DeepGCNs: Can GCNs Go as Deep as CNNs

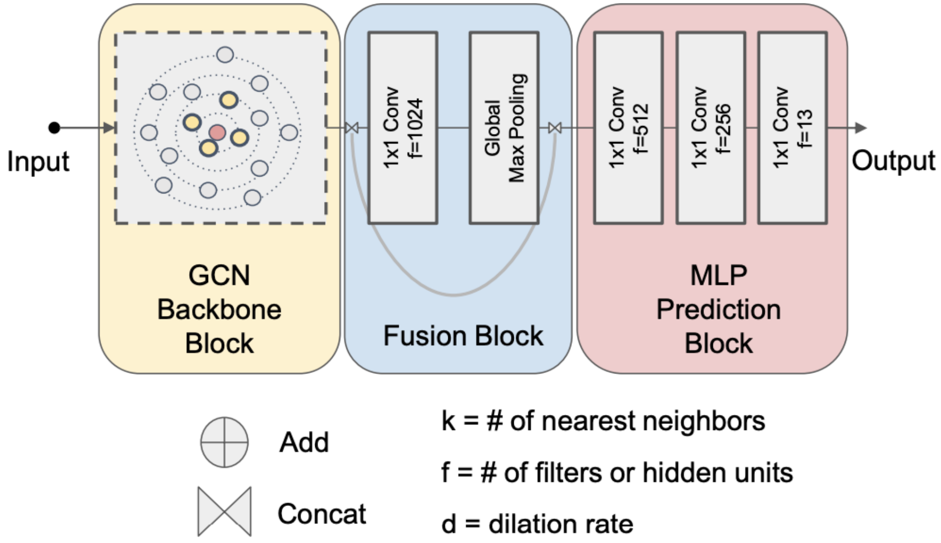
(Li et al. ICCV 2019)

PIAO TAIREN(박태임)

Seoul National University

Introduction

- GCNs(Representation Learning on Graphs)
  - Why GCNs showed promising results in many tasks?
  - Why most SOTA GCN models no deeper than 3 or 4 layers?
- How to make GCNs deeper?
  - Residual Learning for GCNs.
  - Dense connections in GCNs.
  - Dilated Aggregation in GCNs.
- Experiments
  - S3DIS Semantic Segmentation.

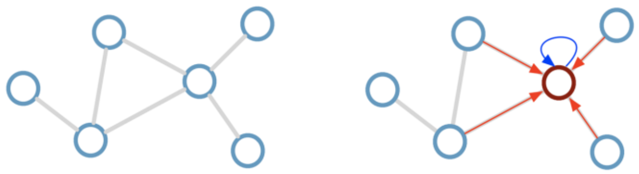


Representation Learning on Graphs

- GCNs(Graph Convolutional Networks)
  - GCNs have been gaining a lot of momentum in the last few years.
  - Two main factors
    - Increasing proliferation of non-Euclidean data in real-world applications
    - Limited performance of CNNs when dealing with such data
    - CNN: Image, Video
    - GCNs: Social graphs, biological data

Consider this undirected graph:

Calculate update for node in red:

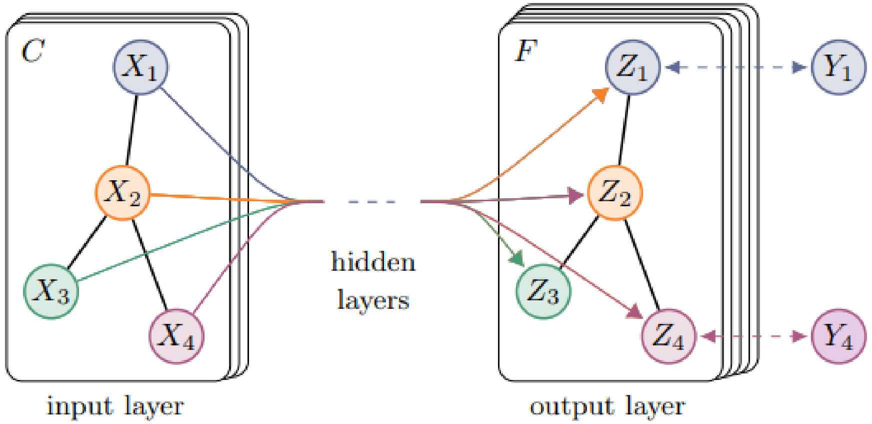


$\mathcal{N}_i$: neighbor indices

c_{ij} : norm. constant (fixed/trainable)

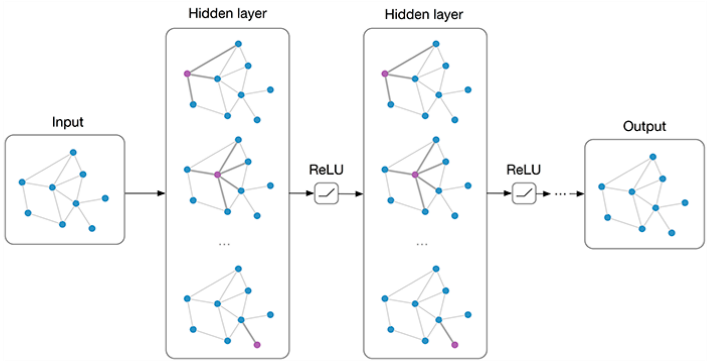
Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

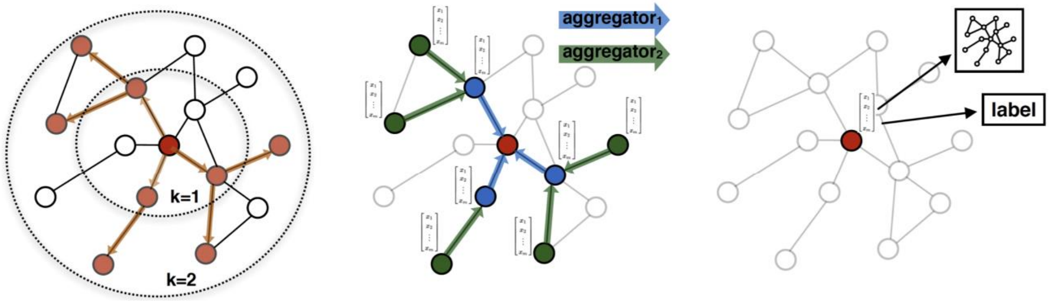


Most SOTA GCN models no deeper than 3 or 4 layers

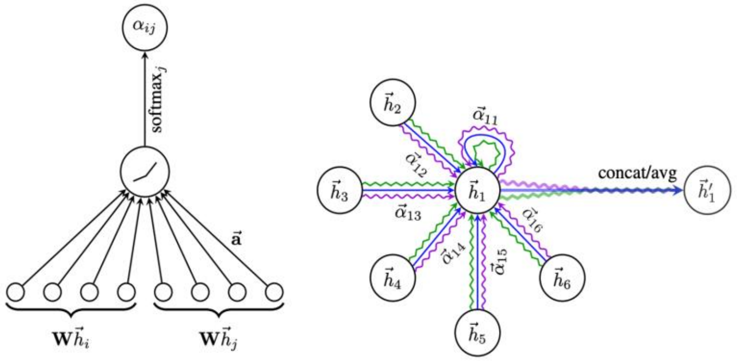
- Why?



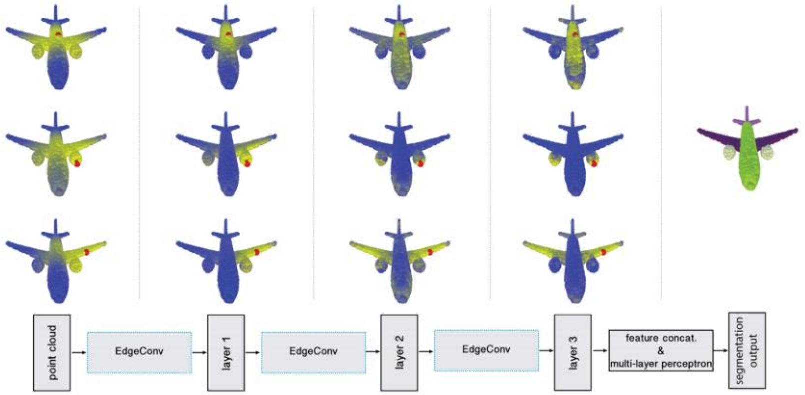
Kipf, T.N. and Welling, M., 2016. Semi-Supervised Classification with Graph Convolutional Networks.



Hamilton, W.L., Ying, R. and Leskovec, J., 2017. Inductive Representation Learning on Large Graphs.



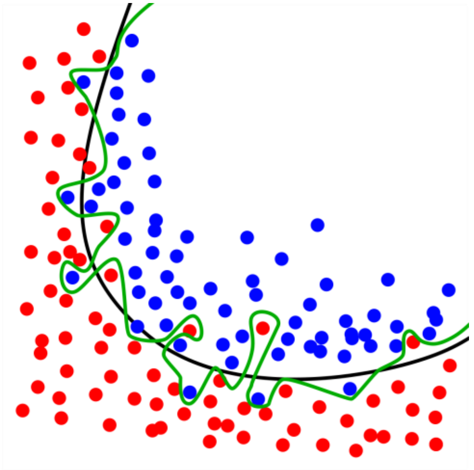
Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P. and Bengio, Y., 2018. Graph Attention Networks.



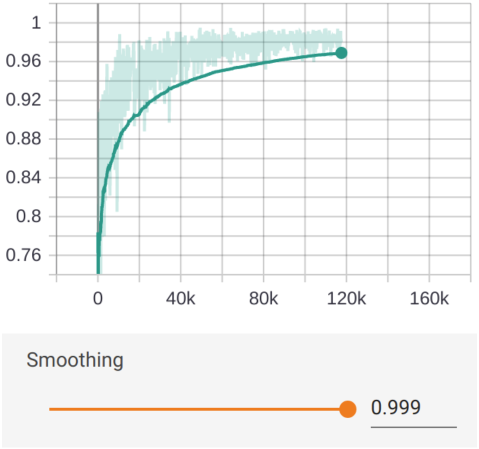
Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M. and Solomon, J.M., 2018. Dynamic Graph CNN for Learning on Point Clouds.

Why GCNs are limited to shallow structures?

- GCNs(Graph Convolutional Networks)
  - Over-fitting
  - Over-smoothing
  - Vanishing gradient



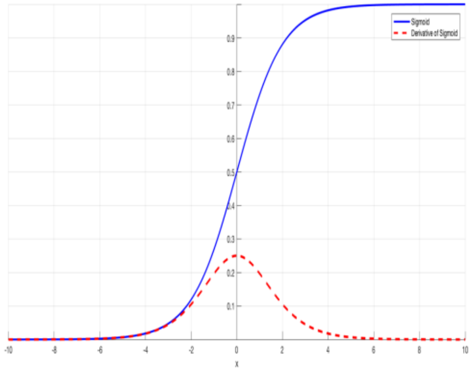
Over−fitting



Smoothing

0.999

Over−smoothing



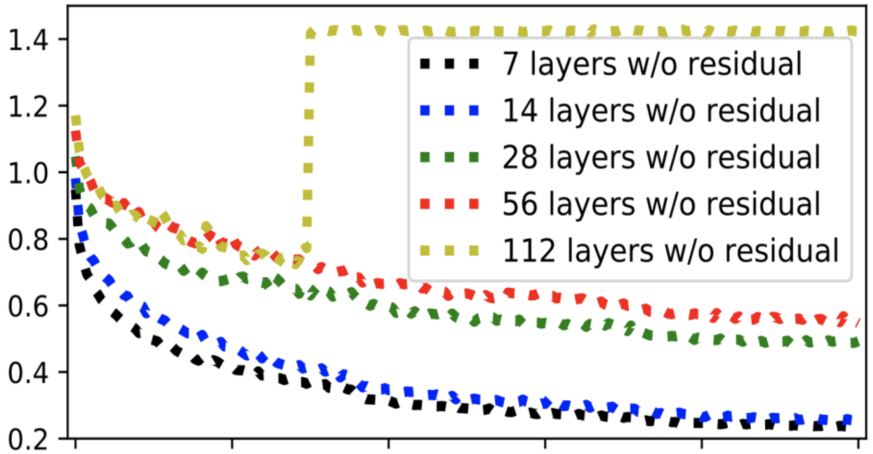
Vanishing Gradient

Figures from https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484

Vanishing Gradient Problem

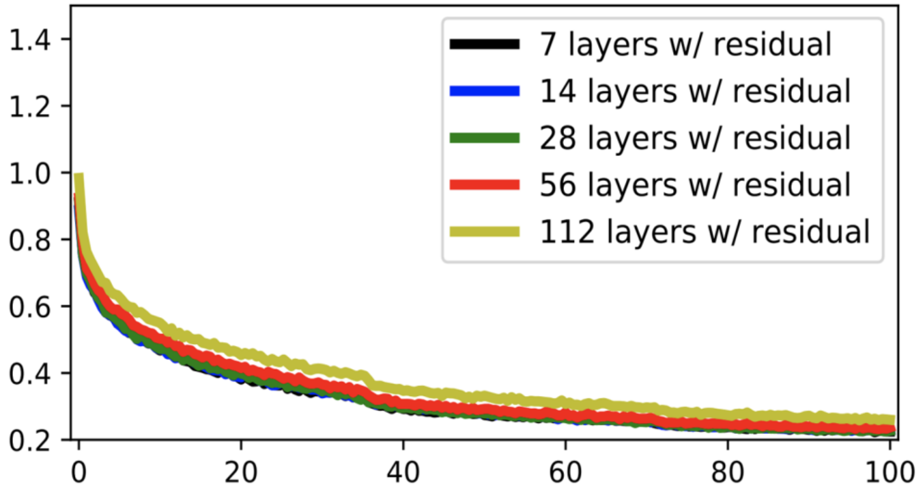
- Vanishing Gradient Problem
    - Training loss for GCNs with 7, 14, 28 and 56 layers, with and without residual connections.
    - Adding more layers without residual connections translates to substantially higher loss.
    - Introduce skip connections, the network converges with no problems.

Deeper GCNs don’t converge well.



PlainGCNs

Even a 112-layer deep GCN converges well!!!



ResGCNs

Residual Learning for GCNs

- Inspired by ResNet(He et al. CVPR 2016)
- A graph residual learning framework
  - Skip connection.
    - Adding the input features to the output features.
  - The deep network converges with no problems.
  - Learns an underlying mapping H by fitting another mapping F
  - ResGCN

$$
\begin{aligned}
\mathcal{G}_{l+1} &= \mathcal{H}(\mathcal{G}_l, \mathcal{W}_l) \\
&= \mathcal{F}(\mathcal{G}_l, \mathcal{W}_l) + \mathcal{G}_l = \mathcal{G}^{res}_{l+1} + \mathcal{G}_l.
\end{aligned}
$$

Residual Learning

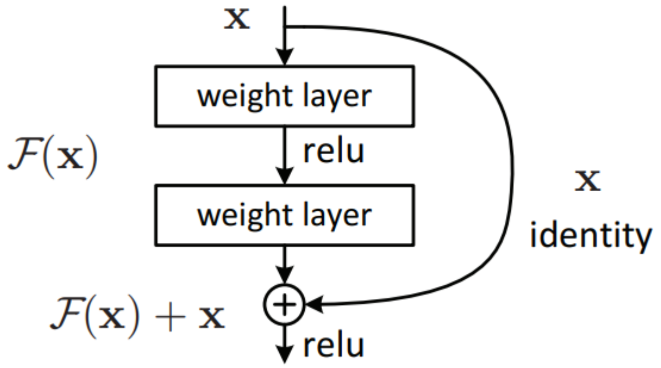
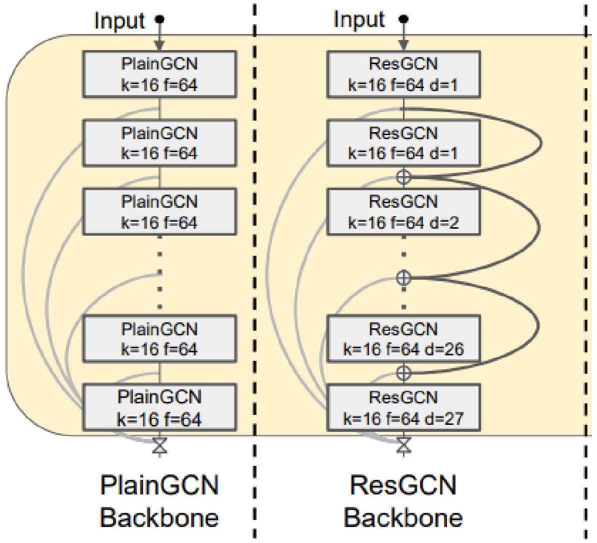

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
$\mathbf{x}$
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2. Residual learning: a building block.



Dense connections in GCNs

- Inspired by DenseNet(Huang et al. CVPR 2017)
  - Improves information flow in the network.
  - Enables efficient reuse of features among layers.
- Dense connectivity among layers.
  - *T* is vertex-wise concatenation function.
  - g_{l+1}consists of all the GCN transitions from previous layers.
  - Instead of adding input features to the output features. DenseGCN concatenate them together.
  - Dimension of each vertex feature of $\mathcal{G}_{l+1}$ is $D_0 + D \times (l+1)$.

$$\begin{aligned}\mathcal{G}_{l+1} &= \mathcal{H}(\mathcal{G}_l, \mathcal{W}_l) \\ &= \mathcal{T}(\mathcal{F}(\mathcal{G}_l, \mathcal{W}_l), \mathcal{G}_l) \\ &= \mathcal{T}(\mathcal{F}(\mathcal{G}_l, \mathcal{W}_l), ..., \mathcal{F}(\mathcal{G}_0, \mathcal{W}_0), \mathcal{G}_0).\end{aligned}$$

Dense connections

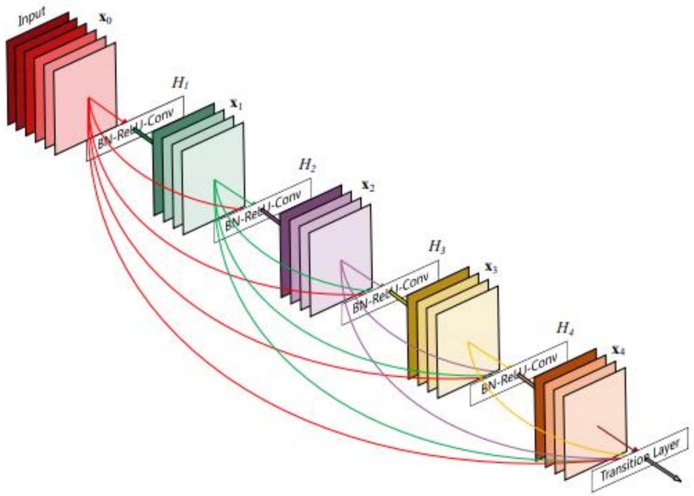
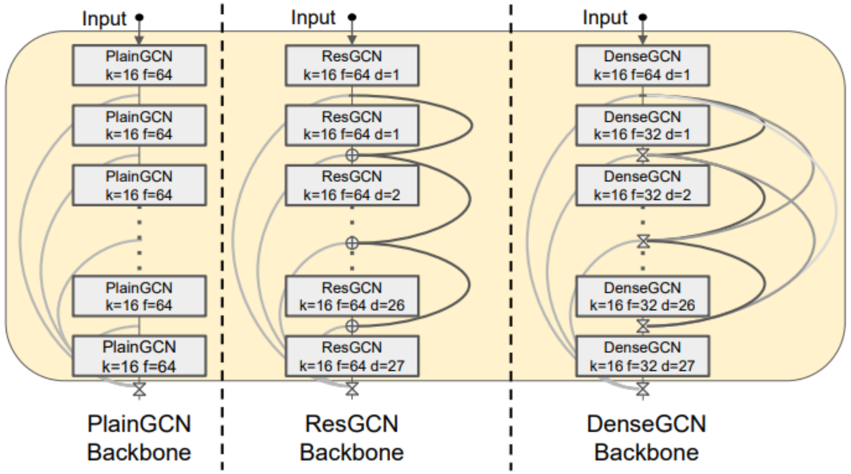


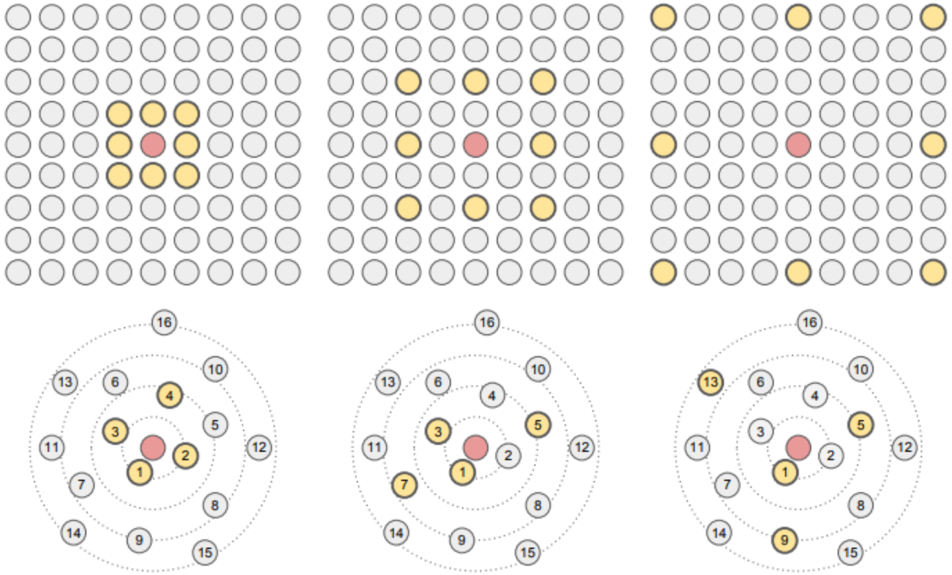
Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.



Dilated Aggregation in GCNs

- Dilated convolutions
  - Increase the receptive field without losing resolution.
  - Stacking layers with different dilation rates and works very well for CNNs.
  - Especially using in semantic segmentation
- To alleviate spatial information loss caused by pooling operation.
- For GCNs, simply dilate the convolution by skipping neighbors.

Dilated aggregation



Experiments

- S3DIS Semantic Segmentation

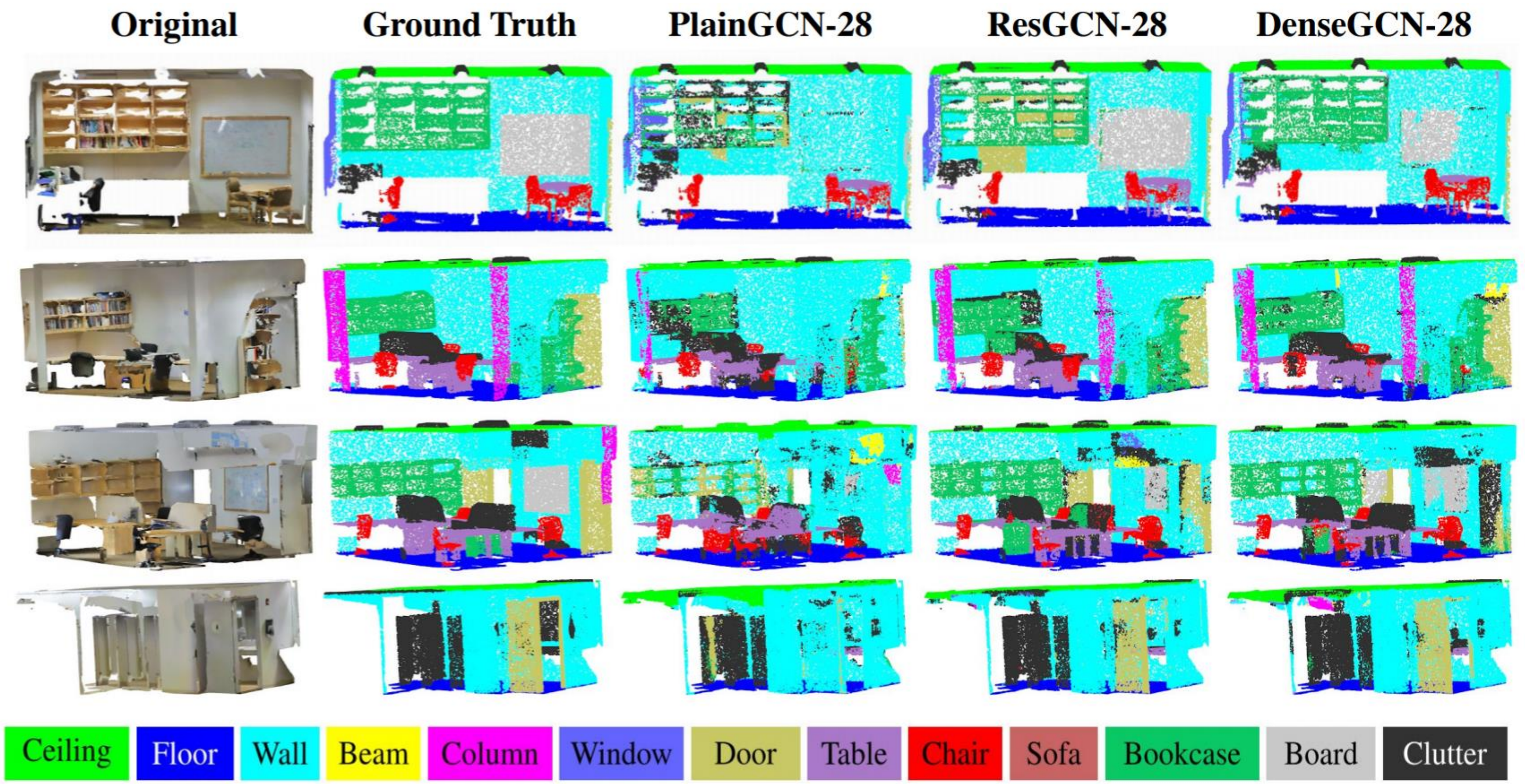
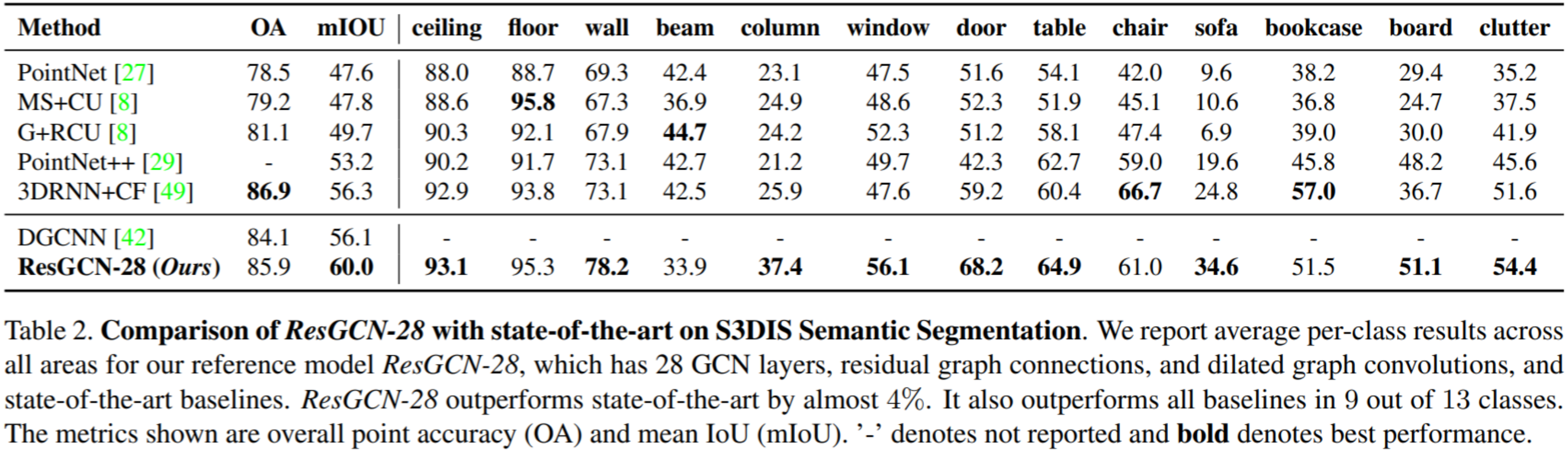


Figure 4. **Qualitative Results on S3DIS Semantic Segmentation**. We show here the effect of adding residual and dense graph connections to deep GCNs. *PlainGCN-28*, *ResGCN-28*, and *DenseGCN-28* are identical except for the presence of residual graph connections in *ResGCN-28* and dense graph connections in *DenseGCN-28*. We note how both residual and dense graph connections have a substantial effect on hard classes like board, bookcase, and sofa. These are lost in the results of *PlainGCN-28*.

Experiments

- S3DIS Semantic Segmentation



Method	OA	mIOU	ceiling	floor	wall	beam	column	window	door	table	chair	sofa	bookcase	board	clutter
PointNet [27]	78.5	47.6	88.0	88.7	69.3	42.4	23.1	47.5	51.6	54.1	42.0	9.6	38.2	29.4	35.2
MS+CU [8]	79.2	47.8	88.6	**95.8**	67.3	36.9	24.9	48.6	52.3	51.9	45.1	10.6	36.8	24.7	37.5
G+RCU [8]	81.1	49.7	90.3	92.1	67.9	**44.7**	24.2	52.3	51.2	58.1	47.4	6.9	39.0	30.0	41.9
PointNet++ [29]	-	53.2	90.2	91.7	73.1	42.7	21.2	49.7	42.3	62.7	59.0	19.6	45.8	48.2	45.6
3DRNN+CF [49]	**86.9**	56.3	92.9	93.8	73.1	42.5	25.9	47.6	59.2	60.4	**66.7**	24.8	**57.0**	36.7	51.6
DGCNN [42]	84.1	56.1	-	-	-	-	-	-	-	-	-	-	-	-	-
ResGCN-28 (*Ours*)	85.9	**60.0**	**93.1**	95.3	**78.2**	33.9	**37.4**	**56.1**	**68.2**	**64.9**	61.0	**34.6**	51.5	**51.1**	**54.4**

Table 2. **Comparison of *ResGCN-28* with state-of-the-art on S3DIS Semantic Segmentation**. We report average per-class results across all areas for our reference model *ResGCN-28*, which has 28 GCN layers, residual graph connections, and dilated graph convolutions, and state-of-the-art baselines. *ResGCN-28* outperforms state-of-the-art by almost 4%. It also outperforms all baselines in 9 out of 13 classes. The metrics shown are overall point accuracy (OA) and mean IoU (mIoU). '-' denotes not reported and **bold** denotes best performance.

Conclusion

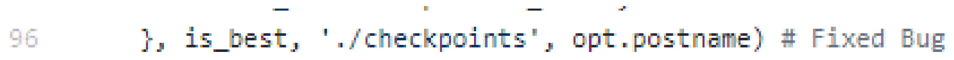
- GCN(Graph Convolution Network)
  - GCNs showed promising results in many tasks
    - Non-Euclidean data.
  - Most SOTA GCN models no deeper than 3 or 4 layers
    - Over-fitting, over-smoothing and vanishing gradient.
- How to make GCNs deeper?
  - Residual skip connections.
  - Dense skip connections.
  - Dilated graph convolutions.
- Experiments
  - S3DIS Semantic Segmentation.

Reproduction of the Paper’s Method

PIAO TAIREN(박태임)
Seoul National University

Code Link

- Code Link of Paper Reproduction
  - https://github.com/ptrandpxq/PIAO_TAIREN_GCN_Final_DeepGCNs
    - Referenced the original code
- Modification
  - I modified the usage guideline to make the code easy to understand.
  - I fixed some bugs of original code
    - Data directory and configuration.
    - Original code has some bugs for running.
    - E.g.: train.py line 96

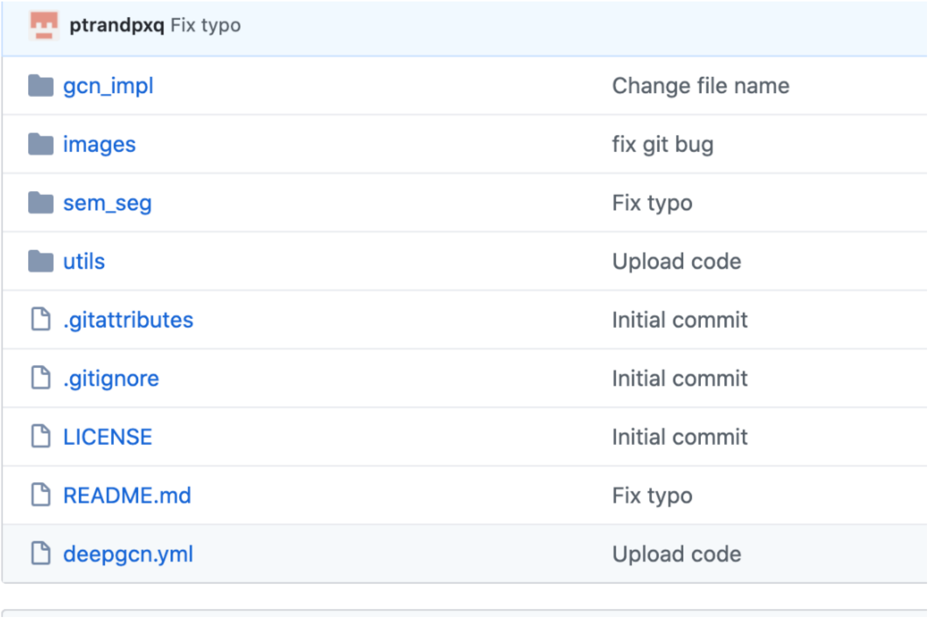


```
96        }, is_best, './checkpoints', opt.postname) # Fixed Bug
```

  - It is easy to run the code to follow the usage guideline.

Code Explanations

- Code Architecture
  - gcn_impl
    - PyTorch implementation of GCN library.
      - Implementation of GCNs and Layers
        - Node, Edge,Resconv2D, MLP
  - sem_seg
    - S3DIS Semantic segmentation task
      - Implementation of the task
      - Architectures, train.py, and configurations.
      - I put a pre-trained model in it.
  - utils
    - Basic module implementation of DeepGCN
      - loss, optimizer, checkpoint and dataloader.
  - images
    - Guideline images

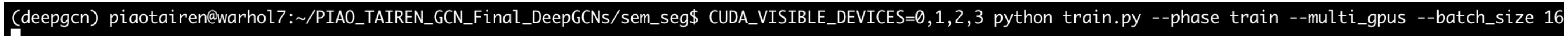


ptrandpxq Fix typo

gcn_impl	Change file name
images	fix git bug
sem_seg	Fix typo
utils	Upload code
.gitattributes	Initial commit
.gitignore	Initial commit
LICENSE	Initial commit
README.md	Fix typo
deepgcn.yml	Upload code

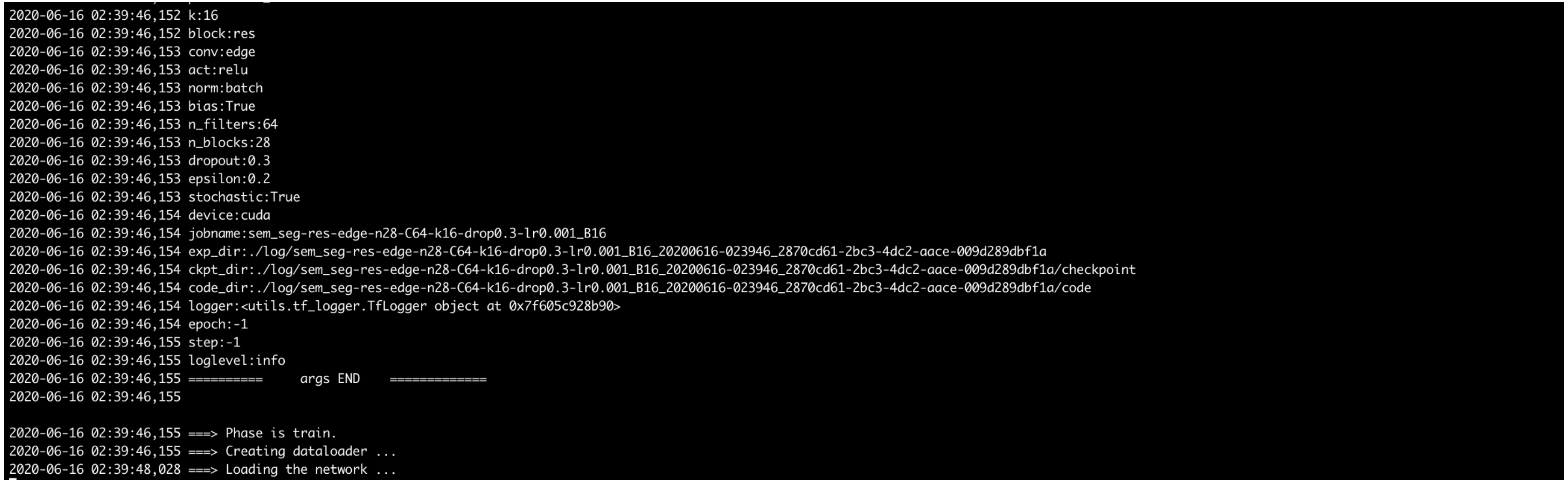
Experiment 1 Training ResGCN-28 from Scratch

- Train the ResGCN-28 from scratch

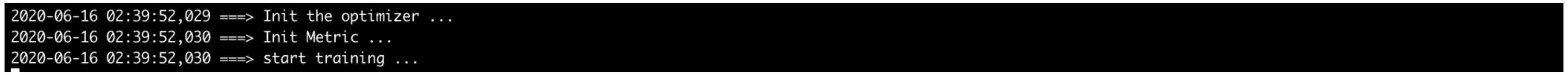


```
(deepgcn) piaotairen@warhol7:~/PIAO_TAIREN_GCN_Final_DeepGCNs/sem_seg$ CUDA_VISIBLE_DEVICES=0,1,2,3 python train.py --phase train --multi_gpus --batch_size 16
```

- Configurations



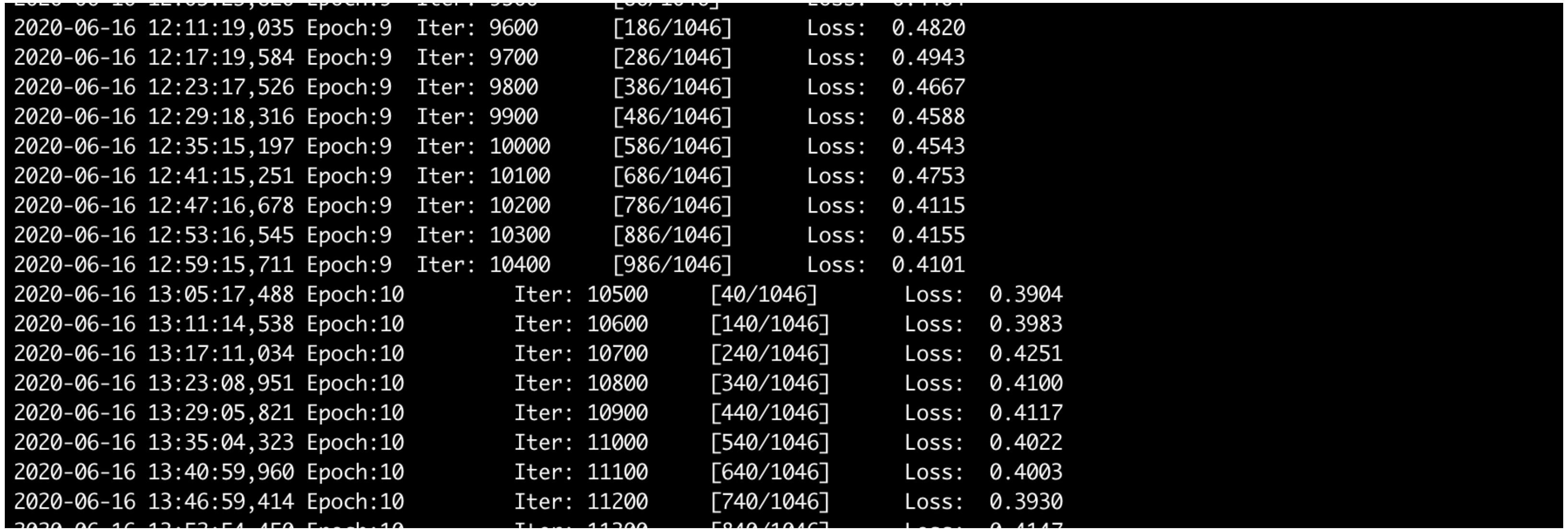
```
2020-06-16 02:39:46,152 k:16
2020-06-16 02:39:46,152 block:res
2020-06-16 02:39:46,153 conv:edge
2020-06-16 02:39:46,153 act:relu
2020-06-16 02:39:46,153 norm:batch
2020-06-16 02:39:46,153 bias:True
2020-06-16 02:39:46,153 n_filters:64
2020-06-16 02:39:46,153 n_blocks:28
2020-06-16 02:39:46,153 dropout:0.3
2020-06-16 02:39:46,153 epsilon:0.2
2020-06-16 02:39:46,153 stochastic:True
2020-06-16 02:39:46,154 device:cuda
2020-06-16 02:39:46,154 jobname:sem_seg-res-edge-n28-C64-k16-drop0.3-lr0.001_B16
2020-06-16 02:39:46,154 exp_dir:./log/sem_seg-res-edge-n28-C64-k16-drop0.3-lr0.001_B16_20200616-023946_2870cd61-2bc3-4dc2-aace-009d289dbf1a
2020-06-16 02:39:46,154 ckpt_dir:./log/sem_seg-res-edge-n28-C64-k16-drop0.3-lr0.001_B16_20200616-023946_2870cd61-2bc3-4dc2-aace-009d289dbf1a/checkpoint
2020-06-16 02:39:46,154 code_dir:./log/sem_seg-res-edge-n28-C64-k16-drop0.3-lr0.001_B16_20200616-023946_2870cd61-2bc3-4dc2-aace-009d289dbf1a/code
2020-06-16 02:39:46,154 logger:<utils.tf_logger.TfLogger object at 0x7f605c928b90>
2020-06-16 02:39:46,154 epoch:-1
2020-06-16 02:39:46,155 step:-1
2020-06-16 02:39:46,155 loglevel:info
2020-06-16 02:39:46,155 ==========     args END     =============
2020-06-16 02:39:46,155

2020-06-16 02:39:46,155 ===> Phase is train.
2020-06-16 02:39:46,155 ===> Creating dataloader ...
2020-06-16 02:39:48,028 ===> Loading the network ...
```



```
2020-06-16 02:39:52,029 ===> Init the optimizer ...
2020-06-16 02:39:52,030 ===> Init Metric ...
2020-06-16 02:39:52,030 ===> start training ...
```

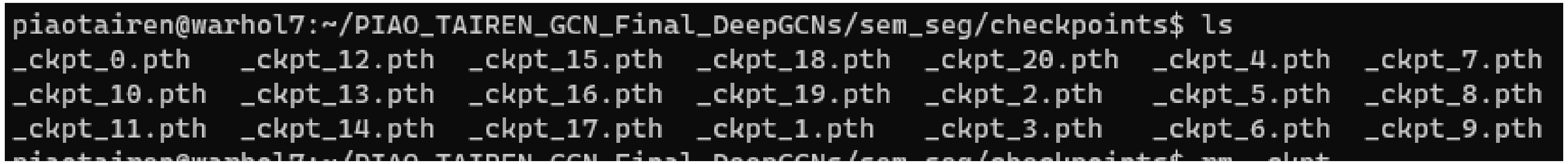
Experiment 1 Training ResGCN-28 from Scratch

- Training Process



```
2020-06-16 12:11:19,035 Epoch:9  Iter: 9600      [186/1046]      Loss:  0.4820
2020-06-16 12:17:19,584 Epoch:9  Iter: 9700      [286/1046]      Loss:  0.4943
2020-06-16 12:23:17,526 Epoch:9  Iter: 9800      [386/1046]      Loss:  0.4667
2020-06-16 12:29:18,316 Epoch:9  Iter: 9900      [486/1046]      Loss:  0.4588
2020-06-16 12:35:15,197 Epoch:9  Iter: 10000     [586/1046]      Loss:  0.4543
2020-06-16 12:41:15,251 Epoch:9  Iter: 10100     [686/1046]      Loss:  0.4753
2020-06-16 12:47:16,678 Epoch:9  Iter: 10200     [786/1046]      Loss:  0.4115
2020-06-16 12:53:16,545 Epoch:9  Iter: 10300     [886/1046]      Loss:  0.4155
2020-06-16 12:59:15,711 Epoch:9  Iter: 10400     [986/1046]      Loss:  0.4101
2020-06-16 13:05:17,488 Epoch:10        Iter: 10500     [40/1046]       Loss:  0.3904
2020-06-16 13:11:14,538 Epoch:10        Iter: 10600     [140/1046]      Loss:  0.3983
2020-06-16 13:17:11,034 Epoch:10        Iter: 10700     [240/1046]      Loss:  0.4251
2020-06-16 13:23:08,951 Epoch:10        Iter: 10800     [340/1046]      Loss:  0.4100
2020-06-16 13:29:05,821 Epoch:10        Iter: 10900     [440/1046]      Loss:  0.4117
2020-06-16 13:35:04,323 Epoch:10        Iter: 11000     [540/1046]      Loss:  0.4022
2020-06-16 13:40:59,960 Epoch:10        Iter: 11100     [640/1046]      Loss:  0.4003
2020-06-16 13:46:59,414 Epoch:10        Iter: 11200     [740/1046]      Loss:  0.3930
```

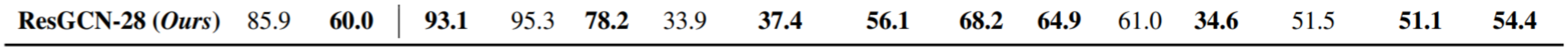
- Checkpoints
  - Trained model with different epochs



```
piaotairen@warhol7:~/PIAO_TAIREN_GCN_Final_DeepGCNs/sem_seg/checkpoints$ ls
_ckpt_0.pth   _ckpt_12.pth  _ckpt_15.pth  _ckpt_18.pth  _ckpt_20.pth  _ckpt_4.pth  _ckpt_7.pth
_ckpt_10.pth  _ckpt_13.pth  _ckpt_16.pth  _ckpt_19.pth  _ckpt_2.pth   _ckpt_5.pth  _ckpt_8.pth
_ckpt_11.pth  _ckpt_14.pth  _ckpt_17.pth  _ckpt_1.pth   _ckpt_3.pth   _ckpt_6.pth  _ckpt_9.pth
```

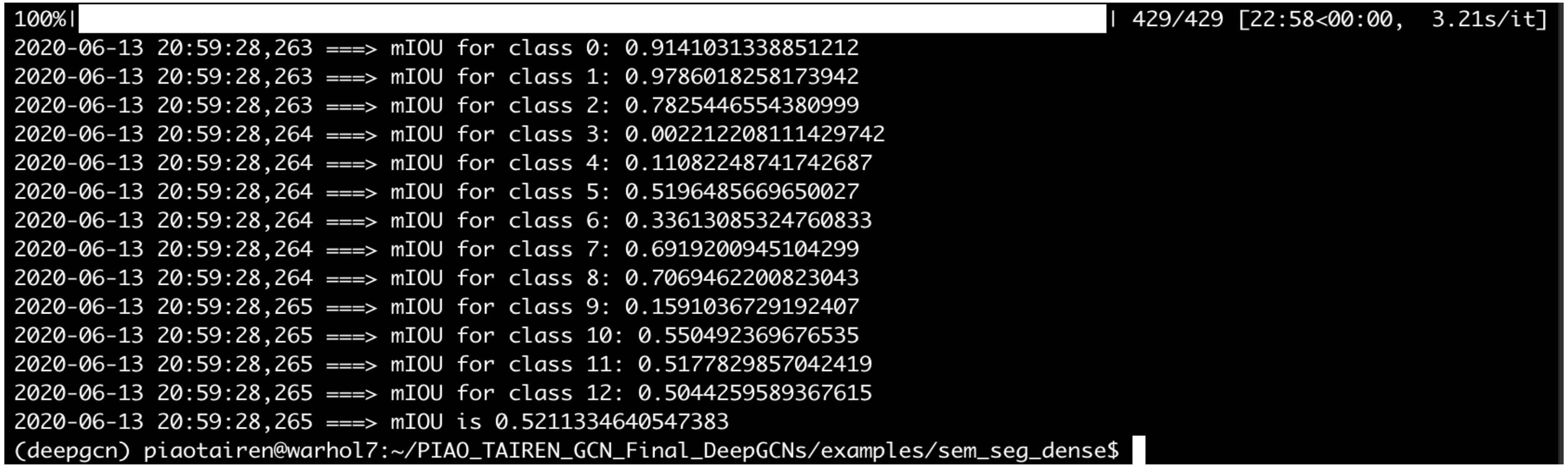
Experiment 2 S3DIS Semantic Segmentation

- Results of the Paper (S3DIS)



ResGCN-28 (*Ours*)	85.9	**60.0**	**93.1**	95.3	**78.2**	33.9	**37.4**	**56.1**	**68.2**	**64.9**	61.0	**34.6**	51.5	**51.1**	**54.4**

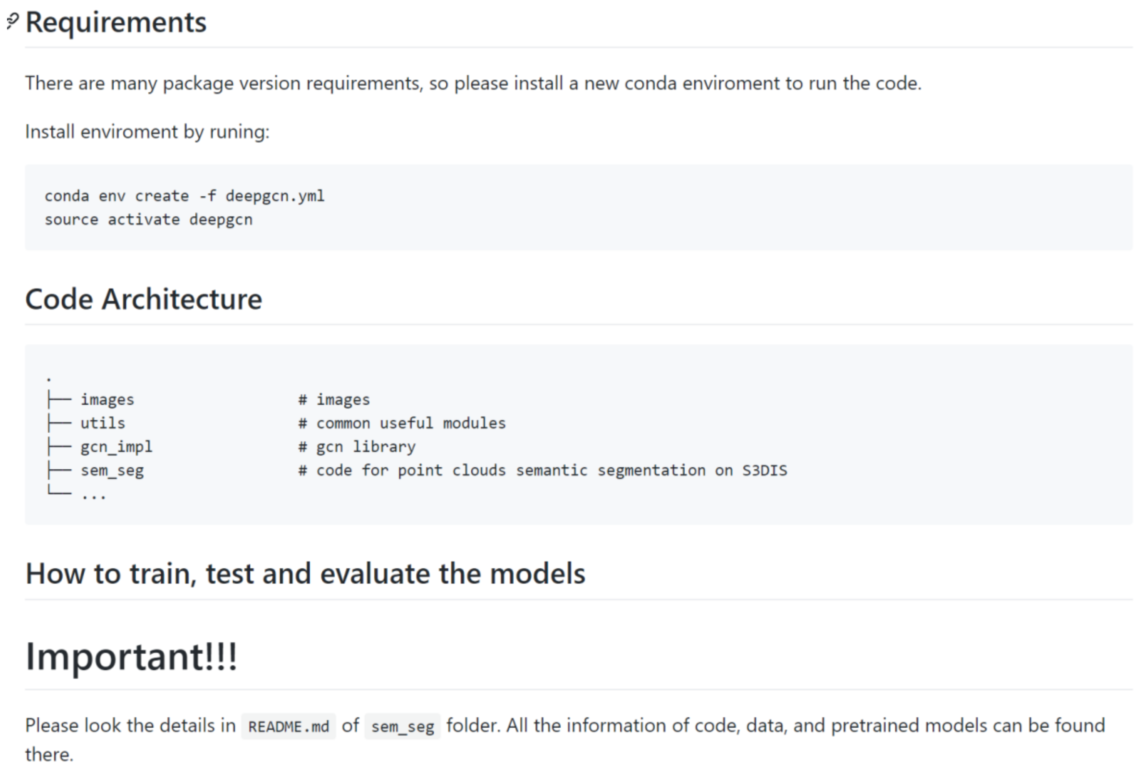
- Results of Reproduction
  - Slightly lower than the results of original paper.
    - But still got 52.1 mIOU score.
    - It means DeepGCNs well trained.



```
100%|████████████████████████████████████████| 429/429 [22:58<00:00,  3.21s/it]
2020-06-13 20:59:28,263 ===> mIOU for class 0: 0.9141031338851212
2020-06-13 20:59:28,263 ===> mIOU for class 1: 0.9786018258173942
2020-06-13 20:59:28,263 ===> mIOU for class 2: 0.7825446554380999
2020-06-13 20:59:28,264 ===> mIOU for class 3: 0.002212208111429742
2020-06-13 20:59:28,264 ===> mIOU for class 4: 0.11082248741742687
2020-06-13 20:59:28,264 ===> mIOU for class 5: 0.5196485669650027
2020-06-13 20:59:28,264 ===> mIOU for class 6: 0.33613085324760833
2020-06-13 20:59:28,264 ===> mIOU for class 7: 0.6919200945104299
2020-06-13 20:59:28,264 ===> mIOU for class 8: 0.7069462200823043
2020-06-13 20:59:28,265 ===> mIOU for class 9: 0.1591036729192407
2020-06-13 20:59:28,265 ===> mIOU for class 10: 0.550492369676535
2020-06-13 20:59:28,265 ===> mIOU for class 11: 0.5177829857042419
2020-06-13 20:59:28,265 ===> mIOU for class 12: 0.5044259589367615
2020-06-13 20:59:28,265 ===> mIOU is 0.5211334640547383
(deepgcn) piaotairen@warhol7:~/PIAO_TAIREN_GCN_Final_DeepGCNs/examples/sem_seg_dense$ 
```

Details about the Code

- I wrote some usage guidelines for my code on the repository.
  - It is easy to run my code by following the instructions.
  - Please check the sem_seg directory for more details about the code.



Requirements

There are many package version requirements, so please install a new conda enviroment to run the code.

Install enviroment by runing:

```
conda env create -f deepgcn.yml
source activate deepgcn
```

Code Architecture

```
.
├── images                  # images
├── utils                   # common useful modules
├── gcn_impl                # gcn library
├── sem_seg                 # code for point clouds semantic segmentation on S3DIS
└── ...
```

How to train, test and evaluate the models

Important!!!

Please look the details in `README.md` of `sem_seg` folder. All the information of code, data, and pretrained models can be found there.

Summary

- I learned much insight into GCN through this course. Its really helped.
- I presented the [paper] about how to train the DeepGCNs by tackle the over-smoothing, vanishing gradient, and over-fitting problems on DeepGCNs.
- I reproduced the paper’s code and uploaded it to my GitHub repository.
- Thank you for listening to my presentation and look at my materials.
- If you are interested in this paper, please run the code and check the results of DeepGCNs.

- Thank you again!