only apply CLUE to points that present uncertainty above a rejection threshold. The rejection thresholds used for each dataset are displayed in Table 5. The same table contains the values of $\lambda_x$ used in all experiments. In practice we define $\lambda_x = \lambda_x - d$, where $d$ is the input space dimensionality of a dataset. This makes the strength of CLUE’s pairwise input space distance metric agnostic to dimensionality. We choose a significantly larger value of $\lambda_x$ for MNIST due to there being a large number of pixels that are always black.

Table 5: Values of CLUE’s input space similarity weight $\lambda_x$ and uncertainty rejection thresholds used for all experiments. Next to each dataset’s name is the the type of uncertainty quantified: standard deviation ($\sigma$) or entropy ($H$). We report $\lambda_x$ upscaled by each dataset’s input dimensionality $d$.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LSAT ($\sigma$)</th>
<th>COMPAS ($H$)</th>
<th>Wine ($\sigma$)</th>
<th>Credit ($H$)</th>
<th>MNIST ($H$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_x \cdot d$</td>
<td>1.5</td>
<td>2</td>
<td>2.5</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>$H$ threshold</td>
<td>1</td>
<td>0.2</td>
<td>2</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

C Multiplicity of CLUEs

We exploit the non-convexity of CLUE’s objective to generate diverse CLUEs. We initialize CLUE with $x_0 = \mu(x) + \epsilon$, where $\epsilon \sim \mathcal{N}(z; 0, \sigma_0 I)$, and perform Algorithm 1 multiple times to obtain different CLUEs. We choose $\sigma_0 = 0.15$. In Figure 12, we showcase different CLUEs for the same original MNIST inputs. Different counterfactuals represent digits of different classes. Despite this, all explanations resemble the original datapoints being explained. Being exposed to this multiplicity could potentially inform practitioners about similarities of an original input to multiple classes that lead their model to be uncertain.
Different initializations lead to CLUEs that explain away different amounts of uncertainty. In a few rare cases CLUE fails: the algorithm does not produce a feature configuration which has significantly lower uncertainty than the original input. This is the case for the third CLUE in the bottom 2 rows of Figure 12. We attribute this to a disadvantageous initialization of $z$.

Figure 12: We generate 5 possible CLUEs for 11 MNIST digits score above the uncertainty rejection threshold. Below each digit or counterfactual is the predictive entropy it is assigned $H$ and the class of maximum probability $c$. 

<table>
<thead>
<tr>
<th>Original</th>
<th>CLUE 0</th>
<th>CLUE 1</th>
<th>CLUE 2</th>
<th>CLUE 3</th>
<th>CLUE 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H=1.75$, $c=8$</td>
<td>$H=0.30$, $c=8$</td>
<td>$H=0.32$, $c=8$</td>
<td>$H=0.51$, $c=8$</td>
<td>$H=0.19$, $c=8$</td>
<td>$H=0.26$, $c=8$</td>
</tr>
<tr>
<td>$H=1.70$, $c=1$</td>
<td>$H=0.04$, $c=1$</td>
<td>$H=0.08$, $c=1$</td>
<td>$H=0.03$, $c=1$</td>
<td>$H=0.07$, $c=1$</td>
<td>$H=0.07$, $c=4$</td>
</tr>
<tr>
<td>$H=1.60$, $c=9$</td>
<td>$H=0.43$, $c=0$</td>
<td>$H=0.07$, $c=0$</td>
<td>$H=0.21$, $c=0$</td>
<td>$H=0.94$, $c=3$</td>
<td>$H=0.68$, $c=3$</td>
</tr>
<tr>
<td>$H=1.56$, $c=3$</td>
<td>$H=0.35$, $c=3$</td>
<td>$H=0.12$, $c=3$</td>
<td>$H=0.86$, $c=8$</td>
<td>$H=0.14$, $c=0$</td>
<td>$H=0.31$, $c=8$</td>
</tr>
<tr>
<td>$H=1.55$, $c=2$</td>
<td>$H=0.06$, $c=7$</td>
<td>$H=0.08$, $c=7$</td>
<td>$H=0.05$, $c=7$</td>
<td>$H=0.14$, $c=2$</td>
<td>$H=0.21$, $c=2$</td>
</tr>
<tr>
<td>$H=1.52$, $c=0$</td>
<td>$H=0.05$, $c=0$</td>
<td>$H=0.14$, $c=0$</td>
<td>$H=0.03$, $c=0$</td>
<td>$H=0.07$, $c=0$</td>
<td>$H=0.60$, $c=9$</td>
</tr>
<tr>
<td>$H=1.47$, $c=3$</td>
<td>$H=1.30$, $c=0$</td>
<td>$H=0.11$, $c=3$</td>
<td>$H=0.04$, $c=0$</td>
<td>$H=0.01$, $c=0$</td>
<td>$H=0.12$, $c=3$</td>
</tr>
<tr>
<td>$H=1.44$, $c=2$</td>
<td>$H=0.16$, $c=0$</td>
<td>$H=0.13$, $c=2$</td>
<td>$H=0.10$, $c=0$</td>
<td>$H=0.15$, $c=2$</td>
<td>$H=0.09$, $c=2$</td>
</tr>
<tr>
<td>$H=1.43$, $c=6$</td>
<td>$H=0.42$, $c=6$</td>
<td>$H=0.06$, $c=0$</td>
<td>$H=0.30$, $c=6$</td>
<td>$H=0.05$, $c=0$</td>
<td>$H=0.38$, $c=6$</td>
</tr>
<tr>
<td>$H=1.15$, $c=8$</td>
<td>$H=0.76$, $c=8$</td>
<td>$H=0.17$, $c=3$</td>
<td>$H=0.73$, $c=8$</td>
<td>$H=0.97$, $c=8$</td>
<td>$H=0.11$, $c=3$</td>
</tr>
<tr>
<td>$H=1.38$, $c=6$</td>
<td>$H=0.12$, $c=6$</td>
<td>$H=1.08$, $c=6$</td>
<td>$H=0.60$, $c=6$</td>
<td>$H=1.45$, $c=8$</td>
<td>$H=0.21$, $c=6$</td>
</tr>
</tbody>
</table>
In Figure 13 we show multiple CLUEs for a single individual from the COMPAS dataset. In this case, uncertainty can be reduced by changing the individual’s prior convictions and charge degree, or by changing their sex and age range. Making both sets of changes simultaneously also reduces uncertainty.

Figure 13: The leftmost entry is an uncertain COMPAS test sample. To its right are four candidate CLUEs. The first three successfully reduce uncertainty past our rejection threshold, while the rightmost does not.

D SENSITIVITY ANALYSIS IN HIGH DIMENSIONAL SPACES

In high-dimensional input spaces, $\nabla_x H$ will often not point in the direction of the data manifold. This can result in meaningless explanations. In Figure 14 we show an example where a step in the direction of $-\nabla_x H$ leads to a seemingly noisy input configuration for which the predictive entropy is low. An “adversarial examples for uncertainty” is generated. Aggregating these steps for every point in the test set leads to an uncertainty sensitivity analysis explanation that resembles white noise.

Figure 14: Left: A digit from the MNIST test set with large predictive entropy. Center: The same digit after a step is taken in the direction of $-\nabla_x H$. Non-zero weight is assigned to pixels that are always zero valued. Right: Uncertainty sensitivity analysis for the entire MNIST test set.

E VISUALIZING OPTIMIZATION IN LATENT SPACE

Figure 15 shows a 2 dimensional latent space trajectory from $\mathbf{z}_0$ to $\mathbf{z}_{\text{CLUE}}$ for a test point from the COMPAS dataset. In practice, we use larger latent spaces to ensure CLUEs are relevant.

Figure 15: Left: CLUE latent trajectory for a test point from the Credit dataset in a two-dimensional latent space. The blue dot marks the start of the trajectory and the orange one marks the end. Uncertainty levels are displayed in greyscale. Right: Changes in aleatoric entropy for inputs regenerated from latent codes along the trajectory.
F Comparing CLUE to Feature Importance Estimators

Among machine learning practitioners, two of the most popular approaches for determining feature importance from back-box models are LIME and SHAP [Bhatt et al. (2020)]. LIME locally approximates the back-box model of interest around a specific test point with a surrogate linear model [Ribeiro et al., 2016]. This surrogate is trained on points sampled from nearby the input of interest. The surrogate model’s weights for each class can be interpreted as each feature’s contribution towards the prediction of said class. Kernel SHAP extends lime by introducing a kernel such that resulting explanations have desirable properties [Lundberg & Lee (2017)]. For SHAP, a reference input is chosen. It allows importance to be only assigned where the inputs are different from the reference. For MNIST, the reference is an entirely black image. Note that alternative versions of SHAP exist that incorporate information about internal NN dynamics into their explanations. However, they produce very noisy explanations when applied to our BNNs. We conjecture that this high variance might be induced by disagreement among the multiple weight configurations from our BNNs.

Figure 16: High confidence MNIST test examples together with LIME and SHAP explanations for the top 3 predicted classes. The model being investigated is a BNN with architecture described in Appendix B. The highest probability class is denoted by $\hat{y}$. 
Figure [16] shows examples of LIME and Kernel SHAP being applied to a BNN for high confidence MNIST test digits. We use the default LIME hyperparameters for MNIST: the “quickshift” segmentation algorithm with kernel size 1, maximum distance 5 and a ratio of 0.2. We plot the top 10 segments with weight greater than 0.01. We draw 1000 samples with both methods.

Using the same configuration, we generate LIME and SHAP explanations for some MNIST digits to which our BNN assigns predictive entropy above our rejection threshold. The results are displayed in Figure [17].

Figure 17: Ten MNIST test digits for which our BNN’s predictive entropy is above the rejection threshold. A single CLUE example is provided for each one. For each digit, the top scoring class is denoted by \( \hat{y} \). LIME and SHAP explanations are provided for the three most likely classes.

A positive CLUE attribution means that the addition of that feature will make our model more certain. A positive feature importance attribution means the presence of that feature serves as evidence towards a predicted class. A negative CLUE attribution means that the absence of that feature will make the model more certain. A negative feature importance attribution means the absence of that feature would serve as evidence for a particular prediction. While CLUE and feature importance techniques solve similar problems and both provide saliency maps, CLUE highlights regions that need to be added or removed to make the input certain to a predictive model. In some cases, we see that feature importance negative attribution aligns with CLUE negative attribution, suggesting the
features which negatively contribute to the model’s predicted probability are the features that need to be removed to increase the models’ certainty. CLUE’s ability to suggest the addition of unobserved features (positive CLUE attribution) is unique.

The feature importance methods under consideration are difficult to retrofit for uncertainty. They are unable to add features; they are limited to explaining the contribution of existing features. This may suffice if our input contains all the information needed to make a prediction for a certain class but otherwise results in noisy, potentially meaningless, explanations.

Generative-model based methods methods are counterfactual because they do not assign importance to the observed features but rather propose alternative features based on the data manifold Chang et al. (2019). This is the case for FIDO and CLUE. Generative modeling allows for increased flexibility, which is required when dealing with uncertain inputs. Quantitatively contrasting feature importance and uncertainty explanations under existing evaluation criteria Bhatt et al. (2020a) is an interesting direction for future work.

Methods like LIME and SHAP require a choice of class to produce explanations. This complicates their use in scenarios where our model is uncertain and multiple classes have similarly high predictive probability. On the other hand CLUEs are class agnostic.

G ADDITIONAL CLUE AND U-FIDO EXAMPLES

We provide additional examples of CLUEs generated for high uncertainty MNIST digits in Figure 18. U-FIDO counterfactuals generated for the same inputs are shown in Figure 19. Both methods often attribute importance to the same features. However, in almost all cases, CLUE is able to reduce the original input’s uncertainty significantly more than U-FIDO. The latter method suggests smaller changes. We attribute this to U-FIDO’s input masking mechanism being less flexible than CLUE’s latent space generation mechanism.

Figure 18: CLUEs generated for MNIST digits for which our BNN’s predictive entropy is above the rejection threshold. The BNNs predictive entropy for both original inputs and CLUEs is shown under the corresponding images.
Figure 19: U-FIDO counterfactuals generated for MNIST digits for which our BNN's predictive entropy is above the rejection threshold. The BNNs predictive entropy for both original inputs and counterfactuals is shown under the corresponding images.
H ADDITIONAL EXPERIMENTAL RESULTS

H.1 ABLATION EXPERIMENTS

In this subsection, we modify some of CLUE’s components individually and observe the effects on the procedure’s results.

Initialization Strategy: Figure 20 compares Algorithm 1’s encoder-based initialization $z_0 = \mu_\phi(z|x_0)$ with $z_0 = 0$ on all datasets under consideration. For the LSAT, COMPAS and Wine datasets, both approaches produce indistinguishable results. On Credit, our second highest dimensional dataset, using an encoder-based initialization allows for CLUEs to stay slightly closer to original inputs in terms of $\ell_1$ distance.

The difference between both approaches is largest on MNIST. We conjecture that this might be due to the higher dimensional nature of the latent space used with this dataset making optimization more difficult. By initializing $z$ as the VAE encoder’s mean, our optimizer starts near a local minima of $d(x,x_0)$ and potentially of $L(z)$. When Algorithm 1 is applied, the magnitude of $\nabla_x H$ might not be large enough to escape this basin of attraction. Thus, CLUE tends to leave most input features unchanged, only addressing those with most potential to reduce uncertainty. This is also desirable behavior for low uncertainty inputs; the closest low uncertainty sample is the input itself.

Figure 20: Initialization strategy experiment results for all datasets under consideration. Colorbars’ horizontal blue line denotes each dataset’s rejection threshold.
Capacity of CLUE’s DGM: To capture our predictive model’s reasoning, CLUE’s DGM must be flexible enough to preserve atypical features in the inputs. As shown in Figure 21, reconstructions from low-capacity VAEs do not preserve the predictive uncertainty of original inputs. The CLUEs generated from these DGMs either leave the inputs unchanged or present large values of $\Delta x$ while barely reducing $H$: these degenerate CLUEs simply emphasize regions of large reconstruction error. As our DGM’s capacity increases, so does the amount of uncertainty preserved in the auto-encoding operation. The amount of predictive uncertainty explained by CLUEs, which is given by the difference between the autoencoded input uncertainty (orange bars) and CLUE uncertainty (blue bars), increases. We see a clear relationship between dataset dimensionality and size of latent space needed for CLUE to be effective.

Figure 21: Amount of uncertainty explained away and $\ell_1$ distance between original inputs and CLUEs for every dataset under consideration and different capacity VAEs.
**Output Space Regularization Parameter** $\lambda_y$: In Figure 22, we show how increasing $\lambda_y$ reduces the proportion of samples for which the predicted class differs between original inputs and CLUEs. Interestingly, on LSAT, Wine and COMPAS, a small, but non-zero, value of $\lambda_y$ results in more uncertainty being explained away by CLUE. However, strongly enforcing similarity of predictions generally comes at the cost of smaller amounts of uncertainty being explained away.

COMPAS predictions stay the same for all values of $\lambda_y$. Class predictions only depend on 2 of this dataset’s input features (Age and Previous Convictions) (Dressel & Farid, 2018). We find that the remaining features can increase or reduce confidence in the prediction given by the two key features, but never change it. CLUEs only change non key features, reinforcing the current classification.

On MNIST, we find that, for certain values of $\lambda_y$, classifying CLUEs results in a lower error rate than classifying original inputs. This is shown in Figure 23. We did not observe this effect for other datasets.

![Figure 22: CLUE $\Delta H$ vs prediction change for all datasets under consideration. Prediction change refers to the proportion of CLUEs classified differently than their corresponding original inputs. All values shown are averages across all testset points above the uncertainty rejection threshold.](image)

**Applying CLUE to non-Bayesian NNs**: These models are unable to capture model uncertainty. We train deterministic NNs on every dataset under consideration using the architectures described in Appendix B.4. We generate counterfactuals for their noise uncertainty. As shown in Figure 24 CLUE is effective at explaining away noise uncertainty for regular NNs. More uncertain inputs are subject to larger changes in terms of $\ell_1$ distance.
Figure 23: Left: Prediction change refers to the proportion of CLUEs classified differently than their corresponding original inputs. Setting a value of $\lambda_y$ of around 0.7 results in class predictions for CLUEs being closer to the true labels than the original class predictions. Right: Reduction in predictive entropy achieved by CLUE. All values shown are averages across all testset points above the uncertainty rejection threshold.

Figure 24: Amount of noise uncertainty explained away vs $\ell_1$ shift in input space for all datasets under consideration when applying CLUE to regular NNs. The colorbar indicates the original samples’ predictive uncertainty.
H.2 Verifying Results from our Computational Evaluation Framework

Our computational evaluation framework relies on generating artificial data. There is reasonable concern that the characteristics of this data may not reflect that of real-world data, biasing our results. As explained in Appendix I, we are careful to use powerful G.T. DGM models that generate high quality artificial data. Be that as it may, we validate the results from our computational evaluation framework by performing an analogous informativeness vs relevance experiment on real data.

As we do not have access to the generative process of the real data, we cannot exactly quantify the uncertainty of inputs or how in-distribution they are. We instead resort to quantifying informativeness as the amount of our BNN’s predictive uncertainty explained away $\Delta H = H(y|x_0) - H(y|x_c)$. We measure relevance as the $L_2$ distance of each counterfactual to its $L_2$ nearest neighbor within the train-set $d_{NN-2}(x_c, D)$.

We report mean values of $\Delta H$, $d_{NN-2}(x_c, D)$ and $\frac{\Delta H}{d_{NN-2}(x_c, D)}$ across all uncertain test points. A test point is deemed to be uncertain according to the criteria outlined in Appendix B.4. The hyperparameters employed for CLUE also match those provided in Appendix B.4. The step size used with local sensitivity analysis $\eta$ and U-FIDO’s $\lambda_0$ parameter are found via grid search with a methodology analogous to the one described in Section 5.4.

Table 6: Quantities of informativeness ($\Delta H$, higher is better), relevance ($d_{NN-2}(x_c, D)$, lower is better) and their ratio ($\frac{\Delta H}{d_{NN-2}(x_c, D)}$, higher is better) obtained on real data from the LSAT, COMPAS and Wine datasets. The numbers in parenthesis indicate dataset dimensionality.

<table>
<thead>
<tr>
<th>Method</th>
<th>LSAT (4) $d_{NN-2}(x_c, x_0)$</th>
<th>$\Delta H$</th>
<th>$\Delta H / d_{NN-2}$</th>
<th>COMPAS (7) $d_{NN-2}(x_c, x_0)$</th>
<th>$\Delta H$</th>
<th>$\Delta H / d_{NN-2}$</th>
<th>Wine (11) $d_{NN-2}(x_c, x_0)$</th>
<th>$\Delta H$</th>
<th>$\Delta H / d_{NN-2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitivity</td>
<td>0.482</td>
<td>0.003</td>
<td>0.0192</td>
<td>7.975</td>
<td>0.265</td>
<td>0.033</td>
<td>3.317</td>
<td>0.481</td>
<td>0.154</td>
</tr>
<tr>
<td>CLUE</td>
<td>0.080</td>
<td>0.092</td>
<td>1.664</td>
<td>0.067</td>
<td>0.014</td>
<td>0.737</td>
<td>1.274</td>
<td>1.409</td>
<td>1.188</td>
</tr>
<tr>
<td>U-FIDO</td>
<td>0.085</td>
<td>0.077</td>
<td>0.969</td>
<td>0.084</td>
<td>0.022</td>
<td>0.627</td>
<td>1.223</td>
<td>1.307</td>
<td>1.241</td>
</tr>
</tbody>
</table>

Table 7: Quantities of informativeness ($\Delta H$, higher is better), relevance ($d_{NN-2}(x_c, D)$, lower is better) and their ratio ($\frac{\Delta H}{d_{NN-2}(x_c, D)}$, higher is better) obtained on real data from the Credit and MNIST datasets. The numbers in parenthesis indicate dataset dimensionality.

<table>
<thead>
<tr>
<th>Method</th>
<th>Credit (23) $d_{NN-2}(x_c, x_0)$</th>
<th>$\Delta H$</th>
<th>$\Delta H / d_{NN-2}$</th>
<th>MNIST (784) $d_{NN-2}(x_c, x_0)$</th>
<th>$\Delta H$</th>
<th>$\Delta H / d_{NN-2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitivity</td>
<td>0.770</td>
<td>0.224</td>
<td>0.121</td>
<td>6.903</td>
<td>0.601</td>
<td>0.087</td>
</tr>
<tr>
<td>CLUE</td>
<td>1.025</td>
<td>0.147</td>
<td>0.293</td>
<td>4.374</td>
<td>0.628</td>
<td>0.153</td>
</tr>
<tr>
<td>U-FIDO</td>
<td>1.863</td>
<td>0.017</td>
<td>0.052</td>
<td>4.887</td>
<td>0.409</td>
<td>0.088</td>
</tr>
</tbody>
</table>

As shown in Table 6 and Table 7, CLUE outperforms U-FIDO in terms of $d_{NN-2}$ on all datasets except wine, where both approaches are very similar. The same is true for the ratio $\frac{\Delta H}{d_{NN-2}}$. Like in the artificial data experiments from Section 5.1, the difference between both methods is most stark for high dimensional datasets (MNIST and Credit). Here, CLUE is able to explain away more uncertainty while providing counterfactuals that are similarly close to the training data. Sensitivity is able to greatly reduce uncertainty in high dimensions. However, this comes at the cost going off the data manifold. In low dimensions, there are less possible directions in which steps can be taken, rendering the direct gradient based approach less powerful.

We note that the similarity of these results to the ones obtained in the analogous experiments from Section 5.1 suggest the unbiasedness of our computational evaluation framework.

H.3 Additional Analysis of User Study

While the main text showed the mean accuracy of CLUE over all tabular questions, we also consider the breakdown of accuracy by dataset and by test point certainty in Table 8. CLUE outperforms all baselines on both datasets. We find that sensitivity does significantly worse in higher dimensions (on COMPAS), lending further credence to the intuition described in Appendix D.
When splitting by the certainty of test points, we immediately notice that accuracy for uncertain test points is quite high for all methods. This similarity is expected since certain context points are the only factor that varies between each method’s survey. Survey participants seemed to not use the certain context points to identify uncertain test points. This is probably due to pilot procedure, wherein Participant A carefully paired test points with relevant uncertain context points. Indeed, the random baseline, which controls for the possibility that our task can be solved without access to a relevant counterfactual, performs best on uncertain test points. However, we note a large difference between methods’ results when identifying certain test points. CLUE’s accuracy almost doubles the second best method’s (Human CLUE). When generating Human CLUEs Participant B had knowledge of the uncertain context point, but not the test point (just like other methods). For this reason, we expect to see dissimilarity in methods’ performance on certain test points. CLUE’s ability to bring about most relevant contrast is one possible explanation for why it does so much better than baselines for certain context points.

Table 8: Accuracy (%) of participants on the Tabular main survey broken down by dataset and by certainty of test points.

<table>
<thead>
<tr>
<th></th>
<th>Combined</th>
<th>LSAT</th>
<th>COMPAS</th>
<th>Certain Test</th>
<th>Uncertain Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLUE</td>
<td>82.22</td>
<td>83.33</td>
<td>81.11</td>
<td>71.00</td>
<td>96.25</td>
</tr>
<tr>
<td>Human CLUE</td>
<td>62.22</td>
<td>61.11</td>
<td>63.33</td>
<td>38.00</td>
<td>92.50</td>
</tr>
<tr>
<td>Random</td>
<td>61.67</td>
<td>62.22</td>
<td>61.11</td>
<td>31.00</td>
<td>100</td>
</tr>
<tr>
<td>Local Sensitivity</td>
<td>52.78</td>
<td>56.67</td>
<td>48.89</td>
<td>20.90</td>
<td>92.50</td>
</tr>
</tbody>
</table>

I ADDITIONAL DETAILS ON THE GENERATIVE MODEL USED IN THE PROPOSED COMPUTATIONALLY GROUNDED EVALUATION FRAMEWORK

The framework described in Figure 6 uses a conditional DGM, specifically a VAEAC (Ivanov et al., 2019), to both generate artificial data and to evaluate explanations for said data. VAEs are known for generating blurry or overly smoothed data. For our evaluation framework to work well, we require the ground truth DGM to generate sharp data, with atypical characteristic that would lead to a predictor being uncertain. We can ensure that this is the case by using a large latent dimensionality. However, this brings forth another well-known issue with VAEs: distribution mismatch (Dai & Wipf, 2019; Antoran & Miguel, 2019; Rosca et al., 2018). The region of latent space where the encoder places probability mass, also known as the aggregate posterior,

\[ q_\phi(z) = \int q_\phi(z|x)p(x)\,dx \]

does not match the prior \( p(z) \).

Figure 25: Predictive entropy estimates for artificial MNIST digits generated from a 2-dimensional VAE latent space. The MNIST test set digits have been projected onto the latent space and are displayed with a different color per class.
To visualize this phenomenon, we train a BNN and a VAE on MNIST. We sample points from the VAE’s latent space and evaluate their uncertainty with the BNN. As shown in Figure 25, clusters of same-class digits form in latent space. The aggregate posterior presents low density in the spaces between clusters. Digits generated from these areas are of low-quality, causing our BNN to be uncertain. The outer regions of latent space, where the isotropic Gaussian prior has low density, also generate uncertain digits.

Recently, Dai & Wipf (2019) have proposed the two-level VAE as a solution to distribution mismatch. After training a standard VAE, a second VAE is trained on samples from the first VAE’s latent space. As illustrated in Figure 26, the aggregate posterior over the inner latent variables, which we denote by \( q(u) \), more closely resembles the prior. The joint distribution over inputs and latent variables factorizes as:

\[
p(x, z, u) = p(x|z)p(z|u)p(u).
\]

We refer the reader to Dai & Wipf (2019) for a detailed analysis. Figure 27 shows that, while generating digits from samples of \( p(z) \) results in a large amount of low-quality or OOD reconstructions, samples from \( p(u) \) map to clean digits. The two-stage mechanism restores the VAE’s pivotal ancestral sampling capability, ensuring that our experiments with artificial data will be representative of methods performance on real data.

Figure 26: In its first stage, the two-level VAE maps input samples to approximate posteriors in the outer latent space. The aggregate posterior over this latent space need not resemble the isotropic Gaussian prior. The second VAE maps samples from the outer latent space to approximate posteriors in the inner latent space. The aggregate posterior over the inner latent space more closely matches the prior.

Figure 27: Left: Digits generated from the inner latent space of a VAEAC trained on MNIST with a two-level mechanism. Right: Digits generated from the latent space of a VAEAC trained on MNIST. \( u \) and \( z \) are drawn from \( \mathcal{N}(0, I) \).

In order to generate artificial data, we draw samples from the auxiliary latent space, map them back to the VAEAC’s latent space and then map them to the input space. This allows for high-quality sample generation. In this way, a single VAEAC can be used for both ancestral sampling and conditional sampling. In addition, it allows us to estimate the log-likelihood of inputs as:

\[
\log p_{gt}(x) = \log \int p_{\theta_1}(x|z)p_{\theta_2}(z|u)p(u) \, dz \, du
\]

In (8) parameter subscripts refer to the outer (1st level) and inner (2nd level) networks. In order to preserve computational tractability, we approximate \( p_{\theta_2}(z|u) \) with a point estimate placed at its mean.
\[ p_{\theta_2}(z|u) \approx \delta(z - \mu_{\theta_2}(z|u)). \]

We further approximate (8) with importance sampling:

\[
\log p_{\text{gt}}(x) \approx \log \frac{1}{K} \sum_{k=1}^{K} p_{\theta_1}(x|z=\mu_{\theta_2}(z|u_k)) p(u_k) \quad u_k \sim q(u|x) \tag{9}
\]

### I.1 Comparison of Methods Under a Ground Truth DGM

The two-level VAEC setup described above partially addresses the concern that our synthetic data might not be diverse enough to highlight differences among the methods being compared. Indeed, our results from Table 1 and Table 2 show noticeable differences in performance across methods.

We now address the opposite concern; methods that leverage auxiliary VAEs might be unfairly advantaged under our functionally grounded framework, as the generative process of our synthetic data is also VAE-based. Because VAEs are very flexible neural network based generative models, using them as a ground truth provides relatively little inductive biases for auxiliary DGMs to take advantage of. Additionally, our ground truth VAEC captures the joint distribution of inputs and targets. The metric of interest, \( \Delta H_{\text{gt}} \), only depends on the conditional distribution over targets \( p_{\text{gt}}(y|x) \). Our auxiliary DGMs only model inputs.

Two of the methods we evaluate, U-FIDO and CLUE, leverage auxiliary DGMs. Thus, both would be equally advantaged. The \( \Delta H_{\text{gt}} \) vs \( \Delta \log p_{\text{gt}} \) metric from Table 2 is the most dependent on the ground truth VAEC. However, we observe the largest difference between CLUE and U-FIDO on in this metric.

### J Details on User Study

#### J.1 Additional Details on Tabular User Study

For our pilot point selection procedure, we take points from each dataset’s test set that score above the uncertainty rejection thresholds described in Appendix B.4 as uncertain points. Points below the thresholds are labeled as certain points. Pilot procedure participants, referred to as participant A in the main text, were not informed that the pools were split up by the points’ certainty with respect to the BNN being explained.
were asked to take the survey did not provide consent and thus exited the form. We still ensured that at least ten participants took each of the four survey variants.

We then include an example question for each dataset, called an “attention check.” An example is shown in Figure 28b. Note that the answer to this example question is provided in line. Later in the survey, we ask participants this exact same question. We ask one attention check per dataset. If participants get the attention check wrong for both datasets, we void their results. We only had to void one result. This did not affect our criteria of ten completed surveys per variant. The consent form and attention check questions were the same for all survey variants. The main survey participants were first asked the ten LSAT questions followed by the ten COMPAS questions: we made this design decision since the dimensionality of LSAT is lower than that of COMPAS, easing participants into the task. Examples of questions from the CLUE survey variant are shown in Figure 29.

J.2 MNIST USER STUDY

In order to validate CLUE on image data, we create a modified MNIST dataset with clear failure modes for practitioners to identify. We first discard all classes except four, seven, and nine. We then manually identify forty sevens from the training set which have dashes crossing their stems. Using K-nearest-neighbors, we identify the twelve sevens closest to each of the ones manually selected. We delete these 520 sevens from our dataset. We repeat the same procedure for fours which have a closed, triangle-shaped top. We do not delete any digits from the test set. We train a BNN on this new dataset. Our BNN presents high epistemic uncertainty when tested on dashed sevens and closed fours as a consequence of the sparsity of these features in the train set.

We evaluate the test set of fours, sevens, and nines with our BNN. Datapoints that surpass our uncertainty threshold are selected as candidates to be shown in our user study as uncertain context examples or test questions. We show example CLUEs for a four and a seven that display the characteristics of interest in Figure 30.
Leveraging the modified MNIST dataset, we run another user study with 10 questions and two variants. Unlike our tabular experiments, we show practitioners a set of five context points to start, as opposed to a pair. This set of context points is chosen at random from the training set. The first variant involves showing users the set of context points, labeled with if their uncertainty surpasses our predefined threshold. We then ask users to predict if new test points will be certain or uncertain to the BNN. The second variant contains the same labeled context points and test datapoints. However, together with uncertain context points, practitioners are shown CLUEs of how the input features can be changed such that the BNN’s uncertainty falls below the rejection threshold. The practitioners are then asked to decide if new points’ predictions will be certain or not. If CLUE works as intended, practitioners taking the second variant should be able to identify points on which the BNN will be uncertain more accurately.

The first variant was shown to 5 graduate students with machine learning expertise who only received context points and rejection labels (uncertain or not). This group was able to correctly classify 67% of the new test points as high or low uncertainty. The second variant was shown to 5 other graduate students with machine learning expertise who received context points together with CLUEs in cases of high uncertainty. This group was able to reach an accuracy of 88% on new test points. This user study suggests CLUEs are useful for practitioners in image-based settings as well.