Efficient Graph Data Augmentation on Contrastive Learning

2021.08.09

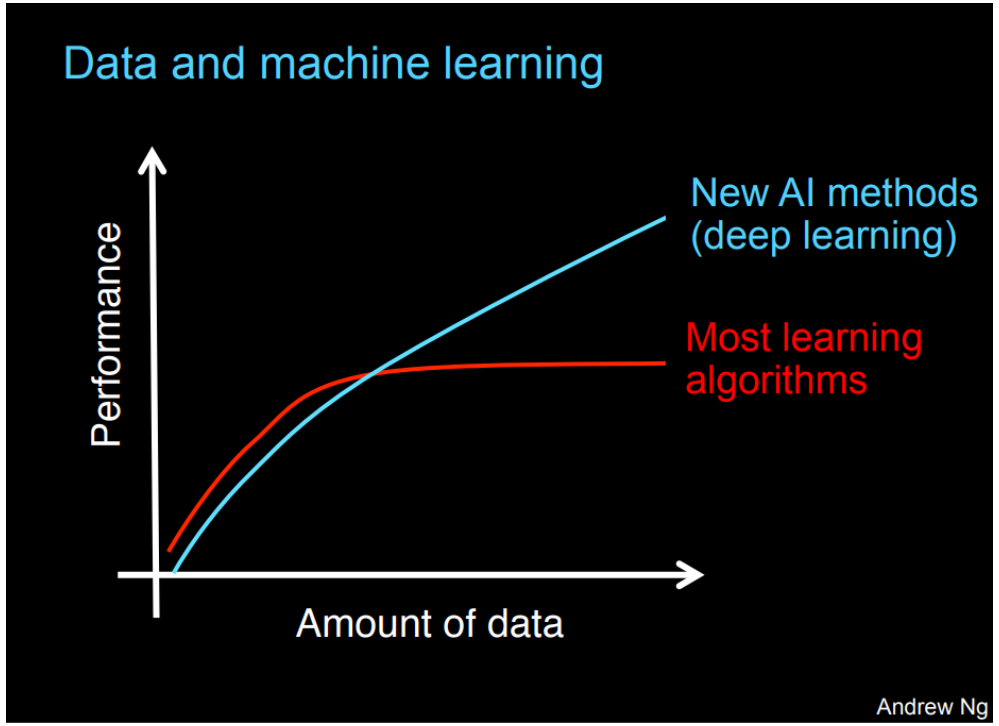
Presented by HYOJUN KIM

CONTENTS

01 Background

- What is Data Augmentation?

  Essential Problem in Deep Learning Task : Data Collection



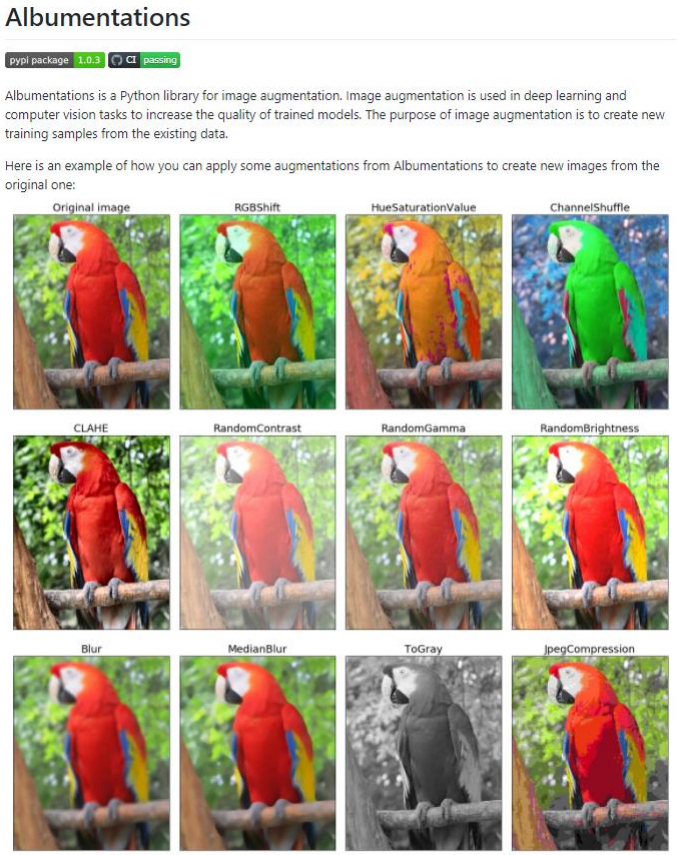
➔ Data 의 특성에 따라, Data를 증강하자

➔ Prevent overfitting
➔ Robustness

http://cs229.stanford.edu/materials/CS229-DeepLearning.pdf

01 Background

- Data Augmentation On Image Data



Shorten, Connor, and Taghi M. Khoshgoftaar. "A survey on image data augmentation for deep learning." Journal of Big Data 6.1 (2019): 1-48.

01 Background

- Data Augmentation On Text Data

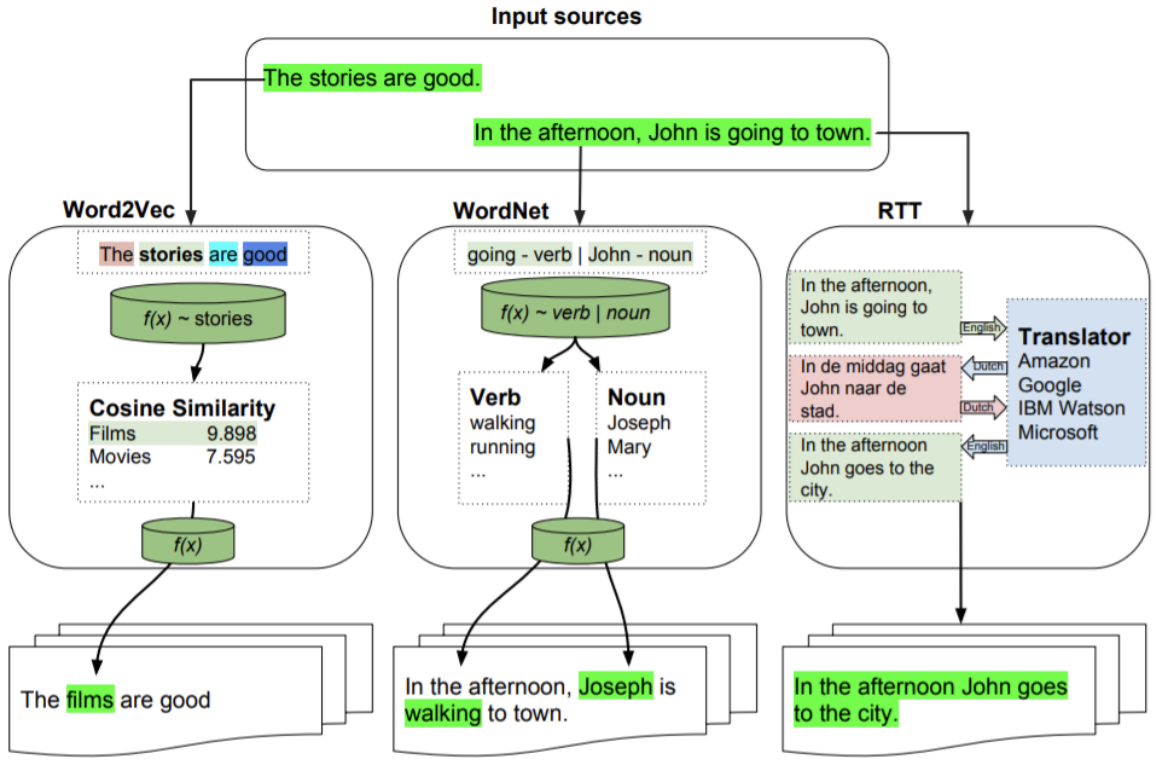
EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks

Jason Wei[1,2] Kai Zou[3]
[1]Protago Labs Research, Tysons Corner, Virginia, USA
[2]Department of Computer Science, Dartmouth College
[3]Department of Mathematics and Statistics, Georgetown University
jason.20@dartmouth.edu kz56@georgetown.edu

Operation	Sentence
None	A sad, superior human comedy played out on the back roads of life.
SR	A ***lamentable***, superior human comedy played out on the ***backward*** road of life.
RI	A sad, superior human comedy played out on ***funniness*** the back roads of life.
RS	A sad, superior human comedy played out on ***roads*** back ***the*** of life.
RD	A sad, superior human out on the roads of life.

	Training Set Size			
Model	500	2,000	5,000	full set
RNN	75.3	83.7	86.1	87.4
+EDA	79.1	84.4	87.3	88.3
CNN	78.6	85.6	87.7	88.3
+EDA	80.7	86.4	88.3	88.8
Average	76.9	84.6	86.9	87.8
+EDA	79.9	85.4	87.8	**88.6**

Table 2: Average performances (%) across five text classification tasks for models with and without EDA on different training set sizes.



Wei, Jason, and Kai Zou. "Eda: Easy data augmentation techniques for boosting performance on text classification tasks." arXiv preprint arXiv:1901.11196 (2019).

Marivate, Vukosi, and Tshephisho Sefara. "Improving short text classification through global augmentation methods." *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*. Springer, Cham, 2020.

01 Background

- Data Augmentation On Graph Data

1. Graph feature augmentation
2. Graph structure augmentation

Table 1: Overview of data augmentations for graphs.

Data augmentation	Type	Underlying Prior
Node dropping	Nodes, edges	Vertex missing does not alter semantics.
Edge perturbation	Edges	Semantic robustness against connectivity variations.
Attribute masking	Nodes	Semantic robustness against losing partial attributes per node.
Subgraph	Nodes, edges	Local structure can hint the full semantics.

02 Motivation

- Deep Graph Infomax (DGI)

C ROBUSTNESS TO CHOICE OF CORRUPTION FUNCTION

Here, we consider alternatives to our corruption function, $\mathcal{C}$, used to produce negative graphs. We generally find that, for the node classification task, DGI is stable and robust to different strategies. However, for learning graph features towards other kinds of tasks, the design of appropriate corruption strategies remains an area of open research.

We demonstrate DGI is stable to other choices of corruption functions in Appendix C, but we find those that preserve the graph structure result in the strongest features

- In paper, Corrupts the features (X~ : via row-wise shuffling of X)

➔ 모든 Feature X를 row-wise shuffling 하는 방법이 아닌
더 효과적인 Corruption Function이 있지 않을까?

02 Motivation

- DGI Objective Function

$$\mathcal{L} = \frac{1}{N+M}\left(\sum_{i=1}^{N}\mathbb{E}_{(\mathbf{X},\mathbf{A})}\left[\log \mathcal{D}\left(\vec{h}_i, \vec{s}\right)\right] + \sum_{j=1}^{M}\mathbb{E}_{(\widetilde{\mathbf{X}},\widetilde{\mathbf{A}})}\left[\log\left(1-\mathcal{D}\left(\vec{\widetilde{h}}_j, \vec{s}\right)\right)\right]\right)$$

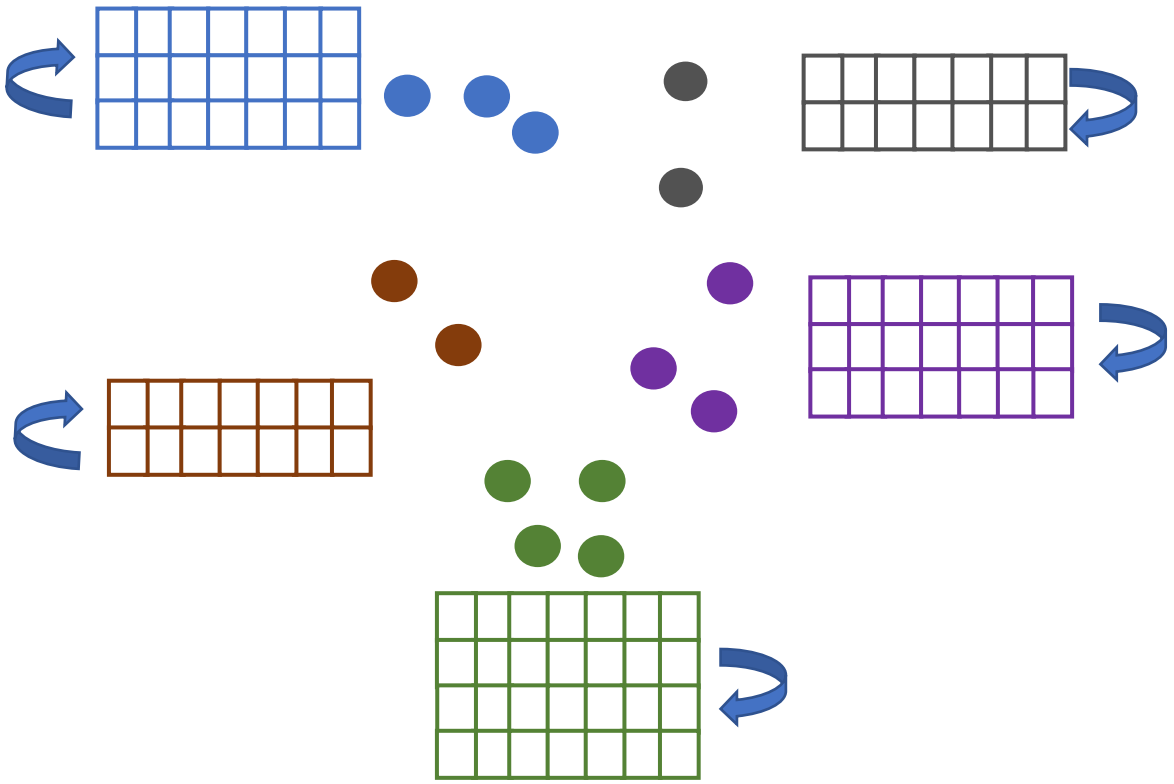
positive sampling에 대해서는 더 가까이, negative sampling에 대해서는 더 멀리 represent하도록 학습

➔ 더 나은 Representation 위해 label group 에 따른 샘플링을 한다면 효과적이지 않을까

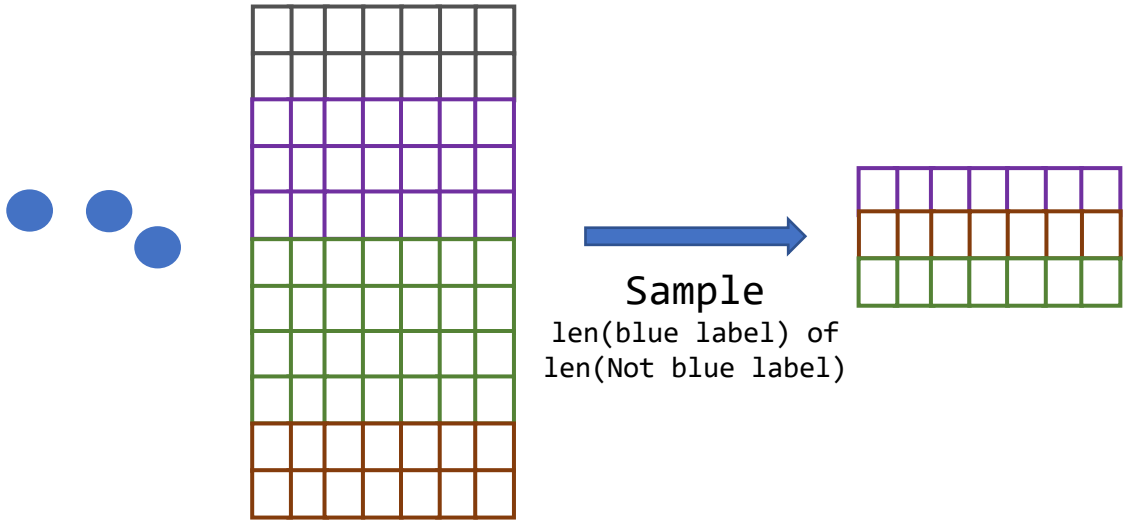
02 Motivation

- **Sampling by label group**

① Same Label Shuffle



② Different Label Shuffle



Not blue label feature

03 Experiments

▪ Experiment 1. Corrupt function

① Same Label Shuffle

```python
def corrupt_data(feat):
    '''
    label이 동일한 group끼리 corrupt하는 함수
    '''
    for num in range(n_class):
        df_feature = pd.DataFrame(feat.numpy())
        df_feature['label'] = labels
        labeld_df = df_feature[df_feature['label'].apply(lambda x: x==num)]
        shuffled_df = labeld_df.sample(frac=1)
        shuffled_df.index = labeld_df.index
        if num ==0:
            all_df = shuffled_df
        else:
            all_df = pd.concat([all_df, shuffled_df])
    all_df.sort_index(inplace=True)
    all_df.drop(columns='label', inplace=True)
    return torch.tensor(np.array(all_df))
```

```
Epoch: 0095 loss_train: 0.0677 time: 3.0119s
Epoch: 0096 loss_train: 0.0673 time: 2.8920s
Epoch: 0097 loss_train: 0.0657 time: 3.8818s
Epoch: 0098 loss_train: 0.0616 time: 3.4079s
Epoch: 0099 loss_train: 0.0580 time: 2.7172s
Epoch: 0100 loss_train: 0.0558 time: 2.6180s
Optimization Finished!
Total time elapsed: 261.4380s
classification Start
Accuracy: 0.7058400511741638
```

② Different Label Shuffle

```python
def corrupt_data_diff_label(feat):
    '''
    label이 동일하지 않은 group을 corrupt하여 할당하는 함수
    '''
    for num in range(n_class):
        df_feature = pd.DataFrame(features.numpy())
        df_feature['label'] = labels
        labeld_df = df_feature[df_feature['label'].apply(lambda x: x==num)]
        not_labeld_df = df_feature[df_feature['label'].apply(lambda x: x!=num)]
        original_index = labeld_df.index
        labeld_df = not_labeld_df.sample(len(labeld_df)) #label이 아닌 것으로 대체
        labeld_df.index = original_index
        if num ==0:
            all_df = labeld_df
        else:
            all_df = pd.concat([all_df, labeld_df])
    all_df.sort_index(inplace=True)
    all_df.drop(columns='label', inplace=True)

    return torch.tensor(np.array(all_df))
```

```
Epoch: 0095 loss_train: 0.0654 time: 2.8833s
Epoch: 0096 loss_train: 0.0611 time: 2.8982s
Epoch: 0097 loss_train: 0.0565 time: 2.7686s
Epoch: 0098 loss_train: 0.0523 time: 2.7506s
Epoch: 0099 loss_train: 0.0621 time: 2.7676s
Epoch: 0100 loss_train: 0.0552 time: 2.8055s
Optimization Finished!
Total time elapsed: 261.9501s
classification Start
Accuracy: 0.8223400712013245
```

- Same Label Shuffle 은 Corruption function으로 적합하지 못함
- 즉, Label이 동일한 특징끼리 shuffling하여 negative sampling을 하는 것은 더 멀리 represent하는 것과 맞지 않음
- Label이 동일한 특징은 비슷한 representation을 의미한다면, 오히려 positive sampling에 Augmentation으로 사용될 수 있지 않을까??

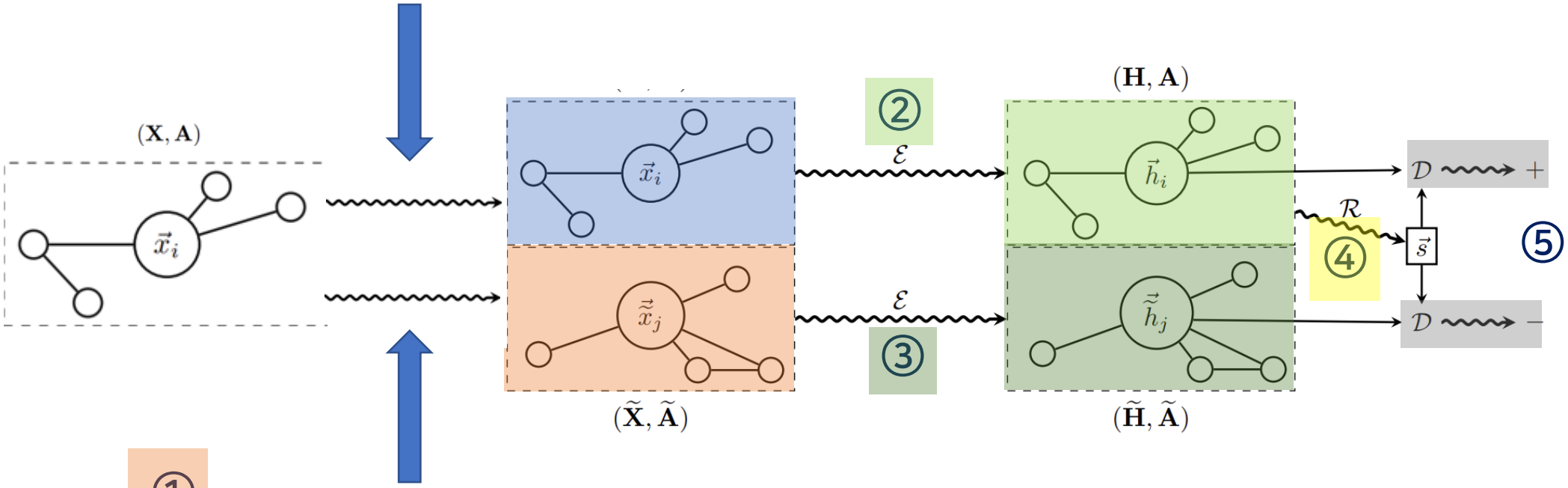
→ Augmentation on positive Sampling!!

03 Experiments

- **Overview of Model**

①

Augment function (Same Label Shuffle)



①

Corrupt function (row-wise Shuffling)

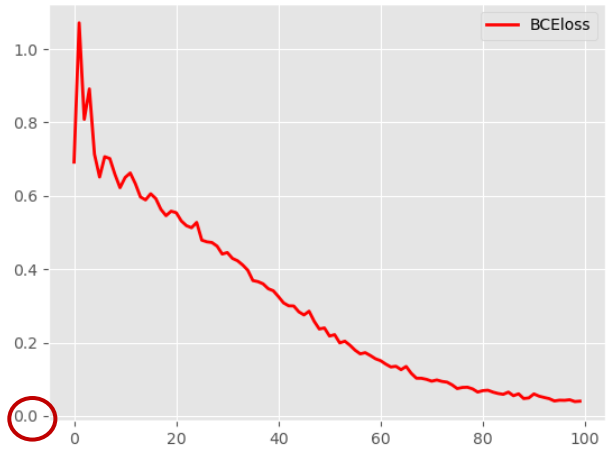
03 Experiments

- **Experiment 2. Augmentation**

In DGI Paper

Transductive

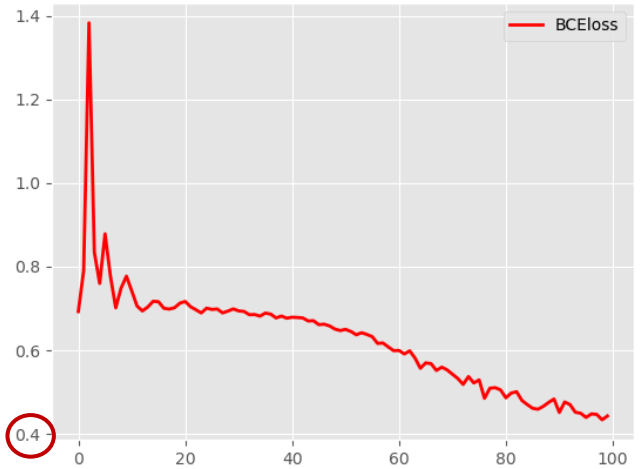
Available data	Method	Cora	Citeseer	Pubmed
X	Raw features	47.9 ± 0.4%	49.3 ± 0.2%	69.1 ± 0.3%
A, Y	LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
A	DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
X, A	DeepWalk + features	70.7 ± 0.6%	51.4 ± 0.5%	74.3 ± 0.9%
X, A	Random-Init (ours)	69.3 ± 1.4%	61.9 ± 1.6%	69.6 ± 1.9%
X, A	**DGI** (ours)	**82.3** ± 0.6%	**71.8** ± 0.7%	**76.8** ± 0.6%
X, A, Y	GCN (Kipf & Welling, 2016a)	81.5%	70.3%	79.0%
X, A, Y	Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%



Data Augment : Same label shuffle

Method	Cora
DGI with Augment(SLS)	84.9 ± 0.5 %

```
augment_features = data_augment(features)
output = model(augment_features, corrupted_features, adj)
```



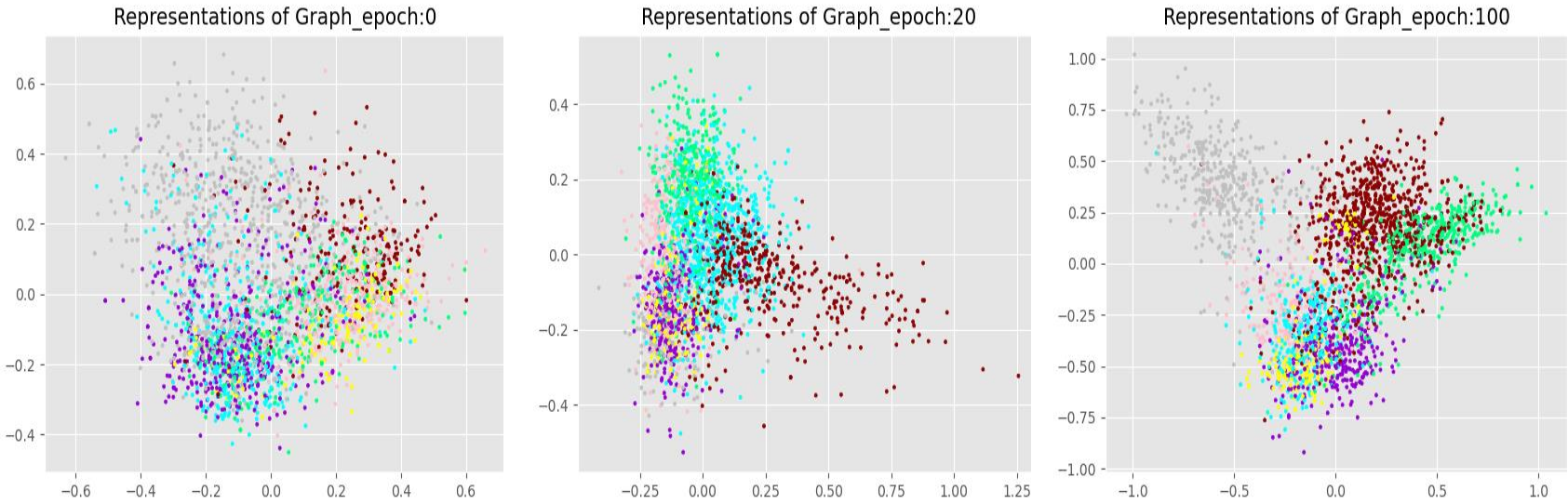

```
Epoch: 0095 loss_train: 0.4296 time: 1.8629s
Epoch: 0096 loss_train: 0.4331 time: 1.8651s
Epoch: 0097 loss_train: 0.4467 time: 1.8990s
Epoch: 0098 loss_train: 0.4339 time: 1.9145s
Epoch: 0099 loss_train: 0.4180 time: 1.7494s
Epoch: 0100 loss_train: 0.4190 time: 1.7795s
Optimization Finished!
Total time elapsed: 186.3760s
classification Start
Accuracy: 0.8520199656486511
```

```
Epoch: 0095 loss_train: 0.4493 time: 3.2318s
Epoch: 0096 loss_train: 0.4393 time: 3.0990s
Epoch: 0097 loss_train: 0.4478 time: 3.0673s
Epoch: 0098 loss_train: 0.4466 time: 3.1246s
Epoch: 0099 loss_train: 0.4337 time: 3.1382s
Epoch: 0100 loss_train: 0.4427 time: 3.1157s
Optimization Finished!
Total time elapsed: 312.8981s
classification Start
Accuracy: 0.8507800102233887
```

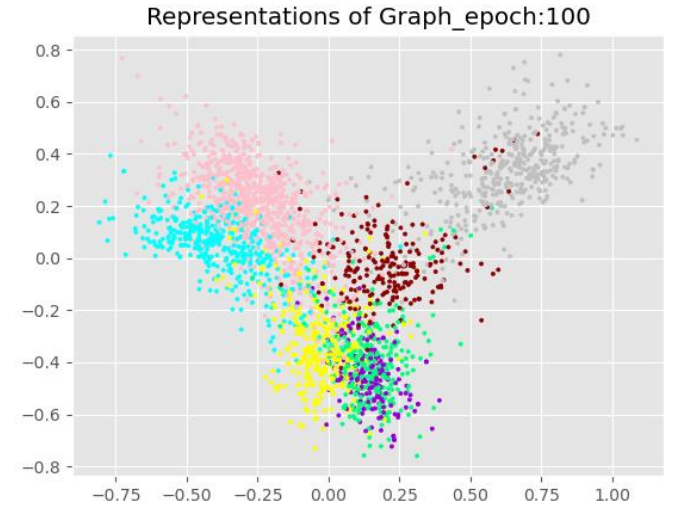
03 Experiments

- **Experiment 3. Compare Visualization**

1. PCA



Original DGI Method



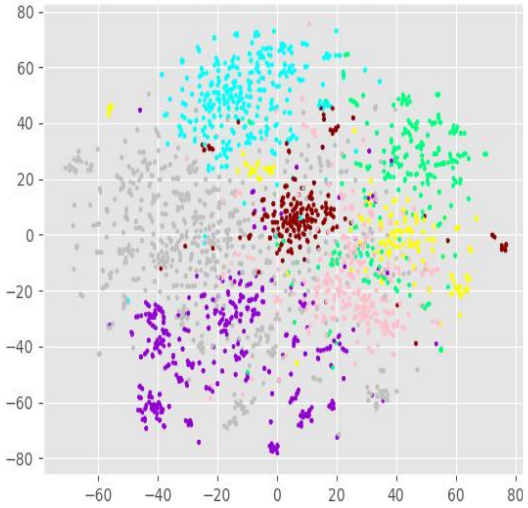

```
augment_features = data_augment(features)
output = model(augment_features, corrupted_features, adj)
```

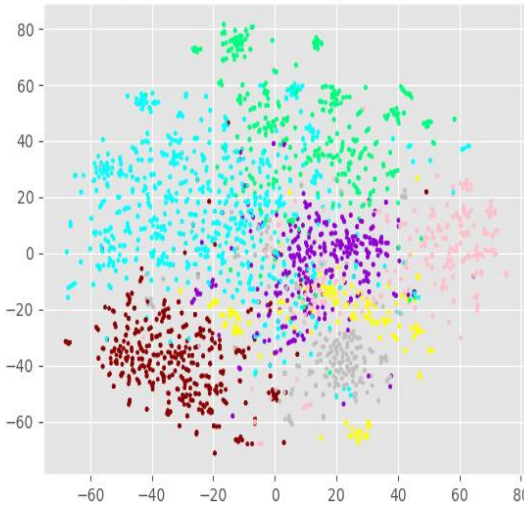
Original DGI Method +
Augment Data

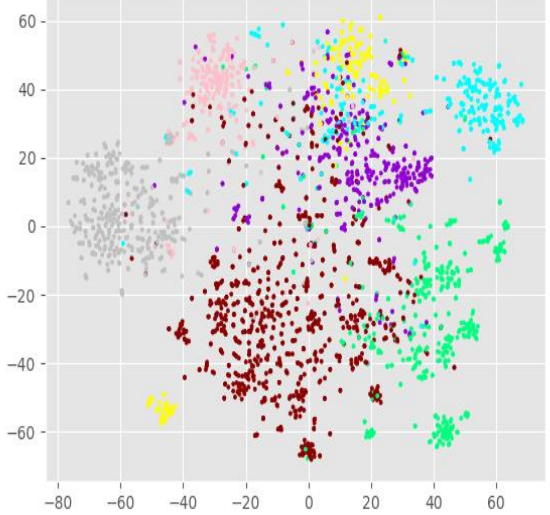
03 Experiments

- **Experiment 3. Compare Visualization**

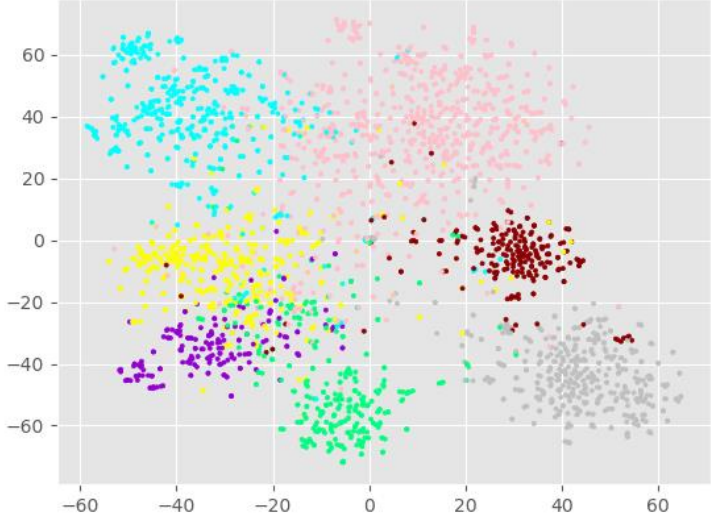
2. T-SNE







Original DGI Method




```
augment_features = data_augment(features)
output = model(augment_features, corrupted_features, adj)
```

Original DGI Method +
Augment Data

03 Experiments

- **Experiment 4.**

  DGI with Augment +
  New Corruption function

① Same Label Shuffle → **Augmentation on Postive Sampling**

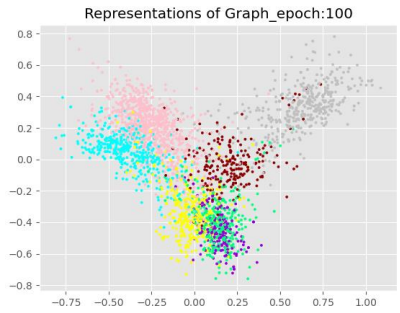
② Different Label Shuffle → **Corruption function on Negative Sampling**

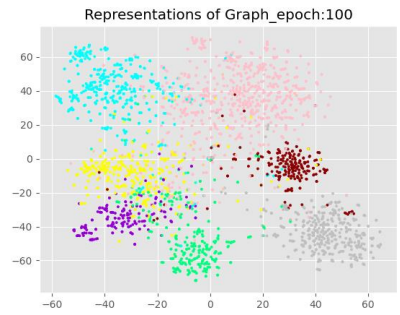
- Performance Result

Method	Cora
DGI (Paper)	82.3 ± 0.1 %
DGI + New Corruption function(DLS)	81.7 ± 0.4 %
DGI with Augment(SLS)	**84.9 ± 0.5 %**
DGI with Augment (SLS)+ New Corruption function(DLS)	**84.8 ± 0.4 %**

DGI with Augment

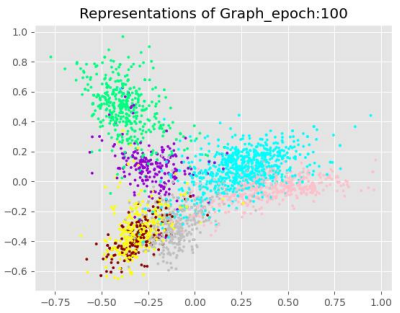
```
augment_features = data_augment(features)
output = model(augment_features, corrupted_features, adj)
```

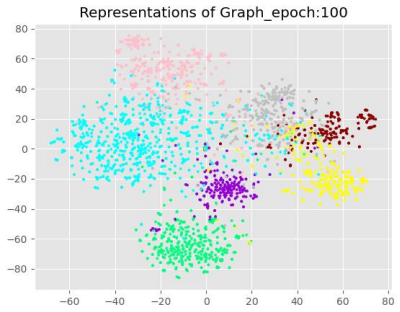




DGI with Augment + New Corruption function

```
# Corrupt input graph (shuffling for row)
corrupted_features = corrupt_data_diff_label(features)
# augment input graph
augment_features = data_augment(features)
output = model(augment_features, corrupted_features, adj)
```





04 Conclusion

Sum up

- "Same Label Shuffle" 방법론은 DGI 모델의 Positive Sampling을 증강시키는데 활용함으로써 Classification Performance가 더 우수함을 보였다.
  - Structure 유지 : Negative graph has similar levels of connectivity to the positive graph
  - Feature shuffling : positive 더 가까이, negative 더 멀리 representation하는데 효과적

- 반면, "Different Label Shuffle" 방법론은 Negative Sampling에서 새로운 Corruption function으로도 가능함 (Performance가 기존 DGI 방법론과 유사함)

What to DO

- Data Augmentation on Graph 관련 논문과의 비교
  1) Zhao, Tong, et al. "Data augmentation for graph neural networks." *arXiv preprint arXiv:2006.06830* (2020).
  2) Kong, Kezhi, et al. "Flag: Adversarial data augmentation for graph neural networks." *arXiv preprint arXiv:2010.09891* (2020).

- Corrupt 비율 (p) 설정, Corrupt feature 비율 (q) 설정 등 방법론 개선 가능 ➔ Framework로의 확장성

- Cora Dataset뿐만 아닌 Citeseer, Reddit 등 Inductive dataset 또한 performance 검증
  (데이터 특성에 따라 "Differenct Label Shuffle"의 Corruption function이 효과적일 수도 있음의 가능성)

Future work

- Semi-supervised 등 다양한 task에도 효과적일까 -> GCN 에도 적용해보자 -> 5.Experiment2
- Label이 없는 데이터 적용 불가능 -> 5. Another IDEA

05 Additional Experiment

▪ Same Label Shuffle on semi-supervised Learning

- Unsupervised graph learning 인 DGI와 괴리감 존재
- 따라서, Semi-supervised 기반의 GCN에 적용시켜보자!

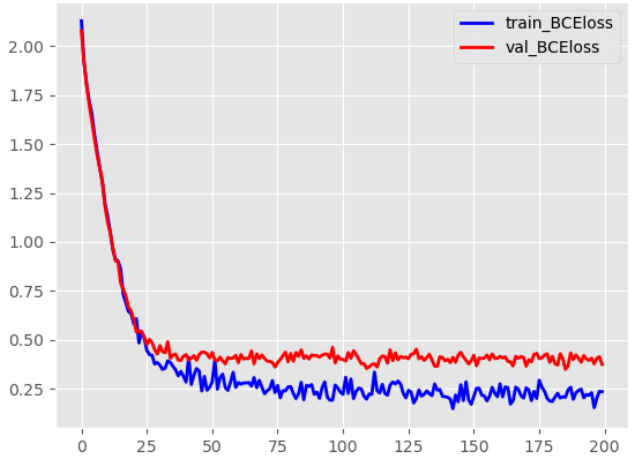
① GCN

- Paper Setting
  - Cora Datasets
  - epochs: 200, hidden units: 32

Total time elapsed: 3.0320s
cora Test set results: loss= 0.5629 accuracy= 0.8280

② GCN with Augment **(SLS)**

- Paper Setting
  - Cora Datasets
  - epochs: 200, hidden units: 32



Total time elapsed: 31.2135s
cora Test set results: loss= 0.3177 accuracy= 0.9030

05 Additional Experiment

▪ Same Label Shuffle on semi-supervised Learning

- Results (Trial and Error)

Method	Cora	Hidden units	Epochs	Total Time
GCN (Paper)	82.8 %	32	200	3.0320s
GCN	82.7 %	16	200	1.7870s
GCN	82.7 %	16	100	1.1096s
GCN with Augment(SLS)	**90.4 %**	16	200	**25.8753s**
GCN with Augment(SLS)	**90.3 %**	32	200	**31.2135s**
GCN with Augment(SLS)	88.7 %	16	100	15.3955s

- "Same Label Shuffle" 방법론은 DGI 모델의 "Corrupt function"으로부터 파생된 Idea!
- 즉, 전체 X feature를 Shuffling : Data Leakage problem 을 고려하지 못해 성능이 좋은 것이 아닐까 고안
- 결론 : Supervised 방식인 GCN 모델에서는 Train, Val, Test 중 Train 에 속하는 feature만 shuffling 해야 한다!

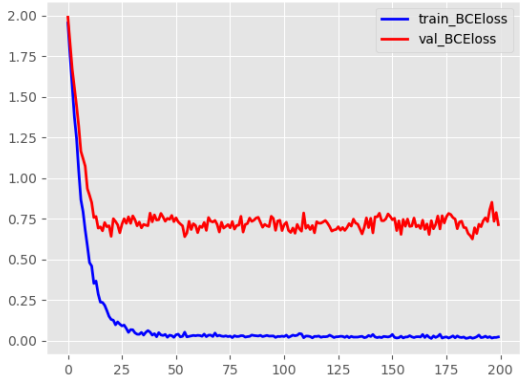
05 Additional Experiment

- Same Label Shuffle on semi-supervised Learning

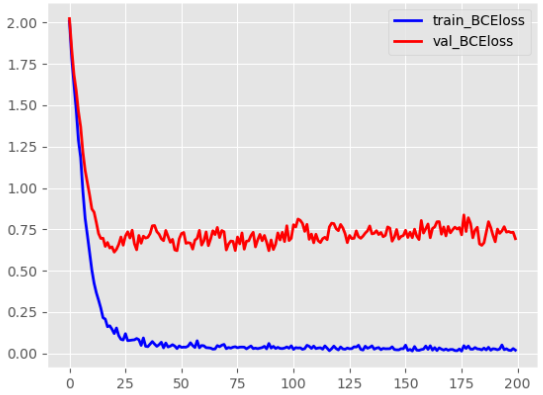
  - Results (Data Leakage Problem ➔ Only Train Data set Augmentation)

Method	Cora	Hidden units	Epochs	Total Time
GCN (Paper)	82.8 %	32	200	3.0320s
GCN with Augment(SLS)	82.5 %	32	200	6.2220s

① GCN



② GCN with Augment **(SLS)**

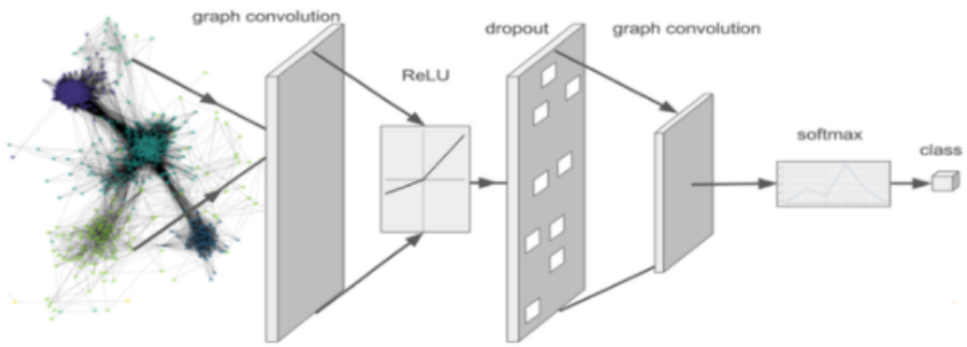


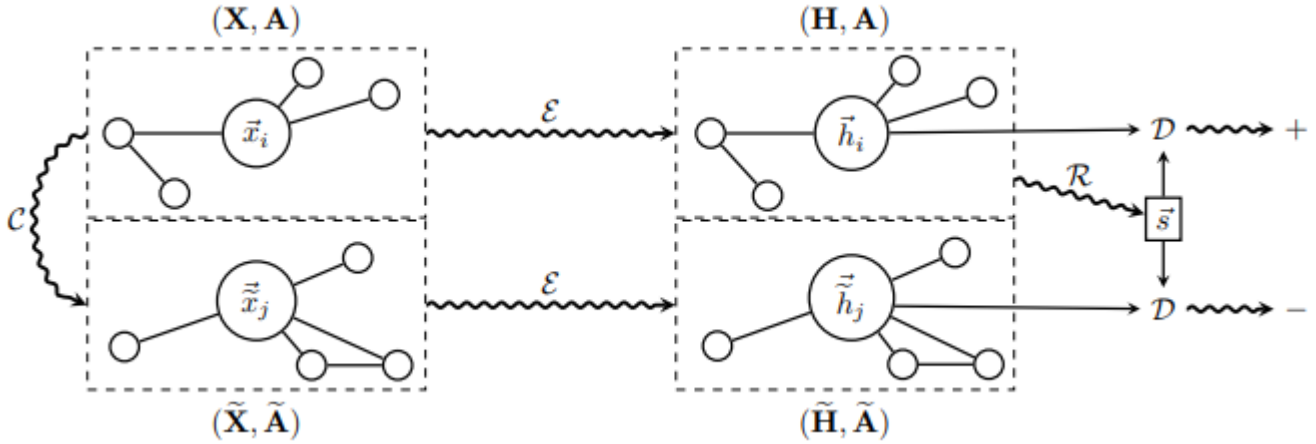
- "Same Label Shuffle" 방법론은 Supervised 기반의 학습이 이루어지는 GCN 모델에 적합하지 않다!

- "그러나, DGI 모델에서 Corruption function 은 Data Leakage 문제를 고려하지 않아도 괜찮은 것일까?"

05 Additional Experiment

- **GCN VS DGI**





$$\mathcal{L} = \frac{1}{N+M}\left(\sum_{i=1}^{N}\mathbb{E}_{(\mathbf{X},\mathbf{A})}\left[\log\mathcal{D}\left(\vec{h}_i,\vec{s}\right)\right] + \sum_{j=1}^{M}\mathbb{E}_{(\tilde{\mathbf{X}},\tilde{\mathbf{A}})}\left[\log\left(1-\mathcal{D}\left(\tilde{\vec{h}}_j,\vec{s}\right)\right)\right]\right)$$

- Class label값을 예측하는 Task

➔ Data Leakage 문제 발생
➔ 단순 Data Augmentation으로는 적합하지 못한 성능
(성능이 저하되지는 않지만, 효과가 미미함)

- Discriminator를 통해 Real, Fake인지의 Loss를 줄이는 Task

➔ Data Leakage 문제 발생하지 않음
➔ 따라서, Corruption function 시, 전체 X를 row-shuffling 하였음

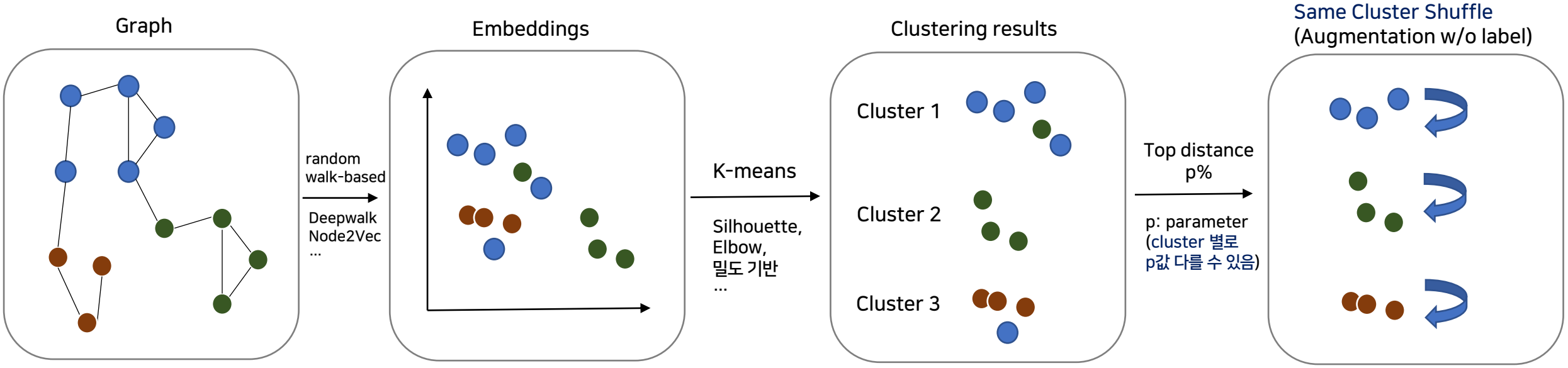
➔ Positive sampling 시에도 전체 X를 row-shuffling 해도 괜찮을 것임
➔ "Same Label Shuffle" 은 contrastive data augmentation 에 효과적인 방법론으로 사료됨!

06 Another IDEA

- **Augmentation on unlabeled data**

  1) Same Cluster Shuffle (SCS) : random walk-based representation ➔ K-means ➔ Shuffle

     - Graph Structure를 고려한 임베딩 ➔ label과는 또 다른 Strength를 가질 수 있다!
     - representation 거리가 가까운 순으로 top p%의 데이터만 shuffle 등의 augmentation을 통해 모델 정확도 개선 가능
     - label이 없는 dataset의 적용 가능성

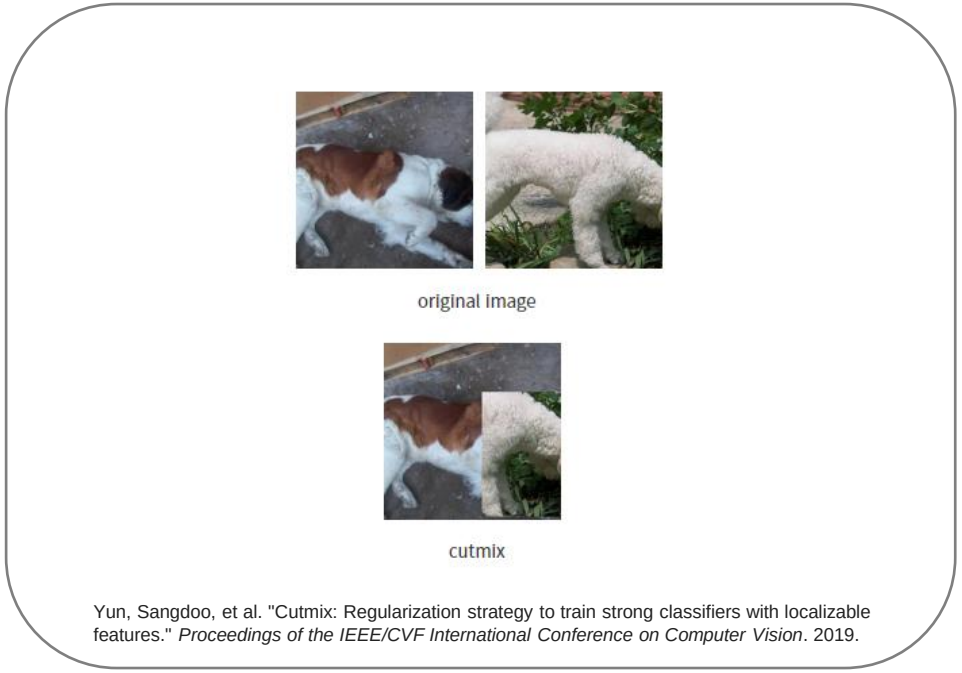


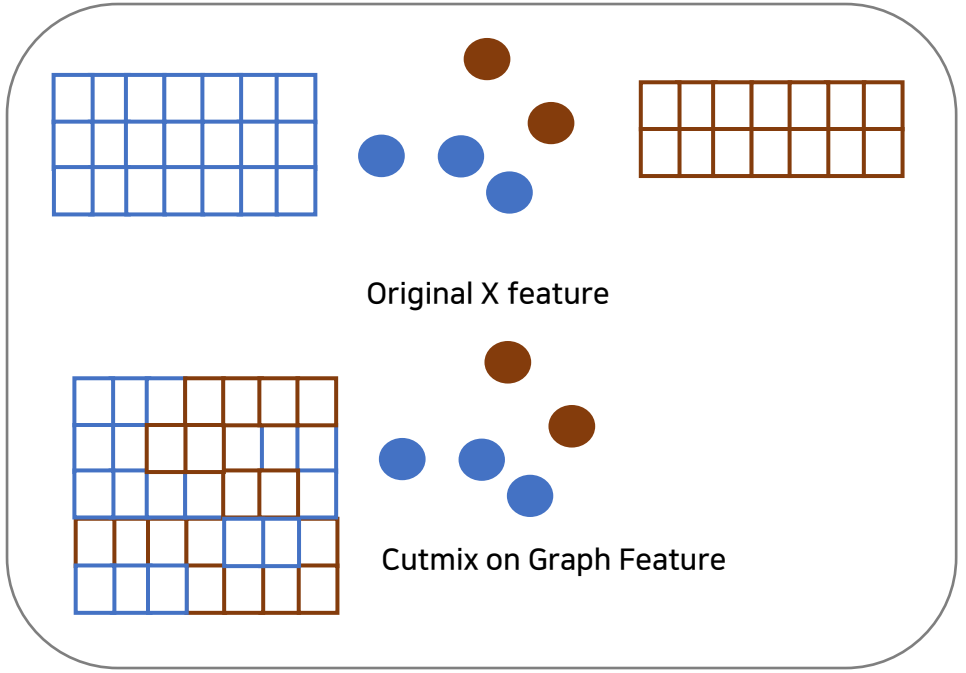
GCN, DGI, GraphSage …
classification task performance improved!!
(Robust!!)

06 Another IDEA

- **Augmentation on unlabeled data**

2) Cutmix to Graph






```
output = model(input)
loss = criterion(output, target_a) * lam + criterion(output, target_b) * (1. - lam)
```

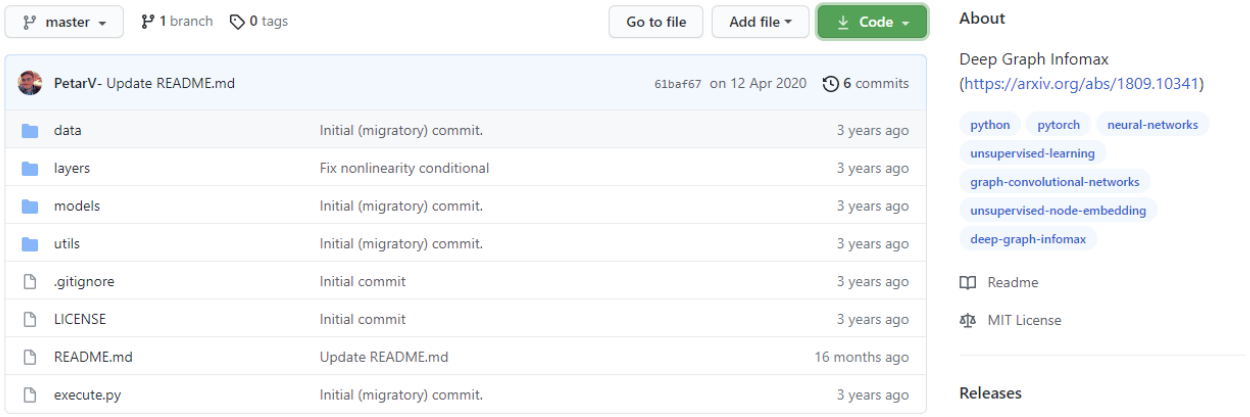
- Graph feature augmentation 으로의 방향성은 활용될 수 있으나, 해당 방법론이 효과적일지 실험 필요
- 즉, 앞서 제시한 Same label Suffle 방법에서 성능 개선이 보이지 않았기에, GCN의 Loss function을 Cutmix 처럼 제시!!

- 기대 효과 : Robustness, feature 왜곡 방지 등

07 Appendix

https://colab.research.google.com/drive/1xcp_R8YEFRU8wKAczxWgjXh2arCz3L6M?usp=sharing

- **Experiments on Official DGI Code**
  - Git clone ➔ Google Colab GPU



- **Results**

① DGI Official Code

```
Loss: tensor(0.0706, device='cuda:0', grad_fn=<BinaryCrossEntropyWithLogitsBackward>)
Loss: tensor(0.0744, device='cuda:0', grad_fn=<BinaryCrossEntropyWithLogitsBackward>)
Loss: tensor(0.0750, device='cuda:0', grad_fn=<BinaryCrossEntropyWithLogitsBackward>)
Loss: tensor(0.0649, device='cuda:0', grad_fn=<BinaryCrossEntropyWithLogitsBackward>)
Loss: tensor(0.0684, device='cuda:0', grad_fn=<BinaryCrossEntropyWithLogitsBackward>)
Early stopping!
Loading 163th epoch
```

```
Average accuracy: tensor([0.8285], device='cuda:0')
tensor(82.8480, device='cuda:0')
tensor(0.1266, device='cuda:0')
```

② DGI with Augment **(SLS)**

```
Loss: tensor(0.3593, device='cuda:0', grad_fn=<BinaryCrossEntropyWithLogitsBackward>)
Loss: tensor(0.3578, device='cuda:0', grad_fn=<BinaryCrossEntropyWithLogitsBackward>)
Loss: tensor(0.3535, device='cuda:0', grad_fn=<BinaryCrossEntropyWithLogitsBackward>)
Loss: tensor(0.3475, device='cuda:0', grad_fn=<BinaryCrossEntropyWithLogitsBackward>)
Loss: tensor(0.3521, device='cuda:0', grad_fn=<BinaryCrossEntropyWithLogitsBackward>)
Early stopping!
Loading 178th epoch
```

```
Average accuracy: tensor([0.9114], device='cuda:0')
tensor(91.1380, device='cuda:0')
tensor(0.0945, device='cuda:0')
```

Github(Local code) :

- **Why Official Code > Local Code ?**

Method	Official Code in Colab	Implementation code in Local
DGI (Paper)	82.3 ± 0.1 %	81.9920 ± 0.3 %
DGI with Augment(SLS)	84.9 ± 0.5 %	91.4 ± 0.3%

1) GPU > CPU

2) official code 가 더 정교함
(특히 early stopping을 통해 적절한 representation 에 맞는 학습을 한 것으로 사료됨)

➔ 데이터 증강이 매 Epoch마다 되면서 더 정교한(고도화된) 모델이 필요할 것
➔ 기존 local 코드보다 Github Official Code 가 더 적합할 것이다!!

Q & A