A Implementation Details

Here we provide implementation details of RoCourseNet and three baseline methods on three datasets listed in Section 4. The code can be found through this anonymous repository (https://github.com/bkghz-orange-blue/counternet_adv).

Feature Engineering. We follow the feature engineering procedure of CounterNet (Guo et al., 2021). Specifically, for continuous features, we scale all feature values into the $[0, 1]$ range. To handle the categorical features, we customize model architecture for each dataset. First, we transform the categorical features into numerical representations via one-hot encoding. In addition, for each categorical feature, we add a softmax layer after the final output layer in the CF generator, which ensures that the generated CF examples respect the one-hot encoding format.

Hyperparameters. For all three datasets, we train the model for up to 50 epochs with Adam. We set dropout rate to 0.3 to prevent overfitting. We use $T = 7$ and $E = 0.1$ to report results in Table 1, and report the impact of attacker steps T and maximum perturbation E to robustness in Figure 3. We use $K = 2$ unrolling steps (same as Huang et al. (2020)) with the step size $\alpha = 2.5 \times \delta/T$ (based on Madry et al. (2017)) for solving the bi-level problem in Equation 3 (via VDS). In addition, Table 2 reports the hyperparameters chosen for each dataset, and Table 3 speficies the architecture used for each dataset.

Table 2: Hyperparameters setting for each dataset.

Dataset	Learning Rate	η	Batch Size	λ_1	λ_2	λ_3
Loan	0.003	0.03	128	1.0	0.2	0.1
German Credit	0.003	0.03	256	1.0	1.0	0.1
Student	0.01	0.01	128	1.0	0.2	0.1

Table 3: Architecture specification of RoCourseNet for each dataset.

Dataset	Encoder Dims	Predictor Dims	CF Generator Dims
Loan	[110,200,10]	[10, 10]	[10, 10]
German Credit	[19, 100,10]	[10, 20]	[10, 20]
Student	[83,50,10]	[10, 10]	[10, 50]

Software and Hardware Specifications. We use Python (v3.7) with Pytorch (v1.82), Pytorch Lightning (v1.10), numpy (v1.19.3), pandas (v1.1.1), scikit-learn (v0.23.2) and higher (v0.2.1) Grefenstette et al. (2019) for the implementations. All our experiments were run on a Debian-10 Linux-based Deep Learning Image on the Google Cloud Platform. The RoCourseNet and baseline methods are trained (or optimized) on a 16-core Intel machine with 64 GB of RAM.

B Additional Experimental Analysis

B.1 Predictive Accuracy

We first show that, similar to CounterNet, the training of RoCourseNet does not come at the cost of degraded predictive accuracy. Table 4 compares RoCourseNet's predictive accuracy against the base prediction model used by baselines. This table shows that RoCourseNet achieves competitive predictive performance – it achieves marginally better accuracy than the base model ($\sim 2\%$). Thus, we conclude that the joint training of RoCourseNet does not come at a cost of reduced predictive performance.

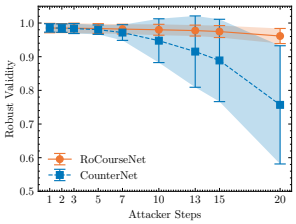
B.2 l_2-norm Projection in Algorithm 1

We provide supplementary results on adopting Δ as the l_2-norm ball (i.e., $\Delta = \{\delta \in \mathbb{R}^n \mid ||\delta||_2 \leq \epsilon\}$) for the maximum perturbation constrains. Figure 5 highlights the results of using the l_2-norm ball in

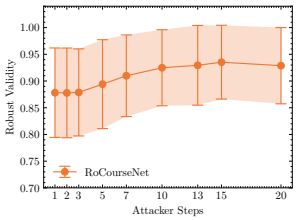
Table 4: Predictive accuracy for each dataset.

Dataset	Base Model	RoCourseNet
Loan	0.886 ± 0.036	0.885 ± 0.035
German Credit	0.714 ± 0.003	0.742 ± 0.014
Student	0.914 ± 0.028	0.906 ± 0.066

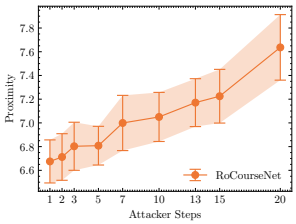
attacking and adversarial training. We observe similar patterns in Figure 3. Thus, this result shows that l_∞-norm constrain can be substitute to other feasible region.



(a) The number of attacker steps T vs attacker effectiveness (↓)



(b) The number of attacker steps T vs robust validity (↑).



(c) The number of attacker steps T vs proximity (↓).

Figure 5: The impact of the number of attacker steps T under the l_2-norm constrains.

C Discussion about Generalized Frameworks to RoCourseNet

This section discusses extending the training of RoCourseNet (see Algorithm 1 & 2) into a broader framework. The training framework of RoCourseNet are not tied to the CounterNet architecture, and we can substitute predictor and CF generator components to other parametric models. Crucially, we can extend the RoCourseNet training into a general framework, which contains two components: (i) a predictor $f(\cdot;\theta)$, which makes an accurate prediction for a given input instance x, (ii) and a CF generator $g(\cdot;\theta_g)$, which generates its corresponding CF explanations. This general framework does not require a shared structure similar to the RoCourseNet architecture. We can choose separate models for the predictor $f(\cdot;\theta)$ and CF generator $g(\cdot;\theta_g)$. In addition, we can train this framework exactly as we train RoCourseNet via Algorithm 2, which optimizes for predictor $f(\cdot;\theta)$ and robust CF generator $g(\cdot;\theta_g)$.

Experimental Settings. We demonstrate the feasibility of training this general framework (denoted as *Robust CF Framework* in Table 5). For fair comparison, we use the same hyperparameters in training RoCourseNet (See Appendix A) to optimize this general framework. In addition, we use the same architecture specifications of RoCourseNet, as the predictor of this framework combines the encoder network and predictor network in RoCourseNet, and the CF generator network combines the encoder network and CF generator in RoCourseNet.

Empirical Results. Table 5 compares this general robust CF framework with RoCourseNet and Roar-LIME. This table highlights two important findings: (i) First, our proposed tri-level robust training (in Algorithm 2) can be extended to a general framework to optimize a robust CF generator. In particular, this robust CF framework outperforms Roar-LIME in terms of proximity and validity, and achieves the same level of robust validity as Roar-LIME. (ii) Additionally, we highlight the importance of RoCourseNet design (by leveraging the design of CounterNet (Guo et al., 2021)). This architecture design enables to generate well-aligned CF explanations by passing the predictor model's decision boundary (i.e., p_x) to the CF generator. From Table 5, we observe that RoCourseNet outperforms this Robust CF Framework in terms of the validity and robust validity, which underscores the importance of RoCourseNet's architecture designs.

Table 5: Evaluating robustness under model shift using a general framework.

Methods	**Metrics**		
	Proximity	**Validity**	**Rob-Validity**
Robust CF Framework	7.150 ± 1.078	0.949 ± 0.022	0.906 ± 0.119
Roar-LIME	7.648 ± 2.248	0.937 ± 0.046	0.908 ± 0.107
RoCourseNet	7.183 ± 0.406	$\mathbf{0.994 \pm 0.002}$	$\mathbf{0.930 \pm 0.152}$

D Discussion about Multi-class Classification

Existing CF explanation literature focuses on evaluating methods under the binary classification settings (Mothilal et al., 2020; Mahajan et al., 2019; Upadhyay et al., 2021; Guo et al., 2021). However, these CF explanation methods can be adapted to the multi-class classification settings. Given an input instance $x \in \mathbb{R}^d$, the RoCourseNet generates (i) a prediction $\hat{y}_x \in \mathbb{R}^k$ for input instance x, and (ii) a CF example x^{cf} as an explanation for input instance x. The prediction $\hat{y}_x \in \mathbb{R}^k$ is encoded as one-hot format as $\hat{y}_x \in \{0,1\}^k$, where $\sum_i^k \hat{y}_x^{(i)} = 1$, k denotes the number of classes. In addition, we assume a desired outcome y' for every input instances x. As such, we can adapt Eq. 4 for binary settings to the multi-class settings as follows:

$$
\begin{aligned}
&\underset{\theta,\theta_g}{\text{argmin}} \frac{1}{N} \sum\nolimits_{(x_i,y_i)\in\mathcal{D}} \Bigg[\lambda_1 \cdot \underbrace{\mathcal{L}\Big(f(x_i;\theta), y_i\Big)}_{\text{Prediction Loss } (L_1)} + \lambda_3 \cdot \underbrace{\mathcal{L}\Big(x_i, x_i^{\text{cf}}\Big)}_{\text{Proximity Loss } (L_3)} \Bigg] \\
&\quad + \max_{\boldsymbol{\delta}, \forall \delta_i \in \Delta} \frac{1}{N} \sum\nolimits_{(x_i,y_i)\in\mathcal{D}} \Bigg[\lambda_2 \cdot \underbrace{\mathcal{L}\Big(f\left(x_i^{\text{cf}}; \theta'_{opt}(\boldsymbol{\delta})\right), y'\Big)}_{\text{Robust Validity Loss } (L_2)} \Bigg] \\
&\quad s.t \;\; \theta'_{opt}(\boldsymbol{\delta}) = \text{argmin}_{\theta'} \frac{1}{N} \sum\nolimits_{(x_i,y_i)\in\mathcal{D}} \Bigg[\mathcal{L}\Big(f(x_i + \delta_i; \theta'), y_i\Big) \Bigg], \; x_i^{\text{cf}} = g(x_i; \theta_g).
\end{aligned}
\tag{5}
$$

To optimize for Eq. 5, we can follow the same procedure outlined in Algorithm 2. For each sampled batch, we first optimize for the predictive accuracy $\theta' = \theta - \nabla_\theta(\lambda_1 \cdot L_1)$. Next, we use the VDS algorithm to optimize for the inner max-min bi-level problem (in Algorithm 1). Finally, we optimize for the CF explanations by updating the model's weight as $\theta''_g = \theta''_g - \nabla_{\theta'_g}(\lambda_2 \cdot L_2 + \lambda_3 \cdot L_3)$.