Response to Reproducibility study of "LICO: Explainable Models with Language-Image Consistency"

Yiming Lei ymlei@fudan.edu.cn
Shanghai Key Laboratory of Intelligent Information Processing, School of Computer Science, Fudan University, Shanghai 200433, China

1 Introduction

We recently read the reproduction study of our paper "LICO: Explainable Models with Language-Image Consistency" published at TMLR. First of all, we appreciate your interest in our work and your contributions to the field of model explainability.

However, it appears that your team encountered challenges in reproducing our results, leading to contrasting conclusions. This includes significant **performance reduction** in both classification accuracy and explainability. Upon careful examination of your paper and code, we identified several impactful errors in implementation that may directly lead to **contradictory** results and minor errors that may have affected the accuracy. These primarily include coding errors and potential misunderstandings of our paper.

Next, in Section 2, we outline the errors found in the implementation, and in Section 3, we provide the results after corrections.

2 Errors in the Implementation

2.1 Implementation of Text Features

There are two main errors in `class WideResNetPrompt()` and `class PromptLearner()`.

- `class WideResNetPrompt()`. We found that the shape of *text_features* in `models/wideresnet_prompt.py` does not match ours, as they should align in the channel dimension to guide the image features effectively. For example, we found the shape of *feature_maps* and *text_features_w* when calculating *w_distance* is $[B, 128, 64]$ and $[B, 64]$ for CIFAR. In our implementation, the output shape of the *self.text_encoder()* is $[n\_cls \times n\_ctx, 512]$, then when calculating OT loss, the two terms should be with the shape of $[B, channel, 64]$ and $[B, n\_ctx, 64]$, *i.e.*, alignment along the $channel - n\_ctx$ dimension. This misalignment in feature dimensions led to incorrect results for the loss functions, negatively impacting the baseline model's performance.
- `class PromptLearner()`.The *ctx_vectors* is firstly initialized as $[1, n\_ctx, ctx\_dim]$, then we use `torch.roll` to implement different tokens, then for each sample, the $ctx\_vectors$ is with $[n\_ctx, n\_ctx, 512]$.

Correcting these errors significantly improved both the accuracy and explainability compared with the baseline models. We provide parts of code comparisons in Figs. 1 and 2. The detailed results can be found in Section 3.

2.2 Dataset Processing

First, we observe that the dataset used in the paper differs from ours. It is important to note that datasets such as CIFAR10 have a resolution of 32×32, which are rather low, raising concerns about its compatibility with the pre-trained CLIP model.

Second, there are **obvious** messed-up issues with class names due to the use of `sorted()` in lines 199 and 309 in `train.py`, which led to mismatched class names and labels of images, as shown

Correct Implementation

```python
# "forward" function in class WideResNetPrompt
def forward(self, targets=None, w_distance=None, mode='train'):
    ...

    # Line 139
    text_features_ot=text_features
    text_features=text_features[targets]

    if mode=='train':
        # (batch_size, prompt_length, ...). This is G_i.
        text_features_w=self.mlp(text_features_ot)
        text_features_w=text_features_w.view(text_features_w.shape
    [0],-1)
        text_features_w=text_features_w.contiguous().view(n_ctx, n_cls
    ,text_features_w.shape[1])
        # select text features using 'targets'
        text_features_w=text_features_w[:,targets,:].permute(1,0,2)

        # (batch_size, num_channels, ...)
        feature_maps=F.normalize(feature_maps, dim=-1)
        # (batch_size, num_context, ...)
        text_features_w=F.normalize(text_features_w, dim=-1)
        # Line139
        ...
```

Error Implementation

```python
# "forward" function in class WideResNetPrompt
def forward(self, targets=None, w_distance=None, mode='train'):
    ...

    # Line 131
     text_features = text_features[targets]  # (batch_size,
    prompt_length, 512)

    if mode == 'train':
        # prompt learning
        text_features_w = self.mlp(text_features)  # (batch_size,
    prompt_length, 64). This is G_i in the paper.
        feature_maps = F.normalize(feature_maps, dim=-1)  # (
    batch_size, num_channels, 64)
        text_features_w = F.normalize(text_features_w, dim=-1)  # (
    batch_size, num_context, 64)
    # Line 139
    ...
```

Figure 1: Some examples of the messed-up dataset due to `sorted()` function in dataset processing.

Correct Implementation

```python
# "__init__" function in class PromptLearner
def __init__(self, classnames, clip_model):
    ...

    # Line 61
    nn.init.normal_(ctx_vectors, std=0.02)
    prompt_prefix = " ".join(["X"] * n_ctx)
    # Rolling to simulate the different tokens, resulting in [1, n\
    _ctx, ctx\_dim] ctx_vectors
    ctx_vectors = torch.cat([torch.roll(ctx_vectors, shifts=1, dims=1)
     for i in range(n_ctx)], dim = 0)

    self.ctx = nn.Parameter(ctx_vectors)

    classnames = [name.replace("_", " ") for name in classnames]
    name_lens = [len(_tokenizer.encode(name)) for name in classnames]
    prompts = [prompt_prefix + " " + name + "." for name in classnames
    ]

    tokenized_prompts = torch.cat([clip.tokenize(p) for p in prompts])
    tokenized_prompts = tokenized_prompts.repeat(n_ctx, 1)

    with torch.no_grad():
        # [n_cls * n_ctx, 77, 512]
        embedding = clip_model.token_embedding(tokenized_prompts).type
    (dtype)

    self.register_buffer("token_prefix", embedding[:, :1, :])
    self.register_buffer("token_suffix", embedding[:, 1 + n_ctx :, :])

    # Line 97
        ...
```

Error Implementation

```python
# "__init__" function in class PromptLearner
def __init__(self, classnames, clip_model):
    ...

    # Line 61
    ctx_vectors = torch.empty(n_cls, n_ctx, ctx_dim, dtype=dtype)
    nn.init.normal_(ctx_vectors, std=0.02)  # define the prompt to be
    trained
    prompt_prefix = " ".join(["X"] * n_ctx)

    self.ctx = nn.Parameter(ctx_vectors)  # to be optimized

    classnames = [name.replace("_", " ") for name in classnames]
    name_lens = [len(_tokenizer.encode(name)) for name in classnames]
    prompts = [prompt_prefix + " " + name + "." for name in classnames
    ]

    tokenized_prompts = torch.cat([clip.tokenize(p) for p in prompts])
    tokenized_prompts = tokenized_prompts.repeat(self.N, 1)

    with torch.no_grad():
        embedding = clip_model.token_embedding(tokenized_prompts).type
    (dtype)

        self.register_buffer("token_prefix", embedding[:, :1, :])
        self.register_buffer("token_suffix", embedding[:, 1 + n_ctx:,
    :])
    # Line 97
        ...
```

Figure 2: Some examples of the messed-up dataset due to `sorted()` function in dataset processing.

in Fig. 3. For instance, in CIFAR10, the term 'plane' was used instead of 'airplane,' resulting in disordered labels after the sort function and potentially misleading the model training. The same mistakes are found in the implementation of the Imagnette dataset.

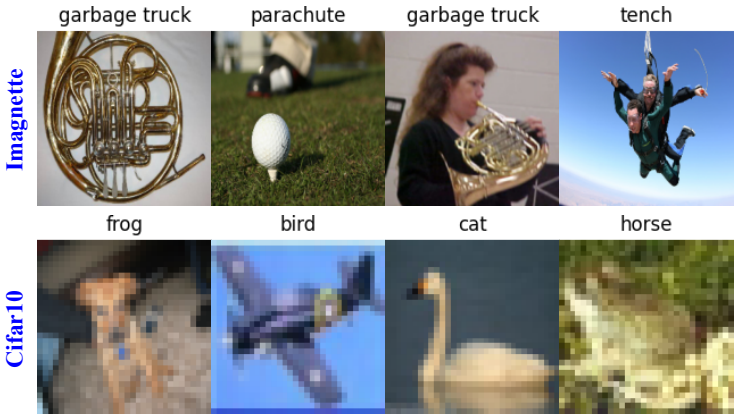

Figure 3: Some examples of the messed-up dataset due to `sorted()` function in dataset processing.

2.3 Other difference in implementation

Here, we list some other points that were NOT used in our implementation.

- Loss scalar. We did not use this scalar during training.

- torch.autocast. In the original implementation of `train.py`, the function `autocast()` with *float16* is utilized, likely due to substantial memory demands stemming from oversized feature maps that were generated by the erroneous implementation of `WideResNetPrompt()`. We observed that this mechanism could affect accuracy and is rarely employed in classification tasks with small backbones, such as ResNet.

Table 1: Performance of whether or not using Loss Scalar and Autocast.

	CIFAR10
w/ Scalar	93.67
w/ autocast	93.72
w/o Scalar, w/o autocast	93.79

3 Results

Based on the code at `https://github.com/robertdvdk/lico-fact` and the above corrections, we evaluated the classification accuracy and Insertion and Deletion scores on CIFAR10/100 and ImageNette. In Table 2, LICO surpasses the baseline WideResNet model. In Table 3, LICO outperforms the baseline model in terms of the most metrics. Only the results of GradCAM++ on CIFAR10 and GradCAM and GradCAM++ on CIFAR100 illustrate the slightly lower performance of LICO. Importantly, we would like to note that measuring interpretability metrics on low-resolution datasets such as CIFAR may be unreasonable since it is difficult to recognize or localize fine-grained objects or features. Note that the Gray values are reported from the paper "LICO: Explainable Models with Language- Image Consistency" published at TMLR. These values are presented for comparison and the **bold** numbers denote the best results obtained by our implementation.

Table 2: Classification accuracy.

	CIFAR10	CIFAR100	ImageNette
Baseline	93.79/93.9	72.90/78.1	87.16/87.3
LICO	**94.47**/93.6	**74.94**/77.3	**87.67**/86.8

Table 3: Insertion and deletion scores. Overall = Ins.−Del.

	CIFAR10			CIFAR100			ImageNette		
	Ins.↑	Del.↓	Overall↑	Ins.↑	Del.↓	Overall↑	Ins.↑	Del.↓	Overall↑
				`GradCAM`					
Baseline	64.34 63.0	**38.58** 37.3	25.76 25.8	**43.71** 44.0	**14.87** 19.2	**28.84** 26.2	**71.85** 77.1	32.29 23.0	39.29 54.1
LICO	**65.41** 63.6	38.93 35.4	**26.48** 28.2	42.20 43.4	16.61 19.8	25.59 23.6	70.20 74.5	**30.78** 25.5	**39.42** 49.0
				`GradCAM++`					
Baseline	55.30 53.8	**44.15** 43.8	**11.15** 9.9	**34.52** 34.1	**22.23** 27.6	**12.28** 6.5	61.23 60.2	39.06 36.9	21.62 23.3
LICO	**55.98** 48.7	48.23 46.9	7.75 1.7	34.29 33.8	22.24 28.6	12.05 4.2	**61.65** 59.2	**36.27** 38.3	**25.37** 21.0
				`RISE`					
Baseline	73.06 73.3	**31.89** 31.8	41.11 41.5	52.45 54.3	12.64 13.4	39.81 40.9	72.64 70.9	40.16 36.2	32.48 34.7
LICO	**75.24** 72.4	32.03 30.0	**43.21** 42.4	**56.17** 54.2	**11.67** 14.3	**44.50** 39.9	**73.60** 70.3	**38.32** 35.8	**35.27** 34.5

4 Conclusion

First, we again extend our gratitude for the reproduction efforts, as we believe they are beneficial to the entire community. We also acknowledge the shortcomings in our initial communication and the

incomplete implementation details provided, which we regret. The reason may come from the fact that we are *preparing our journal version* and *several important ddls* in work. Moving forward, we commit to improving these aspects to support future reproduction efforts better. We will update the source code in the original repository to ensure reproducibility.

However, we also recognize that the implementation errors identified could have influenced the conclusions of your study. We hope this **report** will help in **addressing these issues** to correct erroneous conclusions **resulting from the paper**.